# Oracle® Database Real Application Security Administrator's and Developer's Guide





Oracle Database Real Application Security Administrator's and Developer's Guide, 23c

F47009-02

Copyright © 2012, 2023, Oracle and/or its affiliates.

Primary Author: Gunjan Jain

Contributing Authors: Rod Ward, S. Jeloka, M. Chaliha, T. Das, S. Pelski, R. Leyderman

Contributors: J. Greenberg, S. Adhikari, T. Ahmed, R. Bhatti, C. C. Chui, P. Deshmukh, S. Gul, M. Ho, Y. Hu, S. Jawarikapisha, T. Keefe, P. Knaggs, S. Kwak, H. Li, Y. Li, C. Liang, S. Liu, C. Lei, S. Namuduri, J. Narasinghanallur, G. Narayanan, P. Needham, E. Paapanen, V. Pesati, R. Ramachandra, P. Ramakrishna, D. Raphaely, Y. Ru, J. Samuel, S. Tata, A. Wang, W. Wang, S. Watt, M. Wei, A. Williams, M. Xu, H. Zhang, S. Zhao, S. Zhou

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

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

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

### Contents

### **Preface**

Audience	xxiv
Documentation Accessibility	xxiv
Related Documents	xxiv
Conventions	xxiv

Changes in This Release for Oracle Database Real Application Security Administrator's and Developer's Guide

### 1 Introducing Oracle Database Real Application Security

1.1	What	t Is Oracle Database Real Application Security?	1-1
	1.1.1	Disadvantages of Traditional Security for Managing Application Users	1-2
	1.1.2	Advantages of Real Application Security	1-2
	1.1.3	Architecture of Real Application Security	1-2
1.2	Data	Security Concepts Used in Real Application Security	1-3
	1.2.1	About Data Security with Oracle Database Real Application Security	1-4
	1.2.2	Principals: Users and Roles	1-5
	1.2	2.2.1 Understanding the Difference Between Database Users and Application Users	1-6
	1.2	2.2.2 Understanding the Difference Between Database Roles and Application Roles	1-6
	1.2	2.2.3 Granting Database Privileges to Application Users and Application Roles	1-7
	1.2.3	Application Privileges	1-8
	1.2.4	Security Classes in Oracle Database Real Application Security	1-8
	1.2.5	Access Control Entry (ACE)	1-8
	1.2.6	Access Control List (ACL)	1-8
	1.2.7	Data Security Policy	1-9
1.3	Appli	ication Session Concepts Used in Application Security	1-10
1.4	Flow	of Design and Development	1-10
	1.4.1	Design Phase	1-11
	1.4.2	Development Flow Steps	1-11



1.5 Scenario:	Security Human Resources (HR) Demonstration of Employee Information	
1.5.1 Bas	sic Security HR Demo Scenario: Description and Security Requirements	
1.5.2 Bas	sic HR Scenario: Implementation Overview	
1.6 About Au	diting in an Oracle Database Real Application Security Environment	
1.7 Support f	or Pluggable Databases	
Configuring	Application Users and Application Roles	
2.1 About Co	nfiguring Application Users	
2.1.1 Abo	out Application User Accounts	
2.1.1.1	General Procedures for Creating Application User Accounts	
2.1.2 Cre	ating a Simple Application User Account	
2.1.3 Abo	out Creating a Direct Login Application User Account	
2.1.3.1	Creating Direct Login Application User Accounts	
2.1.3.2	Procedure for Creating the Direct Login Application User Account	
2.1.3.3	Setting a Password Verifier for Direct Application User Accounts	
2.1.3.4	Oracle Label Security Context Is Established in Direct Logon Session	
	setting the Application User's Password with the SQL*Plus PASSWORD mmand	
	nfiguring an Application User Switch	
	idating an Application User	
	nfiguring Application Roles	
	out Application Roles	
	gular and Dynamic Application Roles	
2.2.2.1		
2.2.2.2	Dynamic Application Roles	
2.2.3 Abo	out Configuring an Application Role	
2.2.3.1	Creating a Regular Application Role	
2.2.3.2	Creating a Dynamic Application Role	
2.2.3.3	Validating an Application Role	
2.2.4 Pre	defined Regular Application Roles and Dynamic Application Roles	
2.3 Effective	Dates for Application Users and Application Roles	
2.4 About Gr	anting Application Privileges to Principals	
2.4.1 Abo	out Granting an Application Role to an Application User	
2.4.1.1	Creating a New Application User and Granting This User an Application Role	
2.4.1.2	Granting an Application Role to an Existing Application User	
	unting an Application Role to Another Application Role	
	unting a Database Role to an Application Role	



### 3 Configuring Application Sessions

3.1	Abou	t App	ication Sessions	3-1
	3.1.1	Abou	nt Application Sessions in Real Application Security	3-2
	3.1.2	Adva	intages of Application Sessions	3-3
3.2	Abou	t Crea	ating and Maintaining Application Sessions	3-3
	3.2.1	Crea	ting an Application Session	3-3
	3.2.2	Crea	ting an Anonymous Application Session	3-4
	3.2.3	Attac	ching an Application Session to a Traditional Database Session	3-5
	3.2.4	Setti	ng a Cookie for an Application Session	3-7
	3.2.5	Assi	gning an Application User to an Anonymous Application Session	3-7
	3.2.6		ching a Current Application User to Another Application User in the ent Application Session	3-8
	3.2.7	Abou	it Creating a Global Callback Event Handler Procedure	3-9
	3.2.8	Conf	iguring Global Callback Event Handlers for an Application Session	3-11
	3.2.9	Savii	ng an Application Session	3-12
	3.2.10	Det	aching an Application Session from a Traditional Database Session	3-13
	3.2.11	Des	troying an Application Session	3-14
3.3	Abou	t Man	ipulating the Application Session State	3-15
	3.3.1	Abou	it Using Namespace Templates to Create Namespaces	3-15
	3.3	.1.1	Components of a Namespace Template	3-16
	3.3	.1.2	About Namespace Views	3-17
	3.3	.1.3	Creating a Namespace Template for an Application Session	3-17
	3.3.2	Initia	lizing a Namespace in an Application Session	3-18
	3.3	.2.1	Initializing a Namespace When the Session Is Created	3-18
	3.3	.2.2	Initializing a Namespace When the Session Is Attached	3-19
	3.3	.2.3	Initializing a Namespace When a Named Application User Is Assigned to an Anonymous Application Session	3-20
	3.3	.2.4	Initializing a Namespace When the Application User Is Switched in an	
			Application Session	3-21
		.2.5	Initializing a Namespace Explicitly	3-22
	3.3.3		ng Session Attributes in an Application Session	3-23
	3.3.4		ng Session Attributes in an Application Session	3-24
	3.3.5		ting Custom Attributes in an Application Session	3-25
	3.3.6		ting a Namespace in an Application Session	3-25
	3.3.7		ling Application Roles for a Session	3-26
	3.3.8		oling Application Roles for a Session	3-27
3.4			inistrative APIs for External Users and Roles	3-27
3.5			Application Security Session Privilege Scoping Through ACL	3-28
	251	Gran	ting Session Privileges on a Principal Hsing an ΔCI	3-31



### 4 Configuring Application Privileges and Access Control Lists

4.1 About Application Privileges	4-1
4.1.1 Aggregate Privilege	4-1
4.1.1.1 ALL Privilege	4-3
4.2 About Configuring Security Classes	4-3
4.2.1 About Security Classes	4-3
4.2.2 Security Class Inheritance	4-4
4.2.3 Security Class as Privilege Scope	4-5
4.2.4 DML Security Class	4-5
4.2.5 About Validating Security Classes	4-5
4.2.6 Manipulating Security Classes	4-5
4.3 About Configuring Access Control Lists	4-7
4.3.1 About ACLs and ACEs	4-7
4.3.2 Creating ACLs and ACEs	4-8
4.3.2.1 Deny	4-9
4.3.2.2 Invert	4-10
4.3.2.3 ACE Start-Date and End-Date	4-10
4.3.3 About Validating Access Control Lists	4-10
4.3.4 Updating Access Control Lists	4-10
4.3.5 About Checking ACLs for a Privilege	4-12
4.3.6 About Using Multilevel Authentication	4-12
4.3.7 Principal Types	4-13
4.3.8 Access Resolution Results	4-13
4.3.9 ACE Evaluation Order	4-13
4.3.10 ACL Inheritance	4-13
4.3.10.1 Extending ACL Inheritance	4-13
4.3.10.2 Constraining ACL Inheritance	4-14
4.3.11 About ACL Catalog Views	4-15
4.3.12 About Security Class Catalog Views	4-15
4.4 Data Security	4-15
4.4.1 Data Realms	4-15
4.4.2 Parameterized ACL	4-16
4.5 ACL Binding	4-16
Configuring Data Security	
5.1 About Data Security	5-1
5.2 About Validating the Data Security Policy	5-2
5.3 Understanding the Structure of the Data Security Policy	5-2
5.4 About Designing Data Realms	5-4
5.4.1 About Understanding the Structure of a Data Realm	5-5



5

	5.4.2	About Using Static Data Realms	5-6
	5.4.3	Using Trace Files to Check for Policy Predicate Errors	5-7
	5.5 Applyi	ing Additional Application Privileges to a Column	5-8
	5.6 About	Enabling Data Security Policy for a Database Table or View	5-10
		Enabling Real Application Security Using the APPLY_OBJECT_POLICY Procedure	5-10
		1.1 About Applying Multiple Policies for a Table or View	5-11
		About How the APPLY OBJECT POLICY Procedure Alters a Database Table	5-11
		About How ACLs on Table Data Are Evaluated	5-12
		Creating Real Application Security Policies on Master-Detail Related Tables	5-12
		About Real Application Security Policies on Master-Detail Related Tables	5-12
		About Understanding the Structure of Master Detail Data Realms	5-13
		Example of Creating a Real Application Security Policy on Master-Detail	0 _0
		Related Tables	5-13
	5.8 About	Managing Application Privileges for Data Security Policies	5-21
	5.8.1	About Bypassing the Security Checks of a Real Application Security Policy	5-21
	5.8.2	Using the SQL*Plus SET SECUREDCOL Command	5-22
	5.9 Using	BEQUEATH CURRENT_USER Views	5-23
	5.9.1	Using SQL Functions to Determine the Invoking Application User	5-25
	5.10 Real	Application Security: Putting It All Together	5-26
	5.10.1	Basic HR Scenario: Implementation Tasks	5-26
	5.10	0.1.1 Connecting as User SYS to Create Real Application Security Users and Roles	5-27
	5.10	0.1.2 Creating Roles and Application Users	5-27
	5.10	0.1.3 Creating the Security Class and ACLS	5-29
	5.10	0.1.4 Creating the Data Security Policy	5-31
	5.10	0.1.5 Validating the Real Application Security Objects	5-32
	5.10	0.1.6 Disabling a Data Security Policy for a Table	5-32
	5.10.2	Running the Security HR Demo	5-33
	5.11 Abou	ut Schema Level Real Application Security Policy Administration	5-33
	5.11.1	Setting Up and Enabling a Schema Level Data Security Policy	5-35
		Disabling the Data Security Policy and Revoking the System Privileges from the User	5-36
6	Using Re	eal Application Security in Java Applications	
	6.1 About	: Initializing the Middle Tier	6-1
	6.1.1	About Mid-Tier Configuration Mode	6-1
	6.1.2	Using the getSessionManager Method	6-1
	6.1.3	About Changing the Middle-Tier Cache Setting	6-3
	6.1.3	3.1 About Setting the Maximum Cache Idle Time	6-3
	6.1.3	3.2 About Setting the Maximum Cache Size	6-3



	0.1	L.S.S	About Getting the Maximum Cache fulle Time	0-3
	6.1	L.3.4	About Getting the Maximum Cache Size	6-3
	6.1	L.3.5	About Removing Entries from the Cache	6-4
	6.1	L.3.6	About Clearing the Cache	6-4
6.2	2 Abou	ut Mar	naging Real Application Security Sessions	6-4
	6.2.1	Crea	ating a Real Application Security User Session	6-5
	6.2.2	Atta	ching an Application Session	6-5
	6.2.3	Assi	gning or Switching an Application User	6-6
	6.2.4	Enal	oling Real Application Security Application Roles	6-7
	6.2	2.4.1	Enabling a Real Application Security Application Role	6-7
	6.2	2.4.2	Disabling a Real Application Security Application Role	6-7
	6.2	2.4.3	Checking If a Real Application Security Application Role Is Enabled	6-8
	6.2.5	Abo	ut Performing Namespace Operations as Session User	6-8
	6.2	2.5.1	Creating Namespaces	6-8
	6.2	2.5.2	Deleting Namespaces	6-9
	6.2	2.5.3	Implicitly Creating Namespaces	6-9
	6.2	2.5.4	About Using Namespace Attributes	6-9
	6.2.6	Abo	ut Performing Namespace Operations as Session Manager	6-11
	6.2.7	Abo	ut Performing Miscellaneous Session-Related Activities	6-11
	6.2	2.7.1	About Getting the Oracle Connection Associated with the Session	6-12
	6.2	2.7.2	About Getting the Application User ID for the Session	6-12
	6.2	2.7.3	Getting the Session ID for the Session	6-12
	6.2	2.7.4	About Getting a String Representation of the Session	6-12
	6.2	2.7.5	Getting the Session Cookie	6-12
	6.2	2.7.6	Setting Session Inactivity Timeout as Session Manager	6-12
	6.2	2.7.7	Setting the Session Cookie as Session Manager	6-13
	6.2.8	Deta	aching an Application Session	6-13
	6.2.9	Dest	troying A Real Application Security Application Session	6-13
6.3	3 Auth	entica	ating Application Users Using Java APIs	6-14
6.4	4 Abou	ut Autl	norizing Application Users Using ACLs	6-14
	6.4.1	Con	structing an ACL Identifier	6-14
	6.4.2	Usin	g the checkAcl Method	6-15
	6.4.3	Abo	ut Getting Data Privileges Associated with a Specific ACL	6-15
6.5	5 Hum	an Re	esources Administration Use Case: Implementation in Java	6-15
	Out	put		6-20
0	racle I	Fusi	on Middleware Integration with Real Application Security	
7.1	L Abou	ut Exte	ernal Users and External Roles	7-1
7.2	2 Sess	sion A	PIs for External Users and Roles	7-2
	7.2.1	Nam	nespace for External Users	7-2



7

	7.2.2 Creating a Session	7-2
	7.2.3 Attaching a Session	7-4
	7.2.4 Assigning a User to a Session	7-7
	7.2.5 Saving a Session and Terminating a Session	7-9
8	Application Session Service in Oracle Fusion Middleware	
	8.1 About Real Application Security Concepts	8-1
	8.2 About Application Session Service in Oracle Fusion Middleware	8-3
	8.3 About the Application Session Filter	8-5
	8.3.1 About the Application Session Filter Operation	8-5
	8.4 About Deployment	8-6
	8.5 About Application Configuration of the Application Session Filter	8-7
	8.6 Domain Configuration: Setting Up an Application Session Service to Work with OPSS and Oracle Fusion Middleware	8-8
	8.6.1 Prerequisites	8-8
	8.6.2 Manual Configuration	8-9
	8.6.3 About Automatic Configuration	8-10
	8.7 About Application Session APIs	8-10
	8.7.1 About Application Session APIs	8-11
	8.7.1.1 About Attaching to an Application Session	8-11
	8.7.1.2 Detaching from an Application Session	8-11
	8.7.1.3 Destroying an Application Session	8-12
	8.7.2 About the Privilege Elevation API	8-13
	8.7.2.1 Enabling a Dynamic Role in the Application Session	8-14
	8.7.3 About Namespace APIs	8-15
	8.7.3.1 About Creating a Namespace	8-15
	8.7.3.2 About Deleting a Namespace	8-16
	8.7.3.3 About Setting the Namespace Attribute	8-16
	8.7.3.4 About Deleting a Namespace Attribute	8-17
	8.7.3.5 Getting a Namespace Attribute	8-17
	8.7.4 About the Check Privilege API	8-19
	8.7.4.1 Checking a Privilege on the ACLs	8-19
	8.8 Human Resources Demo Use Case: Implementation in Java	8-21
	8.8.1 Setting Up the HR Demo Application for External Principals (setup.sql)	8-22
	8.8.2 About the Application Session Filter Configuration File (web.xml)	8-26
	8.8.3 About the Sample Servlet Application (MyHR.java)	8-29
	8.8.4 About the Filter to Set Up the Application Namespace (MyFilter.java)	8-35
	8.8.5 About the HR Demo Use Case - User Roles	8-38
	8.8.6 About the HR Demo (1) - Logged in as Employee LPOPP	8-38
	8.8.7 About the HR Demo (2) - Logged in as HRMGR	8-39



### 9 Oracle Database Real Application Security Data Dictionary Views

9.1	DBA_XS_OBJECTS	9-4
9.2	DBA_XS_PRINCIPALS	9-5
9.3	DBA_XS_EXTERNAL_PRINCIPALS	9-5
9.4	DBA_XS_USERS	9-6
9.5	USER_XS_USERS	9-7
9.6	USER_XS_PASSWORD_LIMITS	9-8
9.7	DBA_XS_ROLES	9-8
9.8	DBA_XS_DYNAMIC_ROLES	9-9
9.9	DBA_XS_PROXY_ROLES	9-9
9.10	DBA_XS_ROLE_GRANTS	9-10
9.11	DBA_XS_PRIVILEGES	9-10
9.12	USER_XS_PRIVILEGES	9-11
9.13	ALL_XS_PRIVILEGES	9-11
9.14	DBA_XS_IMPLIED_PRIVILEGES	9-12
9.15	USER_XS_IMPLIED_PRIVILEGES	9-12
9.16	ALL_XS_IMPLIED_PRIVILEGES	9-13
9.17	DBA_XS_PRIVILEGE_GRANTS	9-13
9.18	DBA_XS_SECURITY_CLASSES	9-14
9.19	USER_XS_SECURITY_CLASSES	9-14
9.20	ALL_XS_SECURITY_CLASSES	9-15
9.21	DBA_XS_SECURITY_CLASS_DEP	9-15
9.22	USER_XS_SECURITY_CLASS_DEP	9-15
9.23	ALL_XS_SECURITY_CLASS_DEP	9-16
9.24	DBA_XS_ACLS	9-16
9.25	USER_XS_ACLS	9-17
9.26	ALL_XS_ACLS	9-17
9.27	DBA_XS_ACES	9-18
9.28	USER_XS_ACES	9-19
9.29	ALL_XS_ACES	9-19
9.30	DBA_XS_POLICIES	9-20
9.31	USER_XS_POLICIES	9-21
9.32	ALL_XS_POLICIES	9-21
9.33	DBA_XS_REALM_CONSTRAINTS	9-21
9.34	USER_XS_REALM_CONSTRAINTS	9-22
9.35	ALL_XS_REALM_CONSTRAINTS	9-23
9.36	DBA_XS_INHERITED_REALMS	9-24
9.37	USER_XS_INHERITED_REALMS	9-24



	9.38	ALL_XS_INHERITED_REALMS	9-25
	9.39	DBA_XS_ACL_PARAMETERS	9-25
	9.40	USER_XS_ACL_PARAMETERS	9-26
	9.41	ALL_XS_ACL_PARAMETERS	9-27
	9.42	DBA_XS_COLUMN_CONSTRAINTS	9-27
	9.43	USER_XS_COLUMN_CONSTRAINTS	9-28
	9.44	ALL_XS_COLUMN_CONSTRAINTS	9-28
	9.45	DBA_XS_APPLIED_POLICIES	9-28
	9.46	ALL_XS_APPLIED_POLICIES	9-29
	9.47	DBA_XS_MODIFIED_POLICIES	9-30
	9.48	DBA_XS_SESSIONS	9-30
	9.49	DBA_XS_ACTIVE_SESSIONS	9-31
	9.50	DBA_XS_SESSION_ROLES	9-31
	9.51	DBA_XS_SESSION_NS_ATTRIBUTES	9-32
	9.52	DBA_XS_NS_TEMPLATES	9-32
	9.53	DBA_XS_NS_TEMPLATE_ATTRIBUTES	9-33
	9.54	ALL_XDS_ACL_REFRESH	9-34
	9.55	ALL_XDS_ACL_REFSTAT	9-34
	9.56	ALL_XDS_LATEST_ACL_REFSTAT	9-35
	9.57	DBA_XDS_ACL_REFRESH	9-36
	9.58	DBA_XDS_ACL_REFSTAT	9-36
	9.59	DBA_XDS_LATEST_ACL_REFSTAT	9-37
	9.60	USER_XDS_ACL_REFRESH	9-38
	9.61	USER_XDS_ACL_REFSTAT	9-38
	9.62	USER_XDS_LATEST_ACL_REFSTAT	9-39
	9.63	V\$XS_SESSION_NS_ATTRIBUTES	9-40
	9.64	V\$XS_SESSION_ROLES	9-40
10	Ora	cle Database Real Application Security SQL Functions	
	10.1	COLUMN_AUTH_INDICATOR Function	10-1
	10.2	XS SYS CONTEXT Function	10-2
	10.3	ORA_CHECK_ACL Function	10-4
	10.4	ORA_GET_ACLIDS Function	10-4
	10.5	ORA_CHECK_PRIVILEGE Function	10-5
	10.6	TO_ACLID Function	10-6
11	Ora	cle Database Real Application Security PL/SQL Packages	
	11.1	DBMS_XS_SESSIONS Package	11-1
	1	1.1.1 Security Model	11-2



11.1.2 Cons	stants	11-2
11.1.3 Obje	ect Types, Constructor Functions, Synonyms, and Grants	11-2
11.1.4 Sum	mary of DBMS_XS_SESSIONS Subprograms	11-3
11.1.4.1	CREATE_SESSION Procedure	11-4
11.1.4.2	ATTACH_SESSION Procedure	11-5
11.1.4.3	ASSIGN_USER Procedure	11-7
11.1.4.4	SWITCH_USER Procedure	11-8
11.1.4.5	CREATE_NAMESPACE Procedure	11-9
11.1.4.6	CREATE_ATTRIBUTE Procedure	11-10
11.1.4.7	SET_ATTRIBUTE Procedure	11-11
11.1.4.8	GET_ATTRIBUTE Procedure	11-12
11.1.4.9	RESET_ATTRIBUTE Procedure	11-13
11.1.4.10	DELETE_ATTRIBUTE Procedure	11-14
11.1.4.11	DELETE_NAMESPACE Procedure	11-14
11.1.4.12	ENABLE_ROLE Procedure	11-15
11.1.4.13	DISABLE_ROLE Procedure	11-16
11.1.4.14	SET_SESSION_COOKIE Procedure	11-16
11.1.4.15	REAUTH_SESSION Procedure	11-17
11.1.4.16	SET_INACTIVITY_TIMEOUT Procedure	11-18
11.1.4.17	SAVE_SESSION Procedure	11-19
11.1.4.18	DETACH_SESSION Procedure	11-19
11.1.4.19	DESTROY_SESSION Procedure	11-19
11.1.4.20	ADD_GLOBAL_CALLBACK Procedure	11-20
11.1.4.21	ENABLE_GLOBAL_CALLBACK Procedure	11-21
11.1.4.22	DELETE_GLOBAL_CALLBACK Procedure	11-22
11.2 XS_ACL P	Package	11-23
11.2.1 Secu	urity Model for the XS_ACL Package	11-23
11.2.2 Cons	stants	11-24
11.2.3 Obje	ect Types, Constructor Functions, Synonyms, and Grants	11-24
11.2.4 Sum	mary of XS_ACL Subprograms	11-25
11.2.4.1	CREATE_ACL Procedure	11-25
11.2.4.2	APPEND_ACES Procedure	11-26
11.2.4.3	REMOVE_ACES Procedure	11-27
11.2.4.4	SET_SECURITY_CLASS Procedure	11-28
11.2.4.5	SET_PARENT_ACL Procedure	11-28
11.2.4.6	ADD_ACL_PARAMETER Procedure	11-29
11.2.4.7	REMOVE_ACL_PARAMETERS Procedure	11-30
11.2.4.8	SET_DESCRIPTION Procedure	11-31
11.2.4.9	DELETE_ACL Procedure	11-31
11.3 XS_ADMIN	N_UTIL Package	11-32
11.3.1 Secu	urity Model	11-32



11.	.3.2 (	Constants	11-32
11.	.3.3 (	Object Types, Constructor Functions, Synonyms, and Grants	11-33
11.	.3.4	Summary of XS_ADMIN_UTIL Subprograms	11-33
	11.3.4	4.1 GRANT_SYSTEM_PRIVILEGE Procedure	11-33
	11.3.4	1.2 REVOKE_SYSTEM_PRIVILEGE Procedure	11-34
11.4	XS_DA	ATA_SECURITY Package	11-35
11.	.4.1	Security Model for the XS_DATA_SECURITY Package	11-35
11.	.4.2	Object Types, Constructor Functions, Synonyms, and Grants	11-36
11.	.4.3	Summary of XS_DATA_SECURITY Subprograms	11-38
	11.4.3	3.1 CREATE_POLICY Procedure	11-38
	11.4.3	3.2 APPEND_REALM_CONSTRAINTS Procedure	11-39
	11.4.3	3.3 REMOVE_REALM_CONSTRAINTS Procedure	11-40
	11.4.3	3.4 ADD_COLUMN_CONSTRAINTS Procedure	11-41
	11.4.3	3.5 REMOVE_COLUMN_CONSTRAINTS Procedure	11-42
	11.4.3	3.6 CREATE_ACL_PARAMETER Procedure	11-42
	11.4.3	3.7 DELETE_ACL_PARAMETER Procedure	11-43
	11.4.3	3.8 SET_DESCRIPTION Procedure	11-44
	11.4.3	3.9 DELETE_POLICY Procedure	11-44
	11.4.3	3.10 ENABLE_OBJECT_POLICY Procedure	11-45
	11.4.3	3.11 DISABLE_OBJECT_POLICY Procedure	11-46
	11.4.3	3.12 REMOVE_OBJECT_POLICY Procedure	11-47
	11.4.3	3.13 APPLY_OBJECT_POLICY Procedure	11-48
11.5	XS_DA	ATA_SECURITY_UTIL Package	11-49
11.	.5.1	Security Model	11-49
11.	.5.2	Constants	11-49
11.	.5.3	Summary of XS_DATA_SECURITY_UTIL Subprograms	11-50
	11.5.3	3.1 SCHEDULE_STATIC_ACL_REFRESH Procedure	11-50
	11.5.3	3.2 ALTER_STATIC_ACL_REFRESH Procedure	11-51
11.6	XS_DI	AG Package	11-51
11.	.6.1	Security Model	11-52
11.	.6.2	Summary of XS_DIAG Subprograms	11-52
	11.6.2	2.1 VALIDATE_PRINCIPAL Function	11-52
	11.6.2	2.2 VALIDATE_SECURITY_CLASS Function	11-53
	11.6.2	2.3 VALIDATE_ACL Function	11-54
	11.6.2	2.4 VALIDATE_DATA_SECURITY Function	11-55
	11.6.2	2.5 VALIDATE_NAMESPACE_TEMPLATE Function	11-56
	11.6.2	2.6 VALIDATE_WORKSPACE Function	11-57
11.7	XS_NA	AMESPACE Package	11-57
11.	.7.1	Security Model	11-58
11.	.7.2	Constants	11-58
11.	.7.3	Object Types, Constructor Functions, Synonyms, and Grants	11-58



11.7.4 St	ummary of XS_NAMESPACE Subprograms	11-59
11.7.4.	1 CREATE_TEMPLATE Procedure	11-59
11.7.4.	2 ADD_ATTRIBUTES Procedure	11-60
11.7.4.	3 REMOVE_ATTRIBUTES Procedure	11-61
11.7.4.	4 SET_HANDLER Procedure	11-61
11.7.4.	5 SET_DESCRIPTION Procedure	11-62
11.7.4.	6 DELETE_TEMPLATE Procedure	11-62
11.8 XS_PRI	NCIPAL Package	11-63
11.8.1 Se	ecurity Model	11-63
11.8.2 C	onstants	11-64
11.8.3 O	bject Types, Constructor Functions, Synonyms, and Grants	11-64
11.8.4 St	ummary of XS_PRINCIPAL Subprograms	11-65
11.8.4.	1 CREATE_USER Procedure	11-66
11.8.4.	2 CREATE_ROLE Procedure	11-67
11.8.4.	3 CREATE_DYNAMIC_ROLE Procedure	11-68
11.8.4.	4 GRANT_ROLES Procedure	11-69
11.8.4.	5 REVOKE_ROLES Procedure	11-71
11.8.4.	6 ADD_PROXY_USER Procedure	11-71
11.8.4.	7 REMOVE_PROXY_USERS Procedure	11-73
11.8.4.	8 ADD_PROXY_TO_DBUSER	11-73
11.8.4.	9 REMOVE_PROXY_FROM_DBUSER Procedure	11-74
11.8.4.	10 SET_EFFECTIVE_DATES Procedure	11-74
11.8.4.	11 SET_DYNAMIC_ROLE_DURATION Procedure	11-75
11.8.4.	12 SET_DYNAMIC_ROLE_SCOPE Procedure	11-76
11.8.4.	13 ENABLE_BY_DEFAULT Procedure	11-76
11.8.4.	14 ENABLE_ROLES_BY_DEFAULT Procedure	11-77
11.8.4.	15 SET_USER_SCHEMA Procedure	11-78
11.8.4.	16 SET_GUID Procedure	11-78
11.8.4.	17 SET_ACL Procedure	11-79
11.8.4.	18 SET_PROFILE Procedure	11-80
11.8.4.	19 SET_USER_STATUS Procedure	11-81
11.8.4.	20 SET_PASSWORD Procedure	11-82
11.8.4.	21 SET_VERIFIER Procedure	11-83
11.8.4.	22 SET_DESCRIPTION Procedure	11-85
11.8.4.	23 DELETE_PRINCIPAL Procedure	11-85
11.9 XS_SE	CURITY_CLASS Package	11-86
11.9.1 Se	ecurity Model for the XS_SECURITY_CLASS Package	11-86
11.9.2 St	ummary of XS_SECURITY_CLASS Subprograms	11-87
11.9.2.	1 CREATE_SECURITY_CLASS Procedure	11-87
11.9.2.	2 ADD_PARENTS Procedure	11-88
11.9.2.	3 REMOVE_PARENTS Procedure	11-89



11.9.2.4 A	DD_PRIVILEGES Procedure	11-89
11.9.2.5 F	REMOVE_PRIVILEGES Procedure	11-90
11.9.2.6 A	DD_IMPLIED_PRIVILEGES Procedure	11-91
11.9.2.7 F	REMOVE_IMPLIED_PRIVILEGES Procedure	11-92
11.9.2.8 S	SET_DESCRIPTION Procedure	11-93
11.9.2.9 D	DELETE_SECURITY_CLASS Procedure	11-93
Real Application	on Security HR Demo	
12.1 Overview of t	the Security HR Demo	12-1
12.2 What Each S	cript Does	12-2
12.3 Setting Up th	e Security HR Demo Components	12-4
12.3.1 About 0	Creating Roles and Application Users	12-4
12.3.2 About 0	Creating the Security Class and ACLs	12-5
12.3.3 About 0	Creating the Data Security Policy	12-6
12.3.4 About \	Validating the Real Application Security Objects	12-7
12.3.5 About 9	Setting Up the Mid-Tier Related Configuration	12-7
12.4 Running the	Security HR Demo Using Direct Logon	12-8
10 F D	Security HR Demo Attached to a Real Application Security Session	12-10
12.5 Running the	Coounty The Dome / Maconea to a recall / Spinoalien Coounty Cooper	
· ·	Security HR Demo Cleanup Script	12-12
12.6 Running the 12.7 Running the	Security HR Demo Cleanup Script Security HR Demo in the Java Interface	12-12 12-13
12.6 Running the 12.7 Running the	Security HR Demo Cleanup Script	
12.6 Running the 12.7 Running the Predefined Ob	Security HR Demo Cleanup Script Security HR Demo in the Java Interface	12-13
12.6 Running the 12.7 Running the  Predefined Ob  A.1 Users A.2 Roles	Security HR Demo Cleanup Script Security HR Demo in the Java Interface	12-13 A-1
12.6 Running the 12.7 Running the Predefined Ob  A.1 Users A.2 Roles A.2.1 Regular	Security HR Demo Cleanup Script Security HR Demo in the Java Interface jects in Real Application Security	12-13 A-1 A-1
12.6 Running the 12.7 Running the Predefined Ob  A.1 Users A.2 Roles A.2.1 Regular	Security HR Demo Cleanup Script Security HR Demo in the Java Interface  jects in Real Application Security  Application Roles C Application Roles	12-13 A-1 A-1
12.6 Running the 12.7 Running the Predefined Ob  A.1 Users A.2 Roles A.2.1 Regular A.2.2 Dynamic	Security HR Demo Cleanup Script Security HR Demo in the Java Interface  jects in Real Application Security  Application Roles C Application Roles	A-1 A-1 A-1 A-2
12.6 Running the 12.7 Running the Predefined Ob  A.1 Users A.2 Roles A.2.1 Regular A.2.2 Dynamic A.2.3 Databas	Security HR Demo Cleanup Script Security HR Demo in the Java Interface  jects in Real Application Security  Application Roles C Application Roles See Roles	A-1 A-1 A-2 A-2
12.6 Running the 12.7 Running the Predefined Ob  A.1 Users A.2 Roles A.2.1 Regular A.2.2 Dynamic A.2.3 Databas  A.3 Namespaces	Security HR Demo Cleanup Script Security HR Demo in the Java Interface  jects in Real Application Security  Application Roles C Application Roles See Roles	A-1 A-1 A-1 A-2 A-2 A-3
12.6 Running the 12.7 Running the Predefined Ob  A.1 Users A.2 Roles A.2.1 Regular A.2.2 Dynamic A.2.3 Databas A.3 Namespaces A.4 Security Class A.5 ACLs	Security HR Demo Cleanup Script Security HR Demo in the Java Interface  jects in Real Application Security  Application Roles C Application Roles See Roles	A-1 A-1 A-2 A-3 A-3 A-4
12.6 Running the 12.7 Running the Predefined Ob A.1 Users A.2 Roles A.2.1 Regular A.2.2 Dynamic A.2.3 Databas A.3 Namespaces A.4 Security Class A.5 ACLs  Configuring Occurrences	Security HR Demo Cleanup Script Security HR Demo in the Java Interface  jects in Real Application Security  Application Roles C Application Roles See Roles  Sees	A-1 A-1 A-2 A-3 A-3 A-4
12.6 Running the 12.7 Running the Predefined Ob A.1 Users A.2 Roles A.2.1 Regular A.2.2 Dynamic A.2.3 Databas A.3 Namespaces A.4 Security Class A.5 ACLs  Configuring OC B.1 About Using Compacts	Security HR Demo Cleanup Script Security HR Demo in the Java Interface  jects in Real Application Security  Application Roles C Application Roles See Roles  Sees  CI and JDBC Applications for Column Authorization	A-1 A-1 A-2 A-3 A-4
12.6 Running the 12.7 Running the Predefined Ob A.1 Users A.2 Roles A.2.1 Regular A.2.2 Dynamic A.2.3 Databas A.3 Namespaces A.4 Security Class A.5 ACLs  Configuring OC B.1 About Using Constitution of the security Class	Security HR Demo Cleanup Script Security HR Demo in the Java Interface  jects in Real Application Security  Application Roles Application Roles See Roles  CI and JDBC Applications for Column Authorization  OCI to Retrieve Column Authorization Indicators	A-1 A-1 A-2 A-3 A-4
12.6 Running the 12.7 Running the Predefined Ob A.1 Users A.2 Roles A.2.1 Regular A.2.2 Dynamic A.2.3 Databas A.3 Namespaces A.4 Security Class A.5 ACLs  Configuring Oc B.1 About Using C B.1.1 Example B.1.2 About U	Security HR Demo Cleanup Script Security HR Demo in the Java Interface  jects in Real Application Security  Application Roles C Application Roles See Roles  CI and JDBC Applications for Column Authorization  OCI to Retrieve Column Authorization Indicators e of Obtaining the Return Code	A-1 A-1 A-2 A-3 A-4 I B-1
Predefined Ob A.1 Users A.2 Roles A.2.1 Regular A.2.2 Dynamic A.2.3 Databas A.3 Namespaces A.4 Security Class A.5 ACLs  Configuring OC B.1 About Using C B.1.1 Example B.1.2 About U B.1.3 About the	Security HR Demo Cleanup Script Security HR Demo in the Java Interface  jects in Real Application Security  Application Roles C Application Roles See Roles  CI and JDBC Applications for Column Authorization  OCI to Retrieve Column Authorization Indicators See of Obtaining the Return Code Sing the Return Code and Indicator with Authorization Indicator	A-1 A-1 A-2 A-3 A-4 B-1 B-1 B-2
12.6 Running the 12.7 Running the Predefined Ob A.1 Users A.2 Roles A.2.1 Regular A.2.2 Dynamic A.2.3 Databas A.3 Namespaces A.4 Security Class A.5 ACLs  Configuring Oc B.1 About Using C B.1.1 Example B.1.2 About U B.1.3 About th B.1.4 Using Oc B.1.5 Running the B.1.6 Running the B.1.6 Running the B.1.7 Running the B.1.8 Running the B.1.8 Running the B.1.9 R	Security HR Demo Cleanup Script Security HR Demo in the Java Interface  jects in Real Application Security  Application Roles c Application Roles see Roles  CI and JDBC Applications for Column Authorization  OCI to Retrieve Column Authorization Indicators e of Obtaining the Return Code using the Return Code and Indicator with Authorization Indicator ne Warning for Unknown Authorization Indicator	A-1 A-1 A-2 A-3 A-4 B-1 B-1 B-2 B-3



B.2.2	About Checking User Authorization for a Table Column	
B.2.3	Example of Checking Security Attributes and User Authorization	
Real Ap	plication Security HR Demo Files	
C.1 How	to Run the Security HR Demo	
C.2 Scri	ots for the Security HR Demo	
C.2.1	hrdemo_setup.sql	
C.2.2	hrdemo.sql	
C.2.3	hrdemo_session.sql	
C.2.4	hrdemo.java	
C.2.5	hrdemo_clean.sql	
C.3 Gen	erated Log Files for Each Script	
C.3.1	hrdemo_setup.log	
C.3.2	hrdemo.log	
C.3.3	hrdemo_run_sess.log	
C.3.4	hrdemo.log	
C.3.5	hrdemo_clean.log	
D.1 Abo	ut Real Application Security Diagnostics	
D.1.1	About Using Validation APIs	
D.1.2	About Osing Validation APIS	
0.1.	How to Check Which ACLs Are Associated with a Row for the Current User	
D.1.3		
	How to Check Which ACLs Are Associated with a Row for the Current User	
D.1.3	How to Check Which ACLs Are Associated with a Row for the Current User How to Find If a Privilege Is Granted in an ACL to a User	
D.1.3 D.1.4	How to Check Which ACLs Are Associated with a Row for the Current User How to Find If a Privilege Is Granted in an ACL to a User About Exception State Dumps	
D.1.3 D.1.4 D.1.5 D.1.6	How to Check Which ACLs Are Associated with a Row for the Current User How to Find If a Privilege Is Granted in an ACL to a User About Exception State Dumps About Event-Based Tracing	
D.1.3 D.1.4 D.1.5 D.1.6 D.1.7	How to Check Which ACLs Are Associated with a Row for the Current User How to Find If a Privilege Is Granted in an ACL to a User About Exception State Dumps About Event-Based Tracing About In-Memory Tracing	
D.1.3 D.1.4 D.1.5 D.1.6 D.1.7	How to Check Which ACLs Are Associated with a Row for the Current User How to Find If a Privilege Is Granted in an ACL to a User About Exception State Dumps About Event-Based Tracing About In-Memory Tracing About Statistics	
D.1.3 D.1.4 D.1.5 D.1.6 D.1.7	How to Check Which ACLs Are Associated with a Row for the Current User How to Find If a Privilege Is Granted in an ACL to a User About Exception State Dumps About Event-Based Tracing About In-Memory Tracing About Statistics ut Event-Based Tracing of Real Application Security Components	
D.1.3 D.1.4 D.1.5 D.1.6 D.1.7 D.2 Abo D.2.1	How to Check Which ACLs Are Associated with a Row for the Current User How to Find If a Privilege Is Granted in an ACL to a User About Exception State Dumps About Event-Based Tracing About In-Memory Tracing About Statistics ut Event-Based Tracing of Real Application Security Components About Application Sessions (XSSESSION) Event-Based Tracing	
D.1.3 D.1.4 D.1.5 D.1.6 D.1.7 D.2 Abor D.2.1 D.2.2	How to Check Which ACLs Are Associated with a Row for the Current User How to Find If a Privilege Is Granted in an ACL to a User About Exception State Dumps About Event-Based Tracing About In-Memory Tracing About Statistics at Event-Based Tracing of Real Application Security Components About Application Sessions (XSSESSION) Event-Based Tracing About Application Principals (XSPRINCIPAL) Event-Based Tracing	
D.1.3 D.1.4 D.1.5 D.1.6 D.1.7 D.2 Abor D.2.1 D.2.2 D.2.3	How to Check Which ACLs Are Associated with a Row for the Current User How to Find If a Privilege Is Granted in an ACL to a User About Exception State Dumps About Event-Based Tracing About In-Memory Tracing About Statistics at Event-Based Tracing of Real Application Security Components About Application Sessions (XSSESSION) Event-Based Tracing About Application Principals (XSPRINCIPAL) Event-Based Tracing About Security Classes (XSSECCLASS) Event-Based Tracing	
D.1.3 D.1.4 D.1.5 D.1.6 D.1.7 D.2 Abor D.2.1 D.2.2 D.2.3 D.2.4 D.2.5	How to Check Which ACLs Are Associated with a Row for the Current User How to Find If a Privilege Is Granted in an ACL to a User About Exception State Dumps About Event-Based Tracing About In-Memory Tracing About Statistics at Event-Based Tracing of Real Application Security Components About Application Sessions (XSSESSION) Event-Based Tracing About Application Principals (XSPRINCIPAL) Event-Based Tracing About Security Classes (XSSECCLASS) Event-Based Tracing About ACL (XSACL) Event-Based Tracing	
D.1.3 D.1.4 D.1.5 D.1.6 D.1.7 D.2 Abor D.2.1 D.2.2 D.2.3 D.2.4 D.2.5 D.3 Abor D.3	How to Check Which ACLs Are Associated with a Row for the Current User How to Find If a Privilege Is Granted in an ACL to a User About Exception State Dumps About Event-Based Tracing About In-Memory Tracing About Statistics at Event-Based Tracing of Real Application Security Components About Application Sessions (XSSESSION) Event-Based Tracing About Application Principals (XSPRINCIPAL) Event-Based Tracing About Security Classes (XSSECCLASS) Event-Based Tracing About ACL (XSACL) Event-Based Tracing About Data Security (XSXDS and XSVPD) Event-Based Tracing	



Glossary			
Index			



### List of Examples

2-1	Setting the Password Verifier Using the Hash Algorithm XS_SHA512	2-6
2-2	DBA Resets the Password with a Password Change Operation for User lwuser2	
	When Not Explicitly Attached to a Session	2-8
2-3	User lwuser2 Performs a Self Password Change that Fails When Explicitly Attached	
	to a Session Because the Session Lacks the ALTER USER Privilege	2-9
2-4	A Self Password Change Succeeds When Explicitly Attached to a Session and User	
	lwuser2's Session Has the ALTER USER Privilege	2-9
2-5	Configuring a Proxy Application User	2-11
2-6	Creating a Session and Switching an Application User	2-11
2-7	Creating a Regular Application Role	2-14
2-8	Creating a Dynamic Application Role	2-15
2-9	Setting Effective Dates for an Application User	2-16
2-10	Setting Effective Dates for an Application Role of an Application User	2-17
2-11	Creating a New Application User and Granting This User an Application Role	2-17
2-12	Granting an Application Role to an Existing Application User	2-18
2-13	Granting a Regular Application Role to Another Regular Application Role	2-18
2-14	Granting a Database Role to an Application Role	2-18
3-1	Creating an Application Session	3-4
3-2	Creating an Anonymous Application Session	3-4
3-3	Attaching an Application Session	3-6
3-4	Setting a Cookie for an Application Session	3-7
3-5	Assigning an Application User to an Application Session	3-8
3-6	Switching an Application User to Another Application User in the Current Application	
	Session	3-9
3-7	Registering a Global Callback in an Application Session	3-11
3-8	Saving the Current User Application Session	3-13
3-9	Detaching and Committing an Application Session	3-14
3-10	Detaching and Not Committing an Application Session	3-14
3-11	Destroying an Application Session	3-15
3-12	Creating a Namespace Template	3-17
3-13	Initializing Namespaces When Creating an Application Session	3-18
3-14	Initializing Namespaces When Attaching an Application Session	3-19
3-15	Initializing Namespaces When Assigning an Application User to an Application Session	3-20
3-16	Initializing Namespaces When Switching an Application User in an Application Session	3-21
3-17	Initializing a Namespace Explicitly in an Application Session	3-22



3-18	Setting a Namespace Attribute for an Application Session	3-23
3-19	Getting a Namespace Attribute for an Application Session	3-24
3-20	Creating a Custom Namespace Attribute for an Application Session	3-25
3-21	Deleting a Namespace in an Application Session	3-26
3-22	Enabling a Role in an Application Session	3-26
3-23	Disabling a Role in an Application Session	3-27
4-1	Adding an Aggregate Privilege to a Security Class	4-2
4-2	Adding Implied Privileges to an Aggregate Privilege	4-2
4-3	Using ALL Grant	4-3
4-4	Showing Security Class Inheritance	4-4
4-5	Adding Parent Security Classes for a Specified Security Class	4-6
4-6	Removing One or More Parent Classes for a Specified Security Class	4-6
4-7	Adding One or More Application Privileges to a Security Class	4-6
4-8	Removing One or More Application Privileges from a Specified Security Class	4-6
4-9	Removing all Application Privileges for a Specified Security Class	4-6
4-10	Adding One or More Implied Application Privileges to an Aggregate Privilege	4-6
4-11	Removing a Specified Implied Application Privileges from an Aggregate Privilege	4-7
4-12	Removing all Implied Application Privileges from an Aggregate Privilege	4-7
4-13	Setting a Description String for a Specified Security Class	4-7
4-14	Deleting a Specified Security Class	4-7
4-15	Creating an Access Control List	4-9
4-16	Denying a Privilege	4-9
4-17	Inverting an Application Privilege	4-10
4-18	Setting ACE Start-Date and End-Date	4-10
4-19	Appending an ACE to an Access Control List	4-11
4-20	Removing all ACEs from an ACL	4-11
4-21	Modifying the Security Class for an ACL	4-11
4-22	Setting or Modifying the Parent ACL	4-11
4-23	Removing all ACL Parameters for an ACL	4-11
4-24	Removing the Specified ACL Parameter for an ACL	4-11
4-25	Setting a Description String for an ACL	4-11
4-26	Deleting an ACL	4-12
4-27	Extending ACL Inheritance	4-14
4-28	Constraining ACL Inheritance: Firewall-Specific Authentication Privilege	4-14
4-29	Using a Constraining Application Privilege	4-15
5-1	Structure of a Data Security Policy	5-3
5-2	Components of a Data Realm Constraint	5-6



5-3	Column with an Additional Application Privilege That Has Been Applied	5-10
5-4	Checking Authorized Data and Masking NULL Values	5-10
5-5	Using XS_DATA_SECURITY.APPLY_OBJECT_POLICY	5-11
5-6	A Master Detail Data Realm	5-13
5-7	How a BEQUEATH CURRENT_USER View Works	5-24
5-8	How a BEQUEATH DEFINER View Works	5-25
5-9	Connecting as User SYS	5-27
5-10	Creating the DB_EMP Role	5-28
5-11	Creating the Application Role EMPLOYEE for Common Employees	5-28
5-12	Creating the Application Role IT_ENGINEER for the IT Department	5-28
5-13	Creating the Application Role HR_REPRESENTATIVE for the HR Department	5-28
5-14	Granting DB_EMP Database Role to Each Application Role	5-28
5-15	Creating Application User DAUSTIN	5-29
5-16	Creating Application User SMAVRIS	5-29
5-17	Granting the HR User the Policy Administration Privilege ADMIN_ANY_SEC_POLICY	5-29
5-18	Creating the HRPRIVS Security Class	5-30
5-19	Creating ACLs: EMP_ACL, IT_ACL, and HR_ACL	5-30
5-20	Creating the EMPLOYEES_DS Data Security Policy	5-31
5-21	Applying the EMPLOYEES_DS Security Policy to the EMPLOYEES Table	5-32
5-22	Validating the Real Application Security Objects	5-32
5-23	Disabling a Data Security Policy for a Table	5-33
6-1	How to Get an Instance of the Session Manager in Java Using a Single Connection	6-2
6-2	How to Create a Real Application Security Session in Java	6-5
6-3	How to Attach a Real Application Security Session in Java	6-5
6-4	How to Attach Using a Cookie	6-6
6-5	How to Assign an Application User to a Session in Java	6-6
6-6	How to Switch an Application User in a Session in Java	6-7
6-7	How to Enable a Real Application Security Application Role in Java	6-7
6-8	How to Disable a Real Application Security Application Role in Java	6-7
6-9	How to Test If a Real Application Security Application Role Is Enabled in Java	6-8
6-10	How to Create a Namespace in Java	6-9
6-11	How to Delete a Namespace in Java	6-9
6-12	How to Implicitly Create the Namespace in Java	6-9
6-13	How to Create a Session Namespace Attribute in Java	6-10
6-14	How to Retrieve a Session Namespace Attribute in Java	6-10
6-15	How to List Attributes in Java	6-10
6-16	How to Reset an Attribute in Java	6-11



6-17	How to Delete an Attribute in Java	6-11
6-18	How to Get the Session ID for the Session in Java	6-12
6-19	How to Get the Secure Session Cookie in Java	6-12
6-20	How to Set the Secure Session Cookie in Java	6-13
6-21	How to Detach a Real Application Security Session in Java	6-13
6-22	How to Destroy a Real Application Security Session in Java	6-13
6-23	How to Authenticate Application Users in Java	6-14
6-24	How to Construct an ACL Identifier	6-15
6-25	How to get an ACL for a Specified Data Privilege	6-15
7-1	Creating a Real Application Security Session for External Users	7-3
7-2	Attaching a Real Application Security Session for External Users	7-6
7-3	How to Assign a Real Application Security Session to External Users	7-8
7-4	How to Save a Real Application Security External User Session	7-9
8-1	Granting the Code-Based Permission CredentialAccessPermission to the xsee.jar File	8-6
8-2	Application Session Filter Sample Configuration	8-8
8-3	Application Session APIs: AttachSession and DetachSession	8-12
8-4	Application Session APIs: DestroySession	8-13
8-5	Privilege Elevation API	8-14
8-6	Namespace APIs	8-19
8-7	CheckPrivilege API	8-20
8-8	Set Up the HR Demo Application for External Principals	8-22
8-9	A Complete Application Session Filter Sample Configuration	8-26
8-10	Sample Servlet Application MyHR.java	8-29
8-11	Filter to Set Up Application Namespace	8-35
8-12	User and Group to Application Roles Mapping	8-38
11-1	Set the ACL Privilege CREATE_SESSION on Application User TEST1	11-80
B-1	Retrieving Return Codes from OCI for a Column Authorization	B-2
B-2	Using the OCIDescribeAny Function to Enable an Explicit Describe	B-5
B-3	Check Security Attributes and User Authorization	B-9



### List of Figures

1-1	Oracle Database Real Application Security Components	1-3
1-2	Three Dimensions of Data Security	1-5
3-1	Real Application Security Architecture	3-2
5-1	Real Application Security Data Security Policy Created on the EMPLOYEES Table	5-3
5-2	Real Application Security Data Security Policy Created on Master-Detail Related Tables	5-16
8-1	Application Session Service in Oracle Fusion Middleware	8-4



### List of Tables

3-1	Session Events That Can Use Callback Event Handlers	3-11
3-2	Session Privilege Checking	3-29
3-3	Session Privilege Operations and the Required Privileges to Perform Them	3-30
8-1	Session Service HR Demo(1) Logged in as Employee LPOPP	8-39
8-2	Session Service HR Demo(2) Logged in as HR Manager HRMGR	8-39
8-3	Session Service HR Demo(3) Logged in as Team Manager AHUNOLD	8-40
9-1	Oracle Database Real Application Security Data Dictionary Views	9-1
10-1	Oracle Database Real Application Security SQL Functions and Procedures	10-1
10-2	Predefined Parameters	10-2
11-1	Oracle Database Real Application Security PL/SQL Packages	11-1
11-2	Summary of DBMS_XS_SESSIONS Subprograms	11-3
11-3	Summary of XS_ACL Subprograms	11-25
11-4	Summary of XS_ADMIN_UTIL Subprograms	11-33
11-5	Summary of XS_DATA_SECURITY Subprograms	11-38
11-6	Summary of XS_DATA_SECURITY Subprograms for Managing Data Security Policies on	
	Tables or Views	11-38
11-7	Summary of XS_DATA_SECURITY_UTIL Subprograms	11-50
11-8	Summary of XS_DIAG Subprograms	11-52
11-9	Summary of XS_NAMESPACE Subprograms	11-59
11-10	Summary of XS_PRINCIPAL Subprograms	11-65
11-11	Summary of XS_SECURITY_CLASS Subprograms	11-87
B-1	Authorization Indicator Behavior (By Default)	B-3
B-2	Authorization Indicator Behavior (By Default) - OCI_ATTR_NO_AUTH_WARNING=TRUE	B-4
C-1	HR Demo Scripts and Log Files	C-1
D-1	Summary of XS_DIAG Subprograms	D-2
D-2	Real Application Security Components and Events	D-3
D-3	XSSESSION Trace Contents	D-4
D-4	XSPRINCIPAL Trace Contents	D-7
D-5	XSACL Trace Contents	D-8
D-6	XSXDS Trace Contents	D-9
D-7	XSVPD Trace Contents	D-9
D-8	Real Application Security Components and First-Failure Dump Information	D-9
D-9	Real Application Security Components and Performance Statistics	D-10



### **Preface**

Welcome to *Oracle Database Real Application Security Administrator's and Developer's Guide*. This guide describes how you may configure Oracle Database Real Application Security.

### **Audience**

This guide is intended for database administrators (DBAs), security administrators, application developers, and others tasked with configuring Oracle Database Real Application Security in an Oracle database.

### **Documentation Accessibility**

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

### **Access to Oracle Support**

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

### **Related Documents**

For more information, see these Oracle resources:

- Oracle Database Real Application Security Java API Reference
- Oracle Database Real Application Security Session Service Java API Reference

### Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.



Convention	Meaning
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.



## Changes in This Release for Oracle Database Real Application Security Administrator's and Developer's Guide

This preface contains the changes in this book for Oracle Database 23c.

#### **Deprecated Features**

#### Deprecation of Real Application Security (RAS) mid-tier master session support

Oracle Real Application Security (RAS) mid-tier master session support for Fusion Middleware application session service is deprecated with Oracle Database 23c This filter created the RAS session automatically for the FMW application server. Starting with Oracle Database 23c, you now need to write the code to create the RAS session in the future, just as you would be required to do for any other application server.

#### **Deprecation of Oracle Virtual Directory with Real Application Security**

The use of Oracle Virtual Directory with Oracle Real Application Security is deprecated with Oracle Database 23c.

Using OVD with Oracle Real Application Security is deprecated, because OVD is no longer updated as a separate product

### **Desupported Features**

#### **Desupport of RASADM**

The Real Application Security GUI administration tool (RASADM) is desupported with Oracle Database 23c.

RASADM is no longer supported for use with Oracle Database 23c. Oracle recommends that you use the RAS PL/SQL API which includes all the functionality of RASADM.



1

### Introducing Oracle Database Real Application Security

### This chapter contains:

- What Is Oracle Database Real Application Security?
- Data Security Concepts Used in Real Application Security
- Application Session Concepts Used in Application Security
- · Flow of Design and Development
- Scenario: Security Human Resources (HR) Demonstration of Employee Information
- About Auditing in an Oracle Database Real Application Security Environment
- · Support for Pluggable Databases

### 1.1 What Is Oracle Database Real Application Security?

Oracle Database Real Application Security is a database authorization model that:

- Supports declarative security policies
- Enables end-to-end security for multitier applications
- Provides an integrated solution to secure database and application resources
- Advances the security architecture of Oracle Database to meet existing and emerging demands of applications developed for the Internet

Traditional security was designed for client/server systems. These systems had a significantly smaller number of users than newer applications designed for the Internet. When application developers found traditional security inadequate, they often moved it from the database layer to the application layer. To accomplish this, developers frequently built their own tables and defined their own application users. Because security was encoded in the application layer, rather than in the database, application users and application roles were typically known only to the application. In other words, database users were not application-level users, hence the user identity was not known during the access control decision in the database. Furthermore, database operations were limited to DDLs and DMLs that do not represent application-level tasks or operations, hence the operation context was also not known during the access control decision in the database. These practices exposed the database to vulnerability.

Real Application Security is designed to:

- Manage application security for application users rather than database users
- Enable developers to manage security for application level tasks
- Enable application user identity to be known during security enforcement
- Enable developers to return security to the database layer, either incrementally, or all at once



This section discusses traditional security and Real Application Security, indicating how Real Application Security improves upon traditional security.

This section describes these concepts:

- Disadvantages of Traditional Security for Managing Application Users
- Advantages of Real Application Security
- Architecture of Real Application Security

### 1.1.1 Disadvantages of Traditional Security for Managing Application Users

Using the traditional security model, it was often difficult to manage three-tier applications, especially when performing these security tasks:

- Extending security policies independent of application code
- Enforcing security policies at the database level, where the application user is unknown
- Enforcing least privilege principle as full access is granted to highly privileged twotier components

### 1.1.2 Advantages of Real Application Security

Real Application Security enables these security tasks, which improve database security and performance:

- Three-tier and two-tier applications can declaratively define, provide, and enforce access control requirements at the database layer.
- The database can provide a uniform security model across all tiers and support
  multiple application user stores, including the associated roles, authentication
  credentials, database attributes, and application-defined attributes. This model
  enables application users to have a single unique global identity across an Oracle
  enterprise.
- An Oracle database can natively support the application security context. The
  database supports integrated policy specification and enforcement for both the
  application and the database, so the application does not need to do this through
  application code. Because the database stores the application security context
  information, this also reduces network traffic.
- Developers can use Real Application Security to control application user access to data in an Oracle database throughout all components of an Oracle enterprise in a common manner.

See Configuring Data Security for more information about defining data security policies and access control requirements.

### 1.1.3 Architecture of Real Application Security

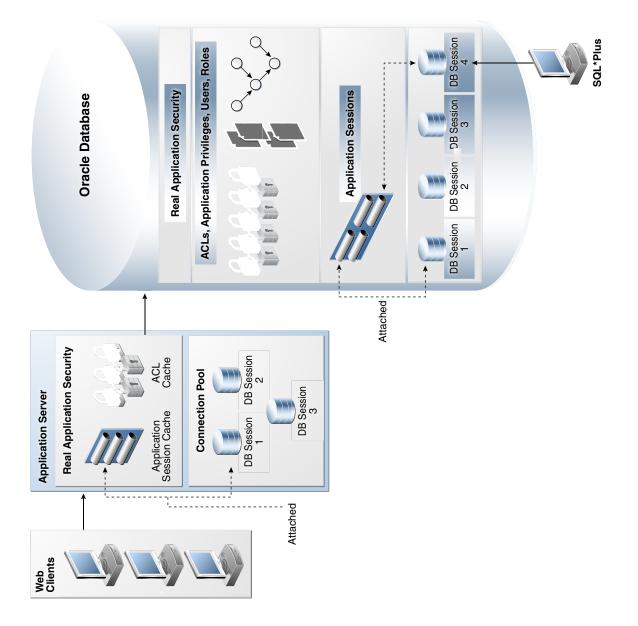
Real Application Security is managed through a collection of PL/SQL and Java APIs. This architecture that enables you to configure its components—application users, application roles, sessions, and other security-related components. With Real



Application Security, you configure application counterparts to the traditional user, role, and session, through the use of entities, which are stored in tables.

Figure 1-1 shows the various components used in Oracle Database Real Application Security. This includes application users, application roles, access control lists, security classes, and application sessions. These components are discussed in the following sections. Figure 1-1 also shows Web applications establishing application sessions to the database.

Figure 1-1 Oracle Database Real Application Security Components



### 1.2 Data Security Concepts Used in Real Application Security

This section describes access control terms and concepts that you need to understand before you can begin to configure Real Application Security. Using the PL/SQL administrative interfaces, you can create and manage the entities described here: application user,

application role, principal, application privilege, security class, *access control list* (ACL), access control entry (ACE), and data realm.



When a term such as *application user* or *application role* is used here, it applies to Real Application Security; when it is important to distinguish the database type, either no qualifier is used or the qualifier *database* is used.

### See Also:

- Configuring Application Users and Application Roles
- Configuring Application Privileges and Access Control Lists

#### This section contains:

- About Data Security with Oracle Database Real Application Security
- Principals: Users and Roles
- Application Privileges
- Security Classes in Oracle Database Real Application Security
- Access Control Entry (ACE)
- Access Control List (ACL)
- Data Security Policy

### 1.2.1 About Data Security with Oracle Database Real Application Security

Effective security requires defining which application users, applications, or functions can have access to which data, to perform which kinds of operations. Thus, effective security has these three dimensions:

- 1. which application users
- 2. can perform which operations
- 3. on which data

You define (1) principals, (2) application privileges, and (3) objects in relation to these three dimensions, respectively. Principals are users and roles. A role can represent attributes of an application user, system state, or a piece of code.

Principals and application privileges are related in a declarative way by defining ACLs. These ACLs are then related to the data by defining Data Security policy that protects rows and columns of table data. For example, you can protect table data by using PL/SQL procedures to set controlling ACLs.

Figure 1-2 illustrates an example where the user, ProjectManager has the ModifyProject privilege on a data realm comprised of Team A's projects.



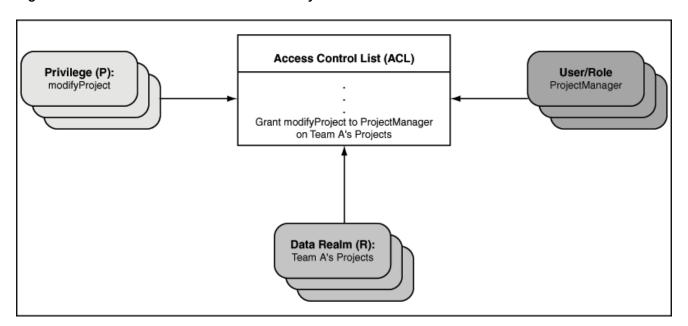


Figure 1-2 Three Dimensions of Data Security

### 1.2.2 Principals: Users and Roles

When discussing fine-grained database access control, a **principal** is an application user or an application role or a database user or a database role. An **application user** can be a person or an autonomous application process that accesses information in the database. An **application role** is a logical grouping of application privileges required to accomplish a real life task. An application role can contain other application roles, but this recursion cannot be circular. You use application roles to associate application users, both database users and application users with privileges.

Oracle Database supports the following as principals:

### Database users and database roles

A database user is also sometimes referred to as a database *schema* or a user *account*. When a person or application logs onto the database, it uses a database user (schema) and password.

A database role corresponds to a set of database privileges that can be granted to database users, applications, or other database roles — see "Understanding the Difference Between Database Roles and Application Roles".

#### Application users and Application roles

The term *application*, as used by Real Application Security, refers to the creation of an application user, application role, or session that contains only information pertinent to the application that the application user is logging onto. Application users and application roles are defined by an application, and they do not need to be tied to any database schema.

Application users can also create heavyweight database sessions by connecting to the database directly. These are called direct login application users. See "About Creating a Direct Login Application User Account". When an application user creates a heavyweight database session, the user's default schema is set to a preconfigured value meant solely for name resolution purposes, such as HR.



An application role can only be granted to an application user or to another application role. You cannot directly grant database privileges to application users and application roles. See "Granting Database Privileges to Application Users and Application Roles" for further details.

### See Also:

- "About Configuring Application Users"
- "About Configuring Application Roles"

This section includes the following sections:

- Understanding the Difference Between Database Users and Application Users
- Understanding the Difference Between Database Roles and Application Roles
- Granting Database Privileges to Application Users and Application Roles

### 1.2.2.1 Understanding the Difference Between Database Users and Application Users

Database users are also referred to as *traditional* users, and have these characteristics:

- They are associated with schemas and passwords.
- They can create heavyweight sessions to schemas with which they are associated.

Application users are defined by an application, and have these characteristics:

- They do not own database schemas.
- They can create application sessions to the database through the middle tier.
- They can create heavyweight database sessions by connecting to the database directly. (See "About Creating a Direct Login Application User Account".)



In a heavyweight session, the user is associated with a default schema.

### 1.2.2.2 Understanding the Difference Between Database Roles and Application Roles

A database role is traditionally thought of as a named set of database privileges.

Database roles have these characteristics:

They are granted privileges, just as database users can be granted privileges.



- They serve as intermediaries for mapping database privileges to database users (and applications) as follows: a role is granted privileges, and the role is then granted to users (giving them the privileges).
  - 1. Grant privileges to database role
  - 2. Grant database role to database user

The database user now has the privileges of the database role.



In traditional database terminology, a role is considered to be the same thing as the set of privileges that are granted to it.

An application role can be regarded as the set of application-defined privileges that are associated with it using the mechanism of a declarative access control list (ACL), discussed in "Access Control List (ACL)".

Application roles have these characteristics:

- They use an access control list (ACL), rather than a database grant, as the intermediary that maps application privileges to users or roles.
- They can be only granted to application users or application roles.
- They cannot be granted to a database role, unlike a database role can be granted to an application role.



In access control terminology, application roles are classified with application users as principals.

### 1.2.2.3 Granting Database Privileges to Application Users and Application Roles

You cannot grant database privileges directly to application users and application roles. Instead, you grant the database privileges to a database role, and then grant the database role to the application role in these steps.

- 1. Grant database privileges to database role.
- Grant database role to the application role.

The statements in the following code do exactly this, effectively granting the database SELECT privilege to the application role, HRREP.

```
CREATE ROLE db_hrrep;
GRANT SELECT ON hr.employees TO db_hrrep;
GRANT db hrrep TO HRREP;
```

Application users already created or subsequently created, with that application role, acquire this application privilege.



### 1.2.3 Application Privileges

An **application privilege** is a particular right or permission that can be granted or denied to a principal. Application developers define application privileges in a security class.

The set of application privileges granted to a principal controls whether or not this principal can perform a given operation on the data that it protects. For example, if the principal (database user) HR wants to perform the SELECT operation on a given resource, then SELECT privileges must be granted to principal HR before the SELECT operation.

Application privileges can also be aggregated. An **aggregate privilege** is an application privilege that implies other application privileges. These implied privileges can be any application privileges defined by the current security class or an inherited privilege. When an aggregate privilege is granted or denied, its implied application privileges are implicitly granted or denied.

Aggregate privileges simplify usability when the number of application privileges grows. For example, instead of granting each application privilege separately, you can group related application privileges into an aggregate privilege. Then, you can use a single grant to enable a principal to access all the application privileges contained in the aggregate privilege.

### 1.2.4 Security Classes in Oracle Database Real Application Security

A **security class** is a scope for a set of application privileges.

A security class includes application privileges that it inherits from other security classes, and it can include application privileges that it defines.

A security class is typically associated with an access control list (ACL), and the ACL can grant application privileges in the security class to specific principals. See "Access Control List (ACL)".

Example 4-4 shows how to create a security class policy.

### 1.2.5 Access Control Entry (ACE)

An **access control entry** (**ACE**) either grants or denies application privileges to a particular principal (application user or application role).

An ACE is an element in an array named <code>ace\_list</code>. The whole array is called by and becomes part of the access control list (ACL).

The ACE does not, itself, specify which data to protect; that is done by associating the ACL with target data, such as a set of rows in an order entry table. You can make this association by creating a **data realm** to restrict the user to modifying only those rows, or by using the PL/SQL procedure XS DATA SECURITY. SET ACLS.

### 1.2.6 Access Control List (ACL)

An **access control list (ACL)** is a list of access control entries (ACEs), which permit or deny application privileges to one or more principals.



If the ACL you create relies on a set of custom application privileges that you define in your own security class, then you must explicitly associate that security class with the ACL. See Example 4-15 for an example.

If the only application privileges used in the ACL are defined in the DML security class, then no security class association is needed as that is the default. See a description in "DML Security Class".

### 1.2.7 Data Security Policy

To protect data within a database table, you must create a data security policy. Database records, both at row and column level, can be protected using the fine-grained access control described in this section.

The data security policy performs the following functions:

• Specifies the data that you want to protect. The data can be indicated by a WHERE clause in a data realm of one or more rows that you design. It can also be defined using named notation by using an association operator to associate the parameter to the left of the arrow (=>) with the actual parameter to the right of the arrow. For example, in Example 5-20, each realm is defined using association operators.

The data security policy can contain one or more data realms.

- Associates each data realm with one or more access control lists (ACLs) that specify the
  application privileges required to access rows and columns of the data realm to form
  what is called a data realm constraint. A given ACL protects a given data realm and
  controls access to particular application users or application roles (called principals). (See
  "Access Control List (ACL)" for more information about ACLs.)
- Optionally applies additional application privileges to protect a particular column to form
  what is known as column constraints. This is useful in cases where you need to add an
  extra layer of security for sensitive data.
- Associates additional custom application privileges. For example, an administrator could create an APPROVE\_TRANSACTION privilege, which controls whether a user can take a particular action on the row. Assuming SELECT privilege is granted to all users, all users could see the row, but only some users can perform the transaction approval action.

In summary, the application user who logs in will only be allowed to perform operations including DML on records within the data realm, including individual rows of data, based on the application privileges in its associated ACLs. Thus, the data security policy is composed of data realm constraints and column constraints that protect the data realm by only allowing access to application users who have application privileges in the associated ACLs.

For example, suppose you have a sales table that lists all sales representatives, their regions, the products they are responsible for, product categories, and product prices. When individual sales representatives log on, each representative would see selected data for all other sales representatives, such as sales representatives for particular product categories based on data realm constraints. If you wanted to restrict the display of product prices to sales representatives by region, you could apply additional application privileges to the column listing product prices, in this case using column constraints.

Configuring Data Security describes in detail how to protect database objects.



### 1.3 Application Session Concepts Used in Application Security

Real Application Security introduces the concept of an application session. Within the context of application sessions, there are three types of user identities:

- Application session user: The user associated with the application session.
   Application session access to database objects is checked against the permissions granted to this user.
- Traditional (heavyweight) session user: The user that established the database session.
  - This user can be an application user or a database user, as long as database authentication credentials are available.
- Schema owner: The database schema is the schema associated with the traditional database session and is only used for object name resolution.

Traditional database user sessions have these characteristics:

- They hold their own database resources, such as transactions and cursors.
- They consume many server resources.

Application sessions have these characteristics:

- They contain information that is pertinent only to the application.
- They can be dedicated to each end application user.
- They can persist until the application user logs out of the application or the application terminates unexpectedly.

See Configuring Application Sessions for more information about application sessions.

### 1.4 Flow of Design and Development

You should be familiar with the concepts introduced in this chapter to take full advantage of Real Application Security.

In general, identify all tasks an application performs that require application privileges to control data access. Then, add the appropriate application privileges to a security class so that you can reference them in an ACL and grant them to the application users and application roles. This process involves these tasks:

- Create a default set of meaningful application roles based on the features the application provides.
- Identify the tables that require data security protection based on the application table design and security requirements, and define the data realms, including column protection.
- Define data security policies based on the application requirements and the rules applied on the tables.
- Ensure that ACLs used in the data security policy and functional security grant the appropriate application privileges to application roles.



#### This section contains:

- Design Phase
- Development Flow Steps

# 1.4.1 Design Phase

In the design phase, you identify all the tasks an application performs that require application privileges to control data access.

For example, during the design phase, the application policy designer must identify:

- 1. The set of application-level operations that require access control.
- 2. The rows and columns of tables and views that can be accessed as part of the application-level operations.
- 3. The set of actors or principals (users and roles) that can perform these operations.
- 4. The runtime application session attributes that identity rows of a table or views. These attribute names are used within the predicates that selects the rows to be authorized, and their values are set during the execution of the application.

## 1.4.2 Development Flow Steps

In the development phase, as the Real Application Security administrator, you use Real Application Security components to develop your data security policies.

Follow these steps to develop your data security policies:

- 1. Create the corresponding application users and roles. If using an external directory server, create the application users and roles or user groups in the directory server. Follow this procedure to create these principals natively in the database:
  - a. Create the application roles and grant application roles to application roles, if needed. See About Configuring Application Roles.
  - **b.** Create the application users and grant application roles to the application users. See About Configuring Application Users.
- 2. Create each privilege class that you plan to use to develop the security policies for your application. Each privilege class consists of one or more appropriate privileges that you define and can reference in an ACL and also grant them to the application users and application roles. Each privilege class authorizes by means of ACLs the required application-level operations of a data security policy. See About Configuring Security Classes and About Configuring Access Control Lists.
- 3. Create one or more session namespaces that can be used across different application sessions. This consists of defining for a session namespace its set of properties (application attributes) and its associated access control policy or ACL that you can choose from a list or create. See About Manipulating the Application Session State.
- Create the data security policy by associating each data realm with an ACL, so as to create both data realm authorization and column authorization as needed. See About Data Security.

This process consists of four parts:



- a. Policy Information You choose the object to be protected and the privilege class to protect it as well as specify the policy name and select the policy owner. See Understanding the Structure of the Data Security Policy.
- b. Column Level Authorization You choose the name of the column to be protected and select the privilege to be granted to access the column, which is associated with the privilege class you selected in Step 4a. See Applying Additional Application Privileges to a Column.
- c. Data Realm Authorization You create a SQL predicate to represent the data realm to be protected and add each to a data realm grant list. Then you choose or create the ACL to protect the data realm. Next, create privilege grants to be added to a privilege grants list consisting of each principal and whether it is allowed authorization or denied authorization by selecting the appropriate privilege. See About Designing Data Realms.
- d. Apply Policy You can apply, remove, enable, or disable the data security policy you are creating and choose to specify certain apply options, allowing the owner of the table or view to bypass this data security policy, and whether to enforce statement types for this policy. See About Enabling Data Security Policy for a Database Table or View.



Scenario: Security Human Resources (HR) Demonstration of Employee Information that describes in detail how the development flow is implemented for an example policy scenario for the security human resources (HR) demonstration of employee information using the concepts and components of Real Application Security.

# 1.5 Scenario: Security Human Resources (HR) Demonstration of Employee Information

This section presents an example policy that provides a high-level overview of Real Application Security. It is a simple scenario aimed at explaining the basic Real Application Security concepts. You should be familiar with the following concepts, introduced in "Data Security Concepts Used in Real Application Security":

- Principals application users and application roles
- Security classes and application privileges
- Access control lists and entries (ACLs and ACEs)
- Data security policy

This same scenario appears throughout the book, to illustrate different components of Real Application Security. It is also described in detail in Real Application Security HR Demo and Real Application Security HR Demo Files to demonstrate how to use advanced concepts of Real Application Security to handle a more complex policy.

This section includes the following topics:

- Basic Security HR Demo Scenario: Description and Security Requirements
- Basic HR Scenario: Implementation Overview



# 1.5.1 Basic Security HR Demo Scenario: Description and Security Requirements

Susan Mavris (SMAVRIS) is an employee in the Human Resources department. Her job title is Human Resources Representative. In this capacity, she is in charge of managing the human resources information for all employees, including department 60 (IT). She can view and update all the employee records, including the SALARY column.

David Austin (DAUSTIN) is an employee in the IT department. His job title is Assistant Department Manager. In this capacity, he can view employee records in the IT department, but he cannot view the SALARY column, except for his own salary record.

Secure authorization requires defining which application users and application roles can have access to which data, to perform which kinds of operations. These three security dimensions must be defined: protected data, principals, and application privileges. (see "About Data Security with Oracle Database Real Application Security").

#### In this basic scenario:

- The data to be protected is employee information and it is protected in three ways:
  - Access to an employee's own record, including the SALARY column.
  - Access to all the records in the IT department, excluding the SALARY column.
  - Access to all employee records, including the SALARY column.
  - Users are allowed access to employee data in the following ways:
    - Each user can view their own record, including the SALARY column.
    - Application user DAUSTIN in his role as Assistant Department Manager is allowed to view all the records in the IT department, excluding the SALARY column.
    - Application user SMAVRIS in her role as human-resources representative is allowed to view and update all employee records, including the SALARY column.
- Database role DB\_EMP is created and granted SELECT, INSERT, UPDATE, and DELETE privileges on HR.EMPLOYEES.
- Application roles are created as follows:
  - EMPLOYEE role is granted to both application users DAUSTIN and SMAVRIS. Database role DB EMP is granted to EMPLOYEE role.
  - IT\_ENGINEER role is granted to only application user DAUSTIN. Database role DB\_EMP is granted to IT ENGINEER role.
  - HR\_REPRESENTATIVE role is granted to only application user SMAVRIS. Database role
     DB EMP is granted to HR REPRESENTATIVE role.
- The VIEW\_SALARY application privilege is created to control access to the SALARY column.
   The HR\_PRIVILEGES security class is created in which to scope the VIEW\_SALARY application privilege.
- ACLs are created to define the degree of access to employee records in the following ways:
  - EMP\_ACL grants the EMPLOYEE role the SELECT database privilege and VIEW\_SALARY
    application privilege to view an employee's own record, including the SALARY column.



- IT\_ACL grants the IT\_ENGINEER role only the SELECT database privilege to view
  the employee records in the IT department, but it does not grant the
  VIEW SALARY privilege that is required for access to the SALARY column.
- HR\_ACL grants the HR\_REPRESENTATIVE role SELECT, INSERT, UPDATE, and
  DELETE database privileges to view and update all employee's records, and
  granting the VIEW SALARY application privilege to view the SALARY column.
- The HR demo secures the HR.EMPLOYEE table by creating and applying the data security policy, EMPLOYEES\_DS, that has the following three data realms and column constraint:
  - An employee's own record realm. The ACL, EMP\_ACL controls this realm, which
    grants application role EMPLOYEE privileges to access the realm, including the
    SALARY column.
  - All the records in the IT department realm. The ACL, IT\_ACL controls this
    realm, which grants application role IT\_ENGINEER privileges to access the
    realm, but excluding the SALARY column.
  - All the employee records realm. The ACL, HR\_ACL controls this realm, which
    grants application role HR\_REPRESENTATIVE privileges to access the realm,
    including the SALARY column.
  - A column constraint that protects the SALARY column by requiring the VIEW SALARY privilege to view its sensitive data.

## 1.5.2 Basic HR Scenario: Implementation Overview

To implement the basic human-resources security scenario, in addition to identifying the protected data, the principals, and the application privileges, you must define the following:

- A database user as the Real Application Security Administrator and then connect as the Real Application Security Administrator to create the components.
- How the principals connect with the database to access the data.
- The access control lists (ACLs) that grant the application privilege and any database privileges to the principals.
- A data security policy that associates the ACLs with the particular data (rows) that the principals need to access.

In this basic scenario, application users SMAVRIS and DAUSTIN connect to the database directly as the principals.

The application user account that is created for application users SMAVRIS and DAUSTIN are principals in this scenario. Each application user account is granted application roles that, ultimately, has the SELECT privilege on the database table that contains the employee information. The application role is a principal in this scenario.

A database role, DB\_EMP serves as intermediary between the application role and the database privilege because database privileges can be granted only to database users and roles. That is, the necessary database privileges are granted to a database role, and that role is granted to each application role (the principal).

The database SELECT privilege applies to the entire table. The principal must also be granted an Real Application Security application privilege such as the DML SELECT



privilege, which can be restricted to certain rows of the database table. This restriction is implemented using an access control list (ACL) and a data security policy.

The HR scenario requires the following components for the security model:

- **Protected data**: Employee information is stored in the table EMPLOYEES of the sample database schema HR (delivered with Oracle Database).
- Application role: Application roles, EMPLOYEE, IT\_ENGINEER, and HR\_REPRESENTATIVE are created for performing tasks. The application roles are defined with the XS PRINCIPAL.CREATE ROLE procedure.
- **Application user:** Application users, SMAVRIS and DAUSTIN, are created and defined. SMAVRIS is granted the application roles EMPLOYEE and HR\_REPRESENTATIVE. DAUSTIN is granted the application roles EMPLOYEE and IT ENGINEER.
- Database access: Application users SMAVRIS and DAUSTIN are given a database
  password for direct database login. In order to grant SELECT, INSERT, UPDATE, and DELETE
  privileges on table EMPLOYEES to application roles EMPLOYEE, IT\_ENGINEER,
  HR\_REPRESENTATIVE a database role, DB\_EMP, is created and granted these database
  privileges. The application roles are then granted this database role.
- Application Privilege: A single security class, HR\_PRIVILEGES, is created which defines
  a single custom application privilege, VIEW\_SALARY. Through inheritance, the predefined
  application privilege SELECT is also available in this security class. These application
  privileges will be used in connection with a data security policy to allow read access to
  employee information. The security class is created by the
  XS.SECURITY CLASS.CREATE SECURITY CLASS procedure.
- ACL: The SELECT and VIEW\_SALARY privileges are granted to application role EMPLOYEE by the access control list (ACL), EMP\_ACL that is created by XS\_ACL.CREATE\_ACL procedure. The SELECT privilege is granted to application role IT\_ENGINEER by the ACL, IT\_ACL that is created by XS\_ACL.CREATE\_ACL procedure. The SELECT, INSERT, UPDATE, and DELETE privileges are granted to application role HR\_REPRESENTATIVE by the ACL, HR\_ACL that is created by XS\_ACL.CREATE\_ACL procedure to view and update all employee's records, and granting the VIEW\_SALARY application privilege to view the SALARY column.
- Data Security Policy: The data security policy is defined and created with the XS\_DATA\_SECURITY.CREATE\_POLICY procedure. This data security policy defines three data realms (an employee's own record realm that can view the realm including the SALARY column, all the records in the IT department realm that can view the IT department excluding the SALARY column, and all the employee records realm that can view the realm including the SALARY column) and a column constraint. The data security policy associates the ACLs EMP ACL, IT ACL, and HR ACL with its respective data realm.

Introducing this example in this chapter provides an overview of the requirements for implementing a policy using Real Application Security. Actual implementation of these tasks requires a systematic understanding of all the Real Application Security concepts introduced in this chapter, and further discussed in subsequent chapters. The complete example, including implementation details, appears in "Real Application Security: Putting It All Together".



# 1.6 About Auditing in an Oracle Database Real Application Security Environment

Another aspect of security is auditing in an Oracle Database Real Application Security environment. Real Application Security administration and run-time actions can be audited by configuring and enabling unified audit policies. For information about unified auditing in an Oracle Database Real Application Security environment, see *Oracle Database Security Guide*.

The following static data dictionary views are defined for auditing policies specifically for Oracle Database Real Application Security:

- DBA\_XS\_AUDIT\_POLICY\_OPTIONS describes the auditing options that were defined for Real Application Security unified audit policies. See *Oracle Database Reference* for more information.
- DBA\_XS\_AUDIT\_TRAIL provides detailed information about Real Application
   Security that were audited. See Oracle Database Reference for more information.
- DBA\_XS\_ENB\_AUDIT\_POLICIES lists users for whom Real Application Security unified audit polices are enabled. See Oracle Database Reference for more information.

# 1.7 Support for Pluggable Databases

The multitenant architecture enables an Oracle database to contain a portable collection of schemas, schema objects, and nonschema objects that appear to an Oracle Real Application Security application user as a separate database. A multitenant container database (CDB) is an Oracle database that includes one or more pluggable databases (PDBs).

Oracle Real Application Security can be used with Oracle Multitenant to provide increased security for consolidation.

Because Oracle Real Application Security entities are scoped within a PDB, each PDB has its own Real Application Security metadata, such as users, roles, privileges, ACLs, data security policies, and so forth. As a result, Real Application Security can prevent privileged user access inside a PDB between and among applications and between the PDB and the common privileged user at the container database.

As SYS is the schema owner for Oracle Real Application Security entities, Real Application Security entities created in root can only be accessed by the SYS user in root. The same is true for other operating systems in that the SYS user is the schema owner for Oracle Real Application Security entities and only the SYS user has access to these entities. Similarly, Real Application Security entities created within a local PDB, can only be accessed in the local PDB.

Since Oracle Real Application Security direct login users have a password associated with them, these users can be provisioned within a PDB, using a single sqlnet.ora parameter to support them.

Oracle Real Application Security administration involves PDB specific administrative privileges and a schema to qualify the name for Real Application Security entities. The



schema name can be common; however, entities created under the naming scope of a common schema are not common.

Oracle Real Application Security auditing is PDB specific.

An Oracle Real Application Security application user can connect to a PDB using a service whose pluggable database property has been set to the relevant PDB.

### See Also:

Introduction to the Multitenant Architecture and Overview of the Multitenant Architecture in *Oracle Database Concepts*.

Oracle Database Administrator's Guide for more information about PDBs and for more details about configuring the services to connect to various PDBs



2

# Configuring Application Users and Application Roles

### This chapter contains:

- About Configuring Application Users
- About Configuring Application Roles
- Effective Dates for Application Users and Application Roles
- About Granting Application Privileges to Principals



"XS\_PRINCIPAL Package"

# 2.1 About Configuring Application Users

This section contains the following topics:

- About Application User Accounts
- Creating a Simple Application User Account
- About Creating a Direct Login Application User Account
- Resetting the Application User's Password with the SQL\*Plus PASSWORD Command
- Configuring an Application User Switch
- Validating an Application User

## 2.1.1 About Application User Accounts

Traditional database users own database schemas and can create traditional heavyweight database sessions to those schemas.

Application users do not own database schemas, but can create application sessions to the database through the middle tier provided they are granted the role or roles with the appropriate object privileges for accessing tables. Application users can also create heavyweight database sessions by connecting to the database directly through direct login application user accounts provided these accounts are associated with a schema and the XSCONNECT application role is granted to these application users. A profile can also be created and assigned to each of these application users.

This section contains: General Procedures for Creating Application User Accounts.

### 2.1.1.1 General Procedures for Creating Application User Accounts

The general procedure for creating an application user account is as follows:

1. Create a security manager user, <code>sec\_mgr</code>, as follows and grant this user <code>create session</code> database privilege and Real Application Security <code>xs\_session\_admin database</code> role. Next, execute the <code>xs\_admin\_util.grant\_system\_privilege call</code> to grant the Real Application Security least system privilege <code>PROVISION</code> to <code>sec\_mgr</code> as a database user. As the security manager, you can now create users and roles, set passwords, and so forth, and administer sessions using the Real Application Security least system privilege.

```
sqlplus /nolog
SQL> connect sys/password as sysdba
SQL> grant create session, xs_session_admin to sec_mgr identified
by password;
SQL> exec sys.xs_admin_util.grant_system_privilege('provision',
'sec_mgr', sys.xs_admin_util.ptype_db);
```

2. Log in to SQL\*Plus as a user who has either the Real Application Security PROVISION system privilege or the database CREATE USER system privilege.

```
sqlplus sec_mgr
Enter password: password
Connected.
```

See "XS\_PRINCIPAL Package" for more information about the XS\_PRINCIPAL package and specifically the "CREATE\_USER Procedure".

You must have the privileges required to create, modify, or drop application users and roles. These privileges are governed by the same system privileges required to create, modify, or drop database users and roles. For more information about these and other SQL statements, see *Oracle Database SQL Language Reference*.

3. Create the application users with the XS PRINCIPAL. CREATE USER procedure.

Select the appropriate type, and follow the instructions in these sections:

- "Creating a Simple Application User Account"
- "About Creating a Direct Login Application User Account"

### Other Tasks

After you create the application user account, you can grant the account a role, which provides privileges for the application users. For more information, see "Granting an Application Role to an Existing Application User".



# 2.1.2 Creating a Simple Application User Account

### Note:

In SQL\*Plus, case sensitivity is an issue for lower case characters and special characters, so keep these guidelines in mind.

 An application user whose name contains lower case or special characters must connect to SQL\*Plus with the account name in double quotation marks:

### For example:

```
CONNECT "lwuser1"
Enter password: password
Connected.
```

• The name of an application role that contains lower case or special characters must be entered in SQL\*Plus enclosed in double quotation marks.

### For example:

```
GRANT cust_role TO "app_regular_role";
```

When you create a simple application user account, the schema argument specifies the schema name to use to resolve unqualified names. This does not give you any privileges, and it is just used for name resolution purposes. If the schema name is not specified, XS\$NULL, is used.

To create a simple application user account, do the following:

1. Log in.

For example, if sec mgr has the CREATE USER privilege, log in as follows:

```
sqlplus sec_mgr
Enter password: password
Connected.
```

2. Create the application user account.

### For example:

```
BEGIN
   SYS.XS_PRINCIPAL.CREATE_USER('lwuser1');
END;
//
```

As a user with DBA role, you can check the user creation by querying the DBA\_XS\_USERS data dictionary view as follows. See "DBA\_XS\_USERS" for more information.

```
SELECT NAME FROM DBA_XS_USERS;

NAME

XSGUEST
LWUSER1
```



This output displays the existing application user accounts. The XSGUEST user account is an already existing or predefined system created user account.

For detailed information about the XS\_PRINCIPAL.CREATE\_USER procedure, see "CREATE USER Procedure".

You can delete an application user account using the XS\_PRINCIPAL.DELETE\_PRINCIPAL procedure, see "DELETE\_PRINCIPAL Procedure".

## 2.1.3 About Creating a Direct Login Application User Account

This section contains:

- Creating Direct Login Application User Accounts
- Procedure for Creating the Direct Login Application User Account
- Setting a Password Verifier for Direct Application User Accounts
- Oracle Label Security Context Is Established in Direct Logon Session

## 2.1.3.1 Creating Direct Login Application User Accounts

You can use an application user account to directly log into the database. This is useful for users who need to perform functions such as logging directly into SQL\*Plus without logging in through SSO or a Web interface. The direct login user must have a password.

### 2.1.3.2 Procedure for Creating the Direct Login Application User Account

To create a direct login application user account:

 Log in as described in "General Procedures for Creating Application User Accounts".

```
sqlplus sec_mgr
Enter password: password
Connected.
```

2. Create the application user account.

For example, to create an application user account, lwuser1, whose default database schema is HR:

```
BEGIN
   SYS.XS_PRINCIPAL.CREATE_USER
        (name => 'lwuser1',
        schema => 'HR');
END;/
```



If the schema does not exist, the direct login fails.



When this Real Application user directly connects to the database for name resolution of unqualified database objects in queries,  ${\tt HR}$  schema is used as the default schema. For example:

```
SELECT COUNT(*) FROM EMPLOYEES;
```

3. Create a password for the application user account.

#### For example:

Set the password as described in "SET\_PASSWORD Procedure". When you use the SET\_PASSWORD procedure, it creates a verifier for you based on the password and the type parameter, and then inserts the verifier and the value of the type parameter into the dictionary table.



Replace *password* with a secure password. See *Oracle Database Security Guide* for more information about password guidelines.

4. Create a profile named prof and assign this profile to the application user account.

### For example:

```
CREATE PROFILE prof LIMIT PASSWORD_REUSE_TIME 1/1440 PASSWORD_REUSE_MAX 3 PASSWORD_VERIFY_FUNCTION Verify_Pass;

BEGIN

SYS.XS_PRINCIPAL.SET_PROFILE('lwuser1','prof');
END;
```

The user assigning the profile must have ALTER\_USER privilege. See the "SET\_PROFILE Procedure" for more information.

5. Grant the role **XSCONNECT** to the user to allow access to the database.

### For example:

Next, you are ready to assign privileges to the application user account. Go to "About Granting Application Privileges to Principals".

Afterward, the user can connect to the database as follows. For example:

```
CONNECT lwuser1
Password: password
```

## 2.1.3.3 Setting a Password Verifier for Direct Application User Accounts

Optionally, you can set a password verifier for this password (a hash transformed password), enabling administrators to migrate users into Real Application Security with knowledge of the

verifier and not the password. If you do not set a password verifier, the default hashing algorithm is XS\_SHA512. For more information, see the SET\_PASSWORD Procedure and the SET\_VERIFIER Procedure.

Example 2-1 shows how to use the XS\_PRINCIPAL.SET\_VERIFIER procedure to set the password verifier to the value as determined from a query of the XS\$VERIFIERS dictionary table, using the hashing algorithm XS\_SHA512 for the application user account LWUSER1 by following these steps:

- 1. Query the view DBA XS OBJECTS to obtain the ID value for user LWUSER1.
- Query the XS\$VERIFIERS dictionary table for user LWUSER1 whose ID is 2147493730. The value of the verifier includes its type as value "T" followed by a colon (:) to denote that it is a verifier type of XS\_SHA512, which is also indicated as being of type# 2.
- 3. Using the entire verifier value including "T:", set the verifier for user LWUSER1. The following example shows each of these steps.

# **Example 2-1** Setting the Password Verifier Using the Hash Algorithm XS\_SHA512

```
sqlplus sec mgr
Enter password: password
Connected.
SQL> column name format A10;
SQL> column owner format A6;
SQL> select NAME, OWNER, ID, TYPE, STATUS from DBA XS OBJECTS where
NAME = 'LWUSER1';
NAME OWNER ID TYPE
                                    STATUS
_____ ___
LWUSER1 SYS 2147493730 PRINCIPAL
                                            VALTD
SQL> column user# format 9999999999;
SQL> column type# format 99;
SQL> column verifier format A62;
SQL> select USER#, VERIFIER, TYPE# from XS$VERIFIERS where USER# =
'2147493730';
     USER#
VERTFIER
                                                          TYPE#
T:9BA95FEF2C2630A2BAACF2E7C5E41B0D50CDC7B0B60C88AD4FE81F8155D0
02F99EEAF9D95477E4749870C67FDE870E154ED17809C359777F979E269010
           823FB981B2A998915EB1439FE3C6C1542A239C
SQL> BEGIN
SYS.XS PRINCIPAL.SET VERIFIER('lwuser1','T:9BA95FEF2C2630A2BAACF2E7C5E4
1B0D50CDC7B0B6
OC88AD4FE81F8155D002F99EEAF9D95477E4749870C67FDE870E154ED17809C359777F9
79E269010823FB
```



```
981B2A998915EB1439FE3C6C1542A239C', XS_PRINCIPAL.XS_SHA512);
END;
/ 2 3 4 5
PL/SQL procedure successfully completed.
```

For this procedure to complete successfully, both the verifier value and its type must match the information in the VERIFIER column of the XS\$VERIFIERS dictionary table for the user whose verifier is being set. Note that when you change the password for an application user, it automatically changes its verifier value with the option of changing its verifier type.

This example set the verifier to its same exact value to show the steps involved. You have the option to set the verifier for a password to any verifier value that displays for an application user when you query the XS\$VERIFIERS dictionary table as long as the verifier value matches the verifier type that you set. For example, if you wanted to change the verifier value and the verifier type to XS SALTED SHA1, do the following.

```
SQL> BEGIN
SYS.XS_PRINCIPAL.SET_VERIFIER('lwuser1','S:14DC0F5ABB72FC869549B1F845C548E0BE
F7B863A116DB24DFAE22F0501E',
XS_PRINCIPAL.XS_SALTED_SHA1);
END;
/ 2 3 4
PL/SQL procedure successfully completed.
```

Note that this is the same verifier value and verifier type that was set for application user LWUSER3 as shown in the SET VERIFIER Procedure.

## 2.1.3.4 Oracle Label Security Context Is Established in Direct Logon Session

Describes Oracle Label Security support for Real Application Security users.

Beginning with Oracle Database 12c Release 2 (12.2), Oracle Label Security supports Real Application Security users. This means that when a Real Application Security user attaches with Real Application Security user session through direct logon, the user can exercise its own Oracle Label Security authorization. Oracle Label Security context is established during the attach session.

### See Also:

- Attaching an Application Session to a Traditional Database Session for more Information about how Oracle Label Security supports Real Application Security users
- Oracle Label Security Administrator's Guide for more Information about Oracle Label Security



# 2.1.4 Resetting the Application User's Password with the SQL\*Plus PASSWORD Command

As the security administrator, <code>sec\_mgr</code>, you have the <code>create session</code> database privilege and Real Application Securityxs\_session\_admin database role and in addition, <code>sec\_mgr</code> is granted the Real Application Security <code>PROVISION</code> least system privilege as a database user. As the security manager, you can now create users and roles, set passwords, and so forth, and administer sessions using the Real Application Security least system privilege. Example 2-2 shows how the security administrator can reset the password for user <code>lwuser2</code> using the SQL\*Plus <code>PASSWORD</code> command.

However, if you as user <code>lwuser2</code>, perform a self password change using the SQL\*Plus <code>PASSWORD</code> command invoked from an explicitly attached session (a session attached using the <code>ATTACH\_SESSION</code> procedure or the <code>attachSession()</code> method in Java), the session must have the <code>ALTER USER</code> privilege and the user name must be provided with the <code>PASSWORD</code> command.

Example 2-3 shows how the application user lwuser2 explicitly attached to a session, performs a self password change that fails because the users session does not have the ALTER USER privilege.

Example 2-4 shows how an application user <code>lwuser2</code> explicitly attached to a session having the <code>ALTER USER</code> privilege can perform a self password change. The user's self password change is successful.

The SET\_PASSWORD procedure does not prompt for old password, but requires either Real Application Security PROVISION privilege as the least privilege, or database ALTER USER privilege. (Note that SET\_PASSWORD is the Real Application Security PL/SQL procedure, not the SQL\*Plus PASSWORD command.) If the user's session has the PROVISION least privilege or the ALTER USER privilege, you can reset the password for any application user from any application user's session (including an explicitly attached and a direct logon session) or the database user session if that session has the PROVISION least privilege or the ALTER USER privilege. The SQL\*Plus PASSWORD command never prompts for the old password if you are changing another application user's password.

# Example 2-2 DBA Resets the Password with a Password Change Operation for User Iwuser2 When Not Explicitly Attached to a Session

```
sqlplus sec_mgr
Enter password: password
Connected.
SQL> BEGIN
    2 SYS.XS_PRINCIPAL.CREATE_USER('lwuser2');
    3 END;
    4/

PL/SQL orocedure successfully completed.

SQL> PASSWORD lwuser2
Changing password for lwuser2
New password: password
Retype new password: password
Password changed
```



# Example 2-3 User lwuser2 Performs a Self Password Change that Fails When Explicitly Attached to a Session Because the Session Lacks the ALTER USER Privilege

```
sqlplus sec mgr
Enter password: password
Connected.
SOL> DECLARE
 2 SESSIONID RAW(16);
 3 BEGIN
 4 SYS.DBMS XS SESSIONS.CREATE SESSION('lwuser2', sessionid);
 5 SYS.DBMS XS SESSIONS.ATTACH SESSION(sessionid);
  6 END;
  7 /
PL/SQL procedure successfully completed.
SQL> CONNECT lwuser2
Enter password: password
Connected.
SQL> SELECT SYS.XS_SYS_CONTEXT('XS$SESSION','USERNAME') FROM DUAL;
XS SYS CONTEXT('XS$SESSION','USERNAME')
LWUSER2
SQL> PASSWORD lwuser2
Changing password for lwuser2
Old password: password
New password: password
Retype new password: password
ERROR.
ORA-01031: insufficient privileges
```

Password unchanged

# Example 2-4 A Self Password Change Succeeds When Explicitly Attached to a Session and User Iwuser2's Session Has the ALTER USER Privilege

```
sqlplus sec_mgr
Enter password: password
Connected.
SQL> CREATE ROLE pwdchg;
Role created.

SQL> GRANT ALTER USER TO pwdchg;
Grant succeeded.

SQL> EXEC SYS.XS_PRINCIPAL.CREATE_ROLE(NAME => 'resetpwd_role', ENABLED => TRUE);
PL/SQL procedure successfully completed.

SQL> GRANT pwdchg TO resetpwd_role;
Grant succeeded.

SQL> EXEC SYS.XS PRINCIPAL.GRANT ROLES('lwuser2','resetpwd role');
```

# 2.1.5 Configuring an Application User Switch

Using the  $XS\_PRINCIPAL.ADD\_PROXY\_USER$  procedure, you can add an application user to proxy another application user and assume the application roles of that application user. You can use the <code>DBMS\_XS\_SESSIONS.SWITCH\_USER</code> procedure to switch application users in a session if the user has been added as a proxy.

Assume app\_user1 has application roles role1 and role2. Example 2-5 allows you to proxy the application roles role1 and role2 of app\_user1 to app\_user2. The call add\_proxy\_user('app\_user1', 'app\_user2', pxy\_roles) allows app\_user2 to switch to app\_user1 and assume app\_user1's roles, role1 and role2. It does not grant the roles to app\_user2.

The query of view DBA\_XS\_ROLE\_GRANTS shows that roles, roles1 and roles2 are still only granted to app\_user1 and not to app\_user2, and that app\_user2 only assumed these roles as a proxy user.

The query of view DBA\_XS\_PROXY\_ROLES shows that app\_user2 is the proxy user, app\_user1 is the target user, and the target roles are role1 and role2.

The query of view DBA\_XS\_SESSIONS also shows that app\_user2 is the proxy user in this session.

As the application user with DBA role, you can create a session for <code>app\_user2</code> and switch application user to <code>app\_user1</code>, as shown in Example 2-6.

This example first creates a session with app\_user2 and attaches to it. Then app\_user2 switches to app\_user1 and assumes app\_user1's roles, role1 and role2.

The query of view DBA\_XS\_ROLE\_GRANTS shows that roles, roles1 and roles2 are still only granted to app\_user1 and not to app\_user2, and that app\_user2 only assumed these roles as a proxy user.

The query of view  $\mbox{DBA}_{XS}_{SESSION}_{ROLES}$  shows that roles  $\mbox{role1}$  and  $\mbox{role2}$  are associated with the same session ID in which  $\mbox{app}_{user1}$  was switched with  $\mbox{app}_{user2}$ .



The query of view DBA\_XS\_SESSIONS also shows that app\_user2 is the proxy user in this session.

### **Example 2-5 Configuring a Proxy Application User**

```
sqlplus sec mgr
Enter password: password
Connected.
SQL> EXEC SYS.XS PRINCIPAL.CREATE ROLE('role1', true);
SQL> EXEC SYS.XS PRINCIPAL.CREATE ROLE('role2', true);
SQL> EXEC SYS.XS PRINCIPAL.CREATE USER('app user1', 'HR');
SQL> EXEC SYS.XS PRINCIPAL.SET PASSWORD('app user1', 'password');
SQL> EXEC SYS.XS PRINCIPAL.CREATE USER('app user2','HR');
SQL> EXEC SYS.XS_PRINCIPAL.SET_PASSWORD('app_user2', 'password');
SQL> EXEC SYS.XS PRINCIPAL.GRANT ROLES('app user1', 'role1');
SQL> EXEC SYS.XS PRINCIPAL.GRANT ROLES('app user1', 'role2');
DECLARE
 pxy roles XS$NAME LIST;
begin
 pxy roles := XS$NAME LIST('role1', 'role2');
 sys.xs principal.add proxy user(target user => 'app user1', proxy user =>
'app user2', target roles => pxy roles);
end;
SQL> SELECT grantee, granted role FROM DBA XS ROLE GRANTS;
SQL> SELECT proxy user, target user, target role FROM DBA XS PROXY ROLES;
SQL> SELECT user name, sessionid, proxy user FROM DBA XS SESSIONS;
```

#### Example 2-6 Creating a Session and Switching an Application User

```
sqlplus sec mgr
Enter password: password
Connected.
SQL> EXEC SYS.XS_PRINCIPAL.CREATE_USER('app_user1','HR');
SQL> EXEC SYS.XS PRINCIPAL.SET PASSWORD('app user1', 'password');
SQL> EXEC SYS.XS PRINCIPAL.CREATE USER('app user2','HR');
SQL> EXEC SYS.XS PRINCIPAL.SET PASSWORD('app user2', 'password');
SQL> EXEC SYS.XS PRINCIPAL.CREATE ROLE('role1', true);
SQL> EXEC SYS.XS PRINCIPAL.CREATE ROLE('role2', true);
SQL> EXEC SYS.XS PRINCIPAL.CREATE USER('app user1', 'HR');
SQL> EXEC SYS.XS PRINCIPAL.SET PASSWORD('app user1', 'password');
SQL> EXEC SYS.XS PRINCIPAL.CREATE USER('app user2', 'HR');
SQL> EXEC SYS.XS PRINCIPAL.SET PASSWORD('app user2', 'password');
SQL> EXEC SYS.XS PRINCIPAL.GRANT ROLES('app user1', 'role1');
SQL> EXEC SYS.XS PRINCIPAL.GRANT ROLES('app user1', 'role2');
declare
 sessionid raw(16);
 sys.dbms xs sessions.create session('app user2', sessionid);
 sys.dbms xs sessions.attach session(sessionid);
```



```
sys.dbms_xs_sessions.switch_user('app_user1');
end;
/

SQL> SELECT grantee, granted_role FROM DBA_XS_ROLE_GRANTS;

SQL> SELECT sessionid, role FROM DBA_XS_SESSION_ROLES;

SQL> SELECT user name, sessionid, proxy user FROM DBA XS SESSIONS;
```

# 2.1.6 Validating an Application User

Oracle recommends that you always validate the Real Application Security objects after administrative configuration changes. The XS\_DIAG package provides a set of validation APIs to help ensure that these changes do not damage the complicated relationships among your Real Application Security objects. To validate an application user account, use the XS\_DIAG.VALIDATE\_PRINCIPAL function. The caller has invoker's rights on this package and must have ADMIN\_ANY\_SEC\_SECURITY privilege to run the XS\_DIAG package.

See the "VALIDATE\_PRINCIPAL Function" for more information.

# 2.2 About Configuring Application Roles

This section contains the following topics:

- · About Application Roles
- Regular and Dynamic Application Roles
- About Configuring an Application Role
- Predefined Regular Application Roles and Dynamic Application Roles

# 2.2.1 About Application Roles

An application role is a role that can only be granted to an application user or to another application role. Application roles provide a way to group application users who must have a common application privilege, identified within an ACL, in order to access an application. The XS\_PRINCIPAL.CREATE\_ROLE procedure can create regular application roles. The XS\_PRINCIPAL.CREATE\_DYNAMIC\_ROLE procedure can create dynamic application roles (one type of application role).

Application roles are conceptually similar to enterprise roles. An enterprise role can only be granted to an enterprise user and that grant occurs outside the database. Similarly, an application role can only be granted to an application user or application role, and that grant occurs outside of the standard database grant mechanisms. Dynamic roles cannot be granted to an application user or another application role, but can only be enabled in an application session as a parameter in an attach session call as described in "Dynamic Application Roles".



### See Also:

- Oracle Database SQL Language Reference for more information about SQL
- Oracle Database PL/SQL Language Reference for more information about PL/SQL APIs

# 2.2.2 Regular and Dynamic Application Roles

Real Application Security allows regular and dynamic application roles.

This section contains the following topics:

- Regular Application Roles
- · Dynamic Application Roles

### 2.2.2.1 Regular Application Roles

A regular application role is an application role that you can grant to an application user or another application role (regular or dynamic). You can specify if you want the regular application role to be enabled by default or not.

## 2.2.2.2 Dynamic Application Roles

A dynamic application role is an application role that is enabled only under certain situations, for example, when a user has logged on using SSL, or during a specific period of time, and so on. Dynamic application roles might be used, for example, if there is some application privilege granted to all application users connecting during weekdays. If that criterion is met, then the application enables those application roles.

The application determines the criteria for enabling a dynamic application role, however the criteria can be evaluated by the application or by the database at the request of the application.

· When the Application Evaluates the Criteria

If the application evaluates the criteria and the application role meets it, then the application, if it is attached to an application session, can enable dynamic application roles for application users. When the application detaches from the application session, the dynamic application role is automatically disabled.

For security reasons, you cannot disable dynamic application roles during the session. This is especially important because they may infer negative application privileges.

When the Database Evaluates the Criteria

If the database evaluates the criteria and the application role meets it, then the database can enable application roles for the application user. The database can disable dynamic application roles based on two types of time-outs: one from the last time the session was accessed, and one from the last time the session was authenticated. Oracle Database checks these time-outs when the session is first attached.

You do not need to grant the dynamic application role formally to a user beforehand. There is no way to enable or disable a dynamic application role through the standard enable and



disable APIs. You cannot grant dynamic application roles to other application roles, but you can grant other application roles to dynamic roles.



"Predefined Regular Application Roles and Dynamic Application Roles"

# 2.2.3 About Configuring an Application Role

This section contains the following topics:

- Creating a Regular Application Role
- Creating a Dynamic Application Role
- Validating an Application Role

### 2.2.3.1 Creating a Regular Application Role

To create a regular application role, log into SQL\*Plus as user <code>sec\_mgr</code> with the <code>CREATE</code> ROLE system privilege, and then use the <code>XS PRINCIPAL.CREATE</code> ROLE procedure.

Example 2-7 shows how to create a regular application role called <code>app\_regular\_role</code>. The <code>start\_date</code> and <code>end\_date</code> parameters specify the active start and end times for this application role. The <code>enable</code> parameter is set to <code>TRUE</code>.

After you create the regular application role, you are ready to grant it to one or more application users or application roles. See the following section:

"About Granting an Application Role to an Application User"

### **Example 2-7 Creating a Regular Application Role**

### 2.2.3.2 Creating a Dynamic Application Role

To create a dynamic application role, log into SQL\*Plus as user <code>sec\_mgr</code> with the <code>CREATE ROLE</code> system privilege and then use the <code>XS\_PRINCIPAL.CREATE\_DYNAMIC\_ROLE</code> procedure.

Example 2-8 shows how to create a dynamic application role called <code>app\_dynamic\_role</code>. The optional <code>duration</code> parameter specifies the period of time (in minutes) the application role is active. The <code>scope</code> parameter specifies the scope for this role, which can be either <code>SESSION\_SCOPE</code> (the default value) or <code>REQUEST\_SCOPE</code>. <code>SESSION\_SCOPE</code> means the enabled dynamic role is still enabled when you detach the session and attach to the session again, unless you explicitly specify that it be disabled in the session reattach. <code>REQUEST\_SCOPE</code> means that the role is disabled after the session is detached.

In this example, the dynamic application role is active for 40 minutes, and the scope is set to <code>SESSION\_SCOPE</code> for the current application session. So the dynamic application role is active even when you detach the session and attach to the session again as long as the time limit has not exceeded 40 minutes after having created the dynamic application role.

### Example 2-8 Creating a Dynamic Application Role

### 2.2.3.3 Validating an Application Role

Oracle recommends that you should always validate Real Application Security objects after administrative configuration changes. The XS\_DIAG package provides a set of validation APIs to help ensure that these changes do not damage the complicated relationships among your Real Application Security objects. To validate an application role, use the XS\_DIAG.VALIDATE\_PRINCIPAL function. See the "VALIDATE\_PRINCIPAL Function" for more information.

See Troubleshooting Oracle Database Real Application Security for troubleshooting advice.

# 2.2.4 Predefined Regular Application Roles and Dynamic Application Roles

Using predefined dynamic application roles in a Real Application Security session, application users can acquire application privileges based on their run-time states. These application roles cannot be acquired by grants.

As an example, an application role may be enabled for application users connecting from within the corporate firewall, which grants application users more application privileges than connecting from outside the firewall.

See "Roles" for a description of Real Application Security predefined regular application roles, dynamic application roles, and database roles.

Regular application roles can be granted to an application user, but dynamic application roles cannot. Dynamic application roles are enabled based on user state.

See "Regular and Dynamic Application Roles" for descriptions.

# 2.3 Effective Dates for Application Users and Application Roles

You can specify effective dates for application users, application roles, and role grants. The application user or application role is available only within the period defined by the effective start and end date. Example 2-9 shows how effective dates are specified for an application user.

Sometimes the effective date restriction does not need to be an attribute of an application user or application role. Instead, it is only needed to restrict the effective dates on a per role grant basis. In this case, you can specify beginning and ending effective dates for an application role grant. This only constrains that particular application role grant and allows for implementing fine-grained access control policy. Example 2-10 shows how effective dates are specified for an application role.

These are the most direct consequences of effective date restrictions:

- If an application user is not currently effective (that is, within the period defined by its start and end date), the session for the particular application user cannot be created.
- If an application role is not currently effective, the application role (and any descendants) is not be available to the application user in the session.
- For application roles that are shared children of multiple application roles, the child application roles are available as long as there is at least one parent that is effective.
- If the application role grant of an application role is not currently effective, the
  application role (and any descendants) is not available to the application user or
  application role to which it is granted.



The effective dates should be used in the policy after a careful consideration of the nature of the restrictions that they impose on the use of application users and application roles.

### **Example 2-9 Setting Effective Dates for an Application User**



```
end_date => endDate);
END;
/
```

### Example 2-10 Setting Effective Dates for an Application Role of an Application User

```
sqlplus sec_mgr
Enter password: password
Connected.

DECLARE
  startDate TIMESTAMP := TO_TIMESTAMP ('2012-01-01 11:00:00','YYYY-MM-DD
  HH:MI:SS');
  endDate TIMESTAMP := TO_TIMESTAMP ('2013-01-01 11:00:00','YYYY-MM-DD
  HH:MI:SS');
BEGIN
  SYS.XS_PRINCIPAL.GRANT_ROLES
   (grantee => 'lwuser1',
    role => 'app_regular_role',
    start_date => startDate,
    end_date => endDate);
END;
//
```

# 2.4 About Granting Application Privileges to Principals

This section contains the following topics:

- · About Granting an Application Role to an Application User
- Granting an Application Role to Another Application Role
- Granting a Database Role to an Application Role

# 2.4.1 About Granting an Application Role to an Application User

This section contains the following topics:

## 2.4.1.1 Creating a New Application User and Granting This User an Application Role

Example 2-11 shows how to grant an application role, appl1\_regular\_role, to an application user, lwuser1, when the application user account is created.

To find a listing of existing application roles, query the DBA XS ROLES data dictionary view.

# Example 2-11 Creating a New Application User and Granting This User an Application Role

```
sqlplus sec_mgr
Enter password: password
Connected.

BEGIN
    SYS.XS_PRINCIPAL.CREATE_USER('lwuser1');
    SYS.XS_PRINCIPAL.GRANT_ROLES('lwuser1', 'appl1_regular_role');
END;
//
```



## 2.4.1.2 Granting an Application Role to an Existing Application User

Example 2-12 shows how to grant an application role, appl1\_regular\_role, to an existing application user, lwuser1. You cannot grant dynamic application roles to an existing application user.

You can find a listing of existing application user accounts by querying the  ${\tt DBA\_XS\_USERS}$  view.

### Example 2-12 Granting an Application Role to an Existing Application User

```
sqlplus sec_mgr
Enter password: password
Connected.

BEGIN
    SYS.XS_PRINCIPAL.GRANT_ROLES('lwuser1', 'appl1_regular_role');
END;
//
```

# 2.4.2 Granting an Application Role to Another Application Role

Example 2-13 shows how to grant a regular application role to another regular application role. You cannot grant dynamic application roles to other regular application roles, but you can grant other regular application roles to dynamic application roles. To find a listing of existing application roles, query the DBA\_XS\_ROLES view (see "DBA\_XS\_ROLES").

# **Example 2-13 Granting a Regular Application Role to Another Regular Application Role**

```
sqlplus sec_mgr
Enter password: password
Connected.

BEGIN
    SYS.XS_PRINCIPAL.GRANT_ROLES(grantee => 'app_regular_role', role => 'appl1_regular_role');
END;
/
```

## 2.4.3 Granting a Database Role to an Application Role

To grant a database role to an application role, use the SQL GRANT statement. You can find a listing of existing database roles by querying the DBA ROLES data dictionary view.

Example 2-14 shows how to grant the database role, cust\_role, to the application role app regular role.

### Example 2-14 Granting a Database Role to an Application Role

```
sqlplus sec_mgr
Enter password: password
Connected.
GRANT cust role TO app regular role;
```



# **Configuring Application Sessions**

### This chapter contains:

- About Application Sessions
- About Creating and Maintaining Application Sessions
- About Manipulating the Application Session State
- About Administrative APIs for External Users and Roles
- About Real Application Security Session Privilege Scoping Through ACL

# 3.1 About Application Sessions

An application session contains information relevant to the application and its user. An application session stores application session state as a collection of attribute-value pairs. These attribute value pairs are divided into namespaces. Unlike traditional *heavyweight* database sessions, an application session does not hold its own database resources, such as transactions and cursors. Because application sessions consume far fewer server resources than heavyweight sessions, an application session can be dedicated to each end application user. An application session can persist in the database and resume later with minimal cost.

To configure an application session, you work in two phases:

- 1. You create and maintain the application session.
- 2. You can manipulate the session state during the life of the session.

You can use either PL/SQL APIs or Java APIs to configure application sessions. This chapter describes the programmatic creation, use, and maintenance of application sessions in PL/SQL, and includes specific links to comparable Java information.

### See Also:

- Oracle Database Real Application Security SQL Functions and Oracle Database Real Application Security PL/SQL Packages for information about PL/SQL API syntax
- Oracle Database Real Application Security Java API Reference for information about Java API syntax (in Javadoc format)
- Using Real Application Security in Java Applications for information about performing tasks with Java APIs

#### This section contains:

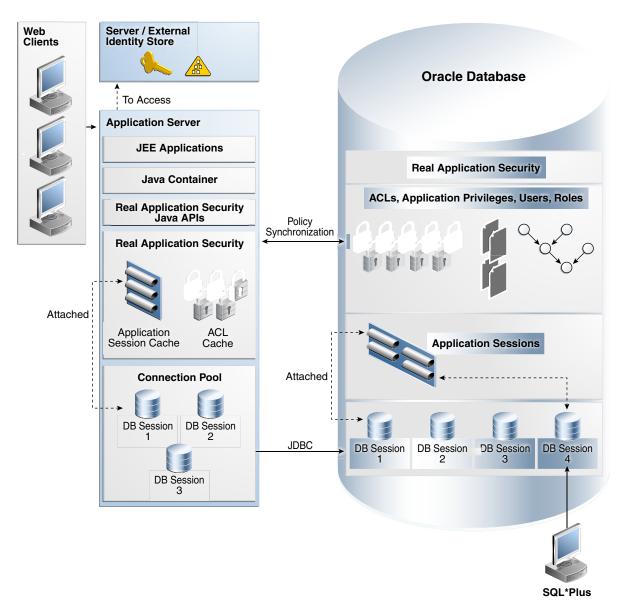
- About Application Sessions in Real Application Security
- Advantages of Application Sessions

# 3.1.1 About Application Sessions in Real Application Security

Figure 3-1 shows a Real Application Security architecture diagram and indicates how application sessions fit into it. The figure shows applications creating application sessions in the database. Some of these application sessions are associated with traditional database (DB) sessions.

Figure 3-1 also shows other components of Real Application Security such as ACLs, application privileges, application users, and application roles.

Figure 3-1 Real Application Security Architecture





# 3.1.2 Advantages of Application Sessions

Application sessions have functional advantages over traditional database sessions. For example, traditional database sessions are typically unaware of the end user identities or the security policies for those end users. On the contrary:

- Application sessions encapsulate end user's security context. They enable applications to use database authorization mechanisms for access control based on the end user identity.
- An application session can be associated with multiple database sessions simultaneously.
- They are accessible by all nodes in an Oracle Real Application Clusters (Oracle RAC) environment.

Application sessions have these performance advantages over traditional database sessions:

- They can be created with less overhead than traditional database sessions.
- They can persist in the database and resume later with minimal cost.
- Real Application Security can collect session attribute changes and session states on the client, using caches. Then, these changes are appended to the database until the next database roundtrip, reducing the number of database roundtrips.

# 3.2 About Creating and Maintaining Application Sessions

This section contains:

- Creating an Application Session
- Creating an Anonymous Application Session
- Attaching an Application Session to a Traditional Database Session
- Setting a Cookie for an Application Session
- Assigning an Application User to an Anonymous Application Session
- Switching a Current Application User to Another Application User in the Current Application Session
- About Creating a Global Callback Event Handler Procedure
- Configuring Global Callback Event Handlers for an Application Session
- Saving an Application Session
- Detaching an Application Session from a Traditional Database Session
- Destroying an Application Session

## 3.2.1 Creating an Application Session

You can create an application session using the DBMS\_XS\_SESSIONS.CREATE\_SESSION procedure in PL/SQL or using the createSession method of the XSSessionManager class in Java. To create an application session, the invoking user needs CREATE\_SESSION application privilege. This privilege can be obtained through XS\_SESSION\_ADMIN Database role or by XS\_ADMIN\_UTIL.GRANT\_SYSTEM\_PRIVILEGE API call (see "GRANT\_SYSTEM\_PRIVILEGE Procedure" for more information). CREATE SESSION procedure populates the unique identifier



of the newly created session in sessionid out parameter. This unique identifier can be used to refer to the session in future calls. The DBA\_XS\_SESSIONS data dictionary view displays all the application sessions in the database.

You can also specify a list of namespaces to be created when the session is created. If you specify namespaces during creation of the session, the caller must have application privileges <code>MODIFY\_NAMESPACE</code> or <code>MODIFY\_ATTRIBUTE</code> on the namespaces, or the <code>ADMIN\_NAMESPACE</code> system privilege.

Example 3-1 shows how to create an application session with lwuser1.

### **Example 3-1 Creating an Application Session**

```
DECLARE
   sessionid RAW(16);
BEGIN
   SYS.DBMS_XS_SESSIONS.CREATE_SESSION('lwuser1', sessionid);
END;
```

### See Also:

- CREATE\_SESSION Procedure for information about the syntax of this PL/SQL procedure
- Oracle Database Real Application Security Java API Reference for information about the syntax of the Java createSession method (in Javadoc format)
- Example 6-2 for information about a Java example of this task

## 3.2.2 Creating an Anonymous Application Session

You can also create an anonymous application session using the <code>DBMS\_XS\_SESSIONS.CREATE\_SESSION</code> procedure in PL/SQL or using the <code>createAnonymousSession</code> method of the <code>XSSessionManager</code> class in Java. To create an anonymous session through the PL/SQL API, you must specify the predefined user <code>name</code> <code>XSGUEST</code>.

**Example 3-2** shows how to create an anonymous session with the predefined user XSGUEST.

After creating an anonymous application session, you can assign a named user to the session.

### **Example 3-2** Creating an Anonymous Application Session

```
DECLARE
sessionid RAW(16);
BEGIN
SYS.DBMS_XS_SESSIONS.CREATE_SESSION('XSGUEST', sessionid);
END:
```



### See Also:

- CREATE\_SESSION Procedure for information about the syntax of this PL/SQL procedure
- Oracle Database Real Application Security Java API Reference for information about the syntax of the Java createAnonymousSession method (in Javadoc format)
- Example 6-2 for information about a Java example of this task

# 3.2.3 Attaching an Application Session to a Traditional Database Session

To use an application session, it must be associated with a database session. This operation is called <code>attach</code>. You can attach an application session to a traditional database session using the <code>DBMS\_XS\_SESSIONS.ATTACH\_SESSION</code> procedure in PL/SQL or the <code>attachSession</code> method of the <code>XSSessionManager</code> class in Java. A database session can only attach one application session at a time. The <code>DBA\_XS\_ACTIVE\_SESSIONS</code> dynamic data dictionary view displays all attached application sessions in the database.

To execute this procedure, the traditional session user must have the ATTACH\_SESSION application privilege. This privilege can be obtained through the XS\_SESSION\_ADMIN Database role or by the XS\_ADMIN\_UTIL.GRANT\_SYSTEM\_PRIVILEGE API call. If you specify namespaces, then the user is required to have the application privileges MODIFY\_NAMESPACE or MODIFY ATTRIBUTE on the namespaces, or ADMIN NAMESPACE system privilege.

Example 3-3 shows how to attach an application session to a database session.

Beginning with Oracle Database 12c Release 2 (12.2), Oracle Label Security supports Real Application Security users. This means that Oracle Label Security context is established in the Real Application Security session during the attach operation, so that Oracle Label Security authorization can be exercised in the Real Application Security user session. Oracle Label Security provides the ability to define data labels, assign user labels and protect sensitive application data within the Oracle database.

For example, using Oracle Label Security data labels allows each row of a table to be labeled based on its level of confidentiality. Data labels consist of 3 components: levels, compartments, and groups. So a given data label should have one level, zero or more compartments and zero or more groups associated with it. Compartments allow defining finer granularity within a level – all data belonging to a particular project can be labeled with the same compartment. Groups are hierarchical and a group can thus be associated with a parent group.

In addition, using Oracle Label Security user labels, each user can be assigned labels that constrain access to labeled data. Each user is assigned a range of levels, compartments, and groups, and each session can operate within that authorized range to access labeled data within that range.

Furthermore, using privileges, Oracle Label Security privileges are policy specific and used to provide users specific rights to perform special operations or to access data beyond their label authorizations. The list of all policy specific privileges is: FULL, READ, COMPACCESS, PROFILE ACCESS, WRITEUP, WRITEDOWN, and WRITEACROSS.



Using Oracle Label Security, trusted stored programs can be used. A trusted stored program unit is a stored procedure, function, or package that has been granted one or more label security privileges. Trusted stored program units are used to let users perform privileged operations in a controlled manner, or update data at several labels. By granting privileges to a program unit, the privileges required for users can be effectively reduced.

Using Oracle Label Security, a policy is applied to a table or an entire schema after defining data labels or user labels or both and assigning appropriate privileges to users. When a policy is applied on a table, label security creates a policy specific NUMBER column on the table to store numeric equivalent of the data labels defined before for the policy. The column can be created as a user visible column or as a hidden column. The user can specify various enforcement options when the policy is applied on the table. The READ\_CONTROL enforcement option for example protects the table from queries and WRITE CONTROL protects it from DML operations.

Establishing Oracle Label Security context in a Real Application Security session therefore lets SELECT and DML operations return results authorized for the Real Application Security user.

To attach a session with dynamic roles, a list of dynamic roles can be passed in the PL/SQL ATTACH SESSION procedure.



When developing the application, ensure that all application end user actions are captured within an ATTACH\_SESSION ... DETACH\_SESSION programming block. (For more information, see "Detaching an Application Session from a Traditional Database Session").

#### **Example 3-3** Attaching an Application Session

```
DECLARE
   sessionid raw(16);
BEGIN
   SYS.DBMS_XS_SESSIONS.CREATE_SESSION('lwuserl', sessionid);
   SYS.DBMS_XS_SESSIONS.ATTACH_SESSION(sessionid);
END;
```

### See Also:

- ATTACH\_SESSION Procedure for information about the syntax of this PL/SQL procedure
- Oracle Database Real Application Security Java API Reference for information about the syntax of the Java attachSession method (in Javadoc format)
- Example 6-3 for information about a Java example of this task
- Oracle Label Security Administrator's Guide for information about Oracle Label Security



# 3.2.4 Setting a Cookie for an Application Session

You can associate a specific cookie with an application session using the <code>DBMS\_XS\_SESSIONS.SET\_SESSION\_COOKIE</code> procedure in PL/SQL or the <code>setCookie</code> method of the <code>XSSessionManager</code> class in Java. The cookie can also be associated at the time of creation of the session through the <code>CREATE\_SESSION</code> PL/SQL API. A cookie is a token embedded in a user's session by a web site during an application session. So the next time the same user requests something from that web site, it sends the cookie to the application session, which allows the server to associate the session with that user.

To execute this procedure, the user must be granted the <code>MODIFY\_SESSION</code> application privilege. This privilege can be obtained through the <code>XS\_SESSION\_ADMIN</code> Database role or by the <code>XS\_ADMIN\_UTIL.GRANT\_SYSTEM\_PRIVILEGE API call.</code>

Example 3-4 shows how to set a cookie for an application session.

### Example 3-4 Setting a Cookie for an Application Session

```
DECLARE
   sessionid raw(16);
BEGIN
   SYS.DBMS_XS_SESSIONS.CREATE_SESSION('lwuserl', sessionid);
   SYS.DBMS_XS_SESSIONS.SET_SESSION_COOKIE('Cookiel', sessionid);
END;
```

### See Also:

- SET\_SESSION\_COOKIE Procedure for information about the syntax of this PL/SQL procedure
- Oracle Database Real Application Security Java API Reference for information about the syntax of the Java setCookie method (in Javadoc format)
- Example 6-20 for information about a Java example of this task

# 3.2.5 Assigning an Application User to an Anonymous Application Session

You can assign a named application user to a currently attached anonymous application session using the <code>DBMS\_XS\_SESSIONS.ASSIGN\_USER</code> procedure in PL/SQL or the <code>assignUser</code> method of the <code>XSSessionManager</code> class in Java. Assigning a user changes the user session from anonymous to a named user.

To execute this procedure, the dispatcher or connection user must have the <code>ASSIGN\_USER</code> application privilege. This privilege can be obtained through the <code>XS\_SESSION\_ADMIN</code> Database role or by the <code>XS\_ADMIN\_UTIL.GRANT\_SYSTEM\_PRIVILEGE</code> API call. If you specify namespaces, then the user is required to be granted application privileges <code>MODIFY\_NAMESPACE</code> or <code>MODIFY\_ATTRIBUTE</code> on the namespaces, or <code>ADMIN\_NAMESPACE</code> system privilege. A list of dynamic roles can also be enabled using the <code>DBMS\_XS\_SESSIONS.ASSIGN\_USER</code> procedure.

Beginning with Oracle Database 12c Release 2 (12.2), Oracle Label Security supports Real Application Security users. If the Real Application Security user is authorized in any Oracle Label Security policy then, during an assign\_user call, the corresponding label security authorization is established for the named Real Application Security user session.

Establishing Oracle Label Security context in a Real Application Security session therefore lets SELECT and DML operations return results authorized for the Real Application Security user.

Example 3-5 shows how to assign the application user lwuser1 to an application session.

### **Example 3-5** Assigning an Application User to an Application Session

```
DECLARE
sessionid raw(16);
BEGIN
SYS.DBMS_XS_SESSIONS.CREATE_SESSION('XSGUEST', sessionid);
SYS.DBMS_XS_SESSIONS.ATTACH_SESSION(sessionid);
SYS.DBMS_XS_SESSIONS.ASSIGN_USER('lwuser1');
END:
```

### See Also:

- ASSIGN\_USER Procedure for information about the syntax of this PL/SQL procedure
- Oracle Database Real Application Security Java API Reference for information about the syntax of the Java assignUser method (in Javadoc format)
- Example 6-5 for information about a Java example of this task
- Attaching an Application Session to a Traditional Database Session for information about how Oracle Label Security supports Real Application Security users
- Oracle Label Security Administrator's Guide for information about Oracle Label Security

# 3.2.6 Switching a Current Application User to Another Application User in the Current Application Session

You can switch or proxy the security context of the current application session to a newly initialized security context for a specified application user using the <code>DBMS\_XS\_SESSIONS.SWITCH\_USER</code> procedure in PL/SQL or the <code>switchUser</code> method of the <code>Session</code> interface in Java. To proxy another application user, the current application session user must be set up as a proxy user for the target user before performing the switch operation. This is performed through the <code>XS\_PRINCIPAL.ADD\_PROXY\_USER\_PL/SQL\_API</code>.

Switching a user changes the user session between two named users.

If the target application user of the proxy operation has a list of filtering roles (proxy roles) set up for the proxy user, they are enabled in the session.

Beginning with Oracle Database 12c Release 2 (12.2), Oracle Label Security supports Real Application Security users. This means that Oracle Label Security context of the target\_user will be established on switching from the proxy\_user session to the target\_user session.



You can either retain or clear the application namespace and attributes after a switch operation. When the keep\_state parameter is set to TRUE, all application namespaces and attributes are retained; otherwise, all previous state in the session is cleared.

If you specify namespaces, then the user is required to be granted application privileges MODIFY\_NAMESPACE or MODIFY\_ATTRIBUTE on the namespaces, or the ADMIN\_NAMESPACE system privilege.

Example 3-6 shows how to switch the application user <code>lwuser1</code> to application user <code>lwuser2</code> in the current application session. Note that namespace templates <code>ns1</code> and <code>ns2</code> should have already have been created by <code>SYSDBA</code>.

# Example 3-6 Switching an Application User to Another Application User in the Current Application Session

### See Also:

- SWITCH\_USER Procedure for information about the syntax of this PL/SQL procedure
- Oracle Database Real Application Security Java API Reference for information about the syntax of the Java assignUser method (in Javadoc format)
- Example 6-6 for information about a Java example of this task
- Attaching an Application Session to a Traditional Database Session for information about how Oracle Label Security supports Real Application Security users
- Oracle Label Security Administrator's Guide for information about Oracle Label Security

## 3.2.7 About Creating a Global Callback Event Handler Procedure

The callback event handler procedure must adhere to the prototype, which includes a specified set of arguments.

For example, the following <code>callback\_procedure</code> specifies an existing PL/SQL procedure, which is the event handler and shows its two possible forms.

```
PROCEDURE callback procedure (sessionid in raw, error out pls integer)
```



This first form includes two parameters, the sessionid in RAW and the out parameter error, which is used for the purpose of setting the error. The sessionid contains the session ID of the session in which the event was triggered. The out error parameter can be used in the event handler code to display the error.

```
PROCEDURE callback_procedure (sessionid in raw, user in varchar2, error out pls integer)
```

This second form includes an additional parameter user in VARCHAR2 to specify the user who triggered this event.

### Note:

The error value must be explicitly set to a value in the PL/SQL body or in the exception block as follows, error:= 0;.

Otherwise, the following error is raised, ORA-46071: Error occured in event handler <name-of-event-handler> followed by another error, ORA-1405: fetched column value is NULL, indicating that the error value is NULL.

The following example shows the explicit setting of the error value using the second form of the callback procedure.

```
CREATE OR REPLACE PACKAGE CALLBACK PACKAGE AS
PROCEDURE CALLBACK PROCEDURE (sessionid in RAW, user in VARCHAR2,
error out PLS INTEGER);
END CALLBACK PACKAGE;
CREATE OR REPLACE PACKAGE BODY CALLBACK PACKAGE AS
PROCEDURE CALLBACK PROCEDURE (sessionid in RAW, user in VARCHAR2,
error out PLS INTEGER) IS
BEGIN
  error := 0;
 dbms output.put line('Inside callback procedure');
EXCEPTION
WHEN OTHERS THEN
        error:=0;
        dbms output.put line('Error');
END CALLBACK PROCEDURE;
END CALLBACK PACKAGE;
```

### See Also:

Configuring Global Callback Event Handlers for an Application Session



# 3.2.8 Configuring Global Callback Event Handlers for an Application Session

A global callback event handler is a predefined PL/SQL procedure that can be invoked to inspect, log, and modify the session state when certain session events of interest occur. You can add multiple global callback event handlers on a session event. After you create the PL/SQL procedure, you can register or deregister, or enable or disable it using these procedures, respectively:

• DBMS XS SESSIONS.ADD GLOBAL CALLBACK

Use this procedure to register a callback event handler.

• DBMS XS SESSIONS.DELETE GLOBAL CALLBACK

Use this procedure to deregister a global callback.

DBMS XS SESSIONS.ENABLE GLOBAL CALLBACK

Use this procedure to enable or disable a global callback procedure by specifying a value of TRUE for enable or FALSE for disable.

To execute these APIs the user must have the CALLBACK application privilege. This can be obtained through the XSPROVISIONER application role or by calling the XS\_ADMIN\_UTIL.GRANT\_SYSTEM\_PRIVILEGE API. You can configure one or more global callback event handlers for use in an application session. If you configure multiple callback event handlers, Oracle Database executes the handlers in the order in which they were created.

Optionally, you can follow these steps to change the execution order:

- 1. Run the DBMS\_XS\_SESSIONS.DELETE\_GLOBAL\_CALLBACK procedure to deregister any existing callback.
- 2. Run the DBMS XS SESSIONS.ADD GLOBAL CALLBACK procedure to register the callback.

### Example 3-7 Registering a Global Callback in an Application Session

```
BEGIN

SYS.DBMS_XS_SESSIONS.ADD_GLOBAL_CALLBACK

(DBMS_XS_SESSIONS.CREATE_SESSION_EVENT,

'CALLBACK_SCHM','CALLBACK_PKG','CALLBACK_PROC');

END;

//
```

Table 3-1 lists session events that can use callback event handlers.

Table 3-1 Session Events That Can Use Callback Event Handlers

Session Event	When the Callback Will Be Executed
Creating a new application session	After the session is created.
Attaching to an existing application session	After the session is attached.
Enabling a dynamic application role	After a dynamic application role is enabled.
Disabling a dynamic application role	After a dynamic application role is disabled.



Table 3-1 (Cont.) Session Events That Can Use Callback Event Handlers

Session Event	When the Callback Will Be Executed
Direct login of an application session	After the session is attached (if the session attach is called as part of the direct logon of an application session).
Assigning a named application user to an anonymous application session	After the named user is assigned to the anonymous application session.
Proxying from one named application user to another named application user	After the application user is switched (if the application user is not proxying back to the original application user).
Proxying back from a named application user to the original application user	After the application user is switched (if the application user is proxying back to the original application user).
Enabling a regular application role	After the application role is enabled.
Disabling a regular application role	After the application role is disabled.
Detaching from an existing application session or database session	Before the session is detached.
Terminating an existing application session or database session	Before the session is destroyed.
Direct logoff of an application session or database session	Before the session is detached (if the session detach is called as part of the direct logoff of an application session).

Suppose you want to initialize certain application-specific states after creating a session. Example 3-7 shows how to register a global callback that sets up the state CALLBACK\_PROC, which is defined in the package CALLBACK\_PKG and owned by the schema CALLBACK\_SCHM.

The state  ${\tt CALLBACK\_PROC}$  is registered as a global callback for the event  ${\tt CREATE}$  SESSION EVENT.

For more examples, and for details about the syntax of these procedures, see the following:

- "ADD\_GLOBAL\_CALLBACK Procedure"
- "DELETE\_GLOBAL\_CALLBACK Procedure"
- "ENABLE\_GLOBAL\_CALLBACK Procedure"

## 3.2.9 Saving an Application Session

You can save the current user application session using the

DBMS\_XS\_SESSIONS.SAVE\_SESSION procedure in PL/SQL or the saveSession method of the XSSessionManager class in Java. Use the save operation when session changes need to be propagated immediately to other sessions using the same session as this one. If the save operation is not used, then the session changes would be reflected in other sessions only after this session is detached.

The calling user requires no privileges to perform this operation.



Example 3-8 shows how to save the current user application session.

#### **Example 3-8 Saving the Current User Application Session**

```
BEGIN
SYS.DBMS_XS_SESSIONS.SAVE_SESSION;
END;
```

## See Also:

- SAVE\_SESSION Procedure for information about the syntax of these PL/SQL procedures
- Oracle Database Real Application Security Java API Reference for information about the syntax of the Java detachSession method (in Javadoc format)
- Example 7-4 for information about a Java example of this task

# 3.2.10 Detaching an Application Session from a Traditional Database Session

You can detach an application session from the traditional database session using either of these procedures:

• DBMS XS SESSIONS.DETACH SESSION(abort => FALSE)

Use this procedure to detach the session and commit any changes that were made since the last time session changes were saved. When you specify the abort parameter as <code>FALSE</code> (the default value), all changes performed in the current session are persisted. The currently attached user can perform this operation without any additional privileges.

DETACH SESSION is always performed on the currently attached session.

• DBMS XS SESSIONS.DETACH SESSION(abort => TRUE)

Use this procedure to detach the session without saving the changes. When you specify the abort parameter as TRUE, it rolls back the changes performed in the current session. The role and namespace changes made to the session since the attach are discarded.

Example 3-9 shows how to detach an application session from a database session and commit the changes. Note that you can call <code>DETACH\_SESSION</code> anywhere to detach the currently attached session.

You can use the detachSession method of the XSSessionManager class in Java.

Example 3-10 shows how to detach a database session from an application session without saving any changes.





When developing the application, ensure that all application end user actions are captured within an ATTACH\_SESSION ... DETACH\_SESSION programming block. (For more information, see "Attaching an Application Session to a Traditional Database Session")

## **Example 3-9 Detaching and Committing an Application Session**

```
DECLARE
  sessionid RAW(16);
BEGIN
  SYS.DBMS_XS_SESSIONS.CREATE_SESSION('lwuser1', sessionid);
  SYS.DBMS_XS_SESSIONS.ATTACH_SESSION(sessionid);
...
  SYS.DBMS_XS_SESSIONS.DETACH_SESSION;
...
END:
```

#### **Example 3-10** Detaching and Not Committing an Application Session

```
DECLARE
   sessionid RAW(16);
BEGIN
   SYS.DBMS_XS_SESSIONS.CREATE_SESSION('lwuserl', sessionid);
   SYS.DBMS_XS_SESSIONS.ATTACH_SESSION(sessionid);
...
   SYS.DBMS_XS_SESSIONS.DETACH_SESSION(TRUE);
END;
```

## See Also:

- DETACH\_SESSION Procedure for information about the syntax of these PL/SQL procedures
- Oracle Database Real Application Security Java API Reference for information about the syntax of the Java detachSession method (in Javadoc format)
- Example 6-21 for information about a Java example of this task

## 3.2.11 Destroying an Application Session

You can terminate an application session using the

DBMS\_XS\_SESSIONS.DESTROY\_SESSION procedure in PL/SQL or using the destroySession method of the XSSessionManager class in Java. This procedure also detaches all traditional sessions from the application session.

To execute this procedure, the invoking user must have the <code>TERMINATE\_SESSION</code> application privilege. This privilege can be obtained through the <code>XS\_SESSION\_ADMIN</code> Database role or by the <code>XS\_ADMIN\_UTIL.GRANT\_SYSTEM\_PRIVILEGE</code> API call.

Example 3-11 shows how to destroy an application session.

#### Example 3-11 Destroying an Application Session

```
DECLARE
sessionid RAW(16);
BEGIN
SYS.DBMS_XS_SESSIONS.CREATE_SESSION('lwuserl', sessionid);
SYS.DBMS_XS_SESSIONS.ATTACH_SESSION(sessionid);
SYS.DBMS_XS_SESSIONS.DETACH_SESSION;
SYS.DBMS_XS_SESSIONS.DESTROY_SESSION(sessionid);
END:
```

## See Also:

- DESTROY\_SESSION Procedure for information about the syntax of this PL/SQL procedure
- Oracle Database Real Application Security Java API Reference for information about the syntax of the Java destroySession method (in Javadoc format)
- Example 6-22 for information about a Java example of this task

# 3.3 About Manipulating the Application Session State

#### This section contains:

- About Using Namespace Templates to Create Namespaces
- Initializing a Namespace in an Application Session
- Setting Session Attributes in an Application Session
- Getting Session Attributes in an Application Session
- Creating Custom Attributes in an Application Session
- Deleting a Namespace in an Application Session
- Enabling Application Roles for a Session
- Disabling Application Roles for a Session

## 3.3.1 About Using Namespace Templates to Create Namespaces

An application uses a namespace to store application defined attribute-value pairs. Often, an application needs to use the same namespace across different application sessions. A namespace template provides a way to define and initialize a namespace.

A namespace template defines the namespace and its properties. It is used to initialize the namespace in an application session. The namespace name must be the same as the template that defines it.

#### This section contains:

- Components of a Namespace Template
- About Namespace Views
- Creating a Namespace Template for an Application Session



## 3.3.1.1 Components of a Namespace Template

A namespace template includes the following:

Name of the namespace

The name of the application namespace uniquely identifies the namespace. This name is used when creating the namespace in an application session.

Namespace handler

The namespace handler is called when an attribute value is set or retrieved. Specifying a handler is optional.

Namespaces can be associated with an event handling function. The server invokes this function whenever an operation on an attribute registered for event handling is performed. The event handling function is provided with the attribute name, attribute value, and the event code as arguments. For example:

```
FUNCTION event_handling_function_name(
    session_id IN RAW,
    namespace IN VARCHAR2,
    attribute IN VARCHAR2,
    old_value IN VARCHAR2,
    new_value IN VARCHAR2,
    event_code IN PLS_INTEGER)

RETURNS PLS_INTEGER;
```

Attribute List

The attribute list includes the attributes defined for the namespace. These attributes are created in the session when the namespace is created.

You can specify the following optional data for attributes:

The default value

The attribute is initialized with the default value when the namespace is created in the application session. The default value and the event types <code>FIRSTREAD\_EVENT</code> and <code>FIRSTREAD\_PLUS\_UPDATE\_EVENT</code> cannot exist at the same time.

Event types

You can specify the following event types for an attribute:

\* FIRSTREAD EVENT

Specify this event type to call the namespace handler when an attribute whose value has not been set is read for the first time. You can specify this event type only if a default value has not been set for the attribute.

\* UPDATE EVENT

Specify this event type to call the namespace handler when the attribute value is updated.

\* FIRSTREAD PLUS UPDATE EVENT

Specify this event type to call the namespace handler when an attribute whose value has not been set is read for the first time, or when its value is updated. You can specify this event type only if a default value has not been set for the attribute.



### Namespace ACL

The privilege model for namespace operations. Namespace operations are protected by the ACL set on the template. By default, <code>NS\_UNRESTRICTED\_ACL</code> is set on a template, which allows unrestricted operation on namespaces created from the templates.

## 3.3.1.2 About Namespace Views

You can find information about namespace templates, namespace template attributes, and namespace attributes in current and all application sessions by querying these data dictionary views:

- "DBA\_XS\_NS\_TEMPLATES"
- "DBA\_XS\_NS\_TEMPLATE\_ATTRIBUTES"
- "DBA XS SESSION NS ATTRIBUTES"
- "V\$XS SESSION NS ATTRIBUTES"

## 3.3.1.3 Creating a Namespace Template for an Application Session

You can create a namespace template using the XS\_NAMESPACE.CREATE\_TEMPLATE procedure in PL/SQL or the createNamespace method of the Session interface in Java.

Example 3-12 shows how to create the namespace template ns1 for an application session. It defines the attributes for this namespace using the list of attributes attrs. Because this namespace template has NS\_UNRESTRICTED\_ACL set on the template, this allows unrestricted operation on namespaces created from the template.

The calling user must have the <code>ADMIN\_ANY\_SEC\_POLICY</code> application privilege, which allows it to administer namespace templates and attributes.

#### **Example 3-12 Creating a Namespace Template**

```
attrs XS$NS ATTRIBUTE LIST;
BEGIN
  attrs := XS$NS ATTRIBUTE LIST();
  attrs.extend(3);
  attrs(1) := XS$NS ATTRIBUTE('attr1','value1',
   XS NAMESPACE. UPDATE EVENT);
  attrs(2) := XS$NS ATTRIBUTE('attr2', null,
   XS NAMESPACE.FIRSTREAD PLUS UPDATE EVENT);
  attrs(3) := XS$NS ATTRIBUTE('attr3','value3');
  SYS.XS_NAMESPACE.CREATE_TEMPLATE(name=>'ns1',
                               description=>'namespace template 1',
                               attr list=>attrs,
                               schema=>'SCOTT',
                               package=>'PKG1',
                               function=>'FN1',
                               acl=>'SYS.NS UNRESTRICTED ACL');
END;
```



- CREATE\_TEMPLATE Procedure for information about the syntax of this PL/SQL procedure
- Oracle Database Real Application Security Java API Reference for information about the syntax of the Java createNamespace method (in Javadoc format)
- Example 6-10 for information about a Java example of this task

## 3.3.2 Initializing a Namespace in an Application Session

A namespace can be initialized, using a namespace template, during any of the following events, as described in this section:

#### This section contains:

- Initializing a Namespace When the Session Is Created
- Initializing a Namespace When the Session Is Attached
- Initializing a Namespace When a Named Application User Is Assigned to an Anonymous Application Session
- Initializing a Namespace When the Application User Is Switched in an Application Session
- Initializing a Namespace Explicitly

## 3.3.2.1 Initializing a Namespace When the Session Is Created

When you create an application session using the <code>DBMS\_XS\_SESSIONS.CREATE\_SESSION</code> procedure in PL/SQL or the <code>createSession</code> method of the <code>XSSessionManager</code> class in Java, you can specify a list of namespaces to initialize.

Example 3-13 shows how to initialize two namespaces, ns1 and ns2, while creating an application session.

If you specify namespaces during creation of the session, the caller is required to be granted application privileges <code>MODIFY\_NAMESPACE</code> or <code>MODIFY\_ATTRIBUTE</code> on the namespaces, or be granted the <code>ADMIN\_NAMESPACE</code> system privilege.



The namespaces used in Example 3-13 must already have corresponding namespace templates defined.

#### Example 3-13 Initializing Namespaces When Creating an Application Session

DECLARE
 nsList DBMS\_XS\_NSATTRLIST;
 sessionid RAW(16);



```
BEGIN
    nsList := DBMS_XS_NSATTRLIST(DBMS_XS_NSATTR('ns1'),DBMS_XS_NSATTR('ns2'));
    SYS.DBMS_XS_SESSIONS.CREATE_SESSION('lwuser1', sessionid, FALSE, FALSE, nsList);
END;
//
```

- CREATE\_SESSION Procedure for information about the syntax of this PL/SQL procedure
- Oracle Database Real Application Security Java API Reference for information about the syntax of the Java createSession method (in Javadoc format)
- Example 6-2 for information about a Java example of this task

## 3.3.2.2 Initializing a Namespace When the Session Is Attached

When you attach the session using the <code>DBMS\_XS\_SESSIONS.ATTACH\_SESSION</code> procedure in PL/SQL or using the <code>attachSession</code> method of the <code>XSSessionManager</code> class in Java, you can specify a list of namespaces to initialize.

Example 3-14 shows how to initialize two namespaces, ns1 and ns2, while attaching an application session.

If you specify namespaces, then the user is required to be granted application privileges MODIFY\_NAMESPACE or MODIFY\_ATTRIBUTE on the namespaces, or the ADMIN\_NAMESPACE system privilege.

## Note:

The namespaces used in Example 3-14 must already have corresponding namespace templates defined.

#### Example 3-14 Initializing Namespaces When Attaching an Application Session

```
DECLARE
  nsList DBMS_XS_NSATTRLIST;
  sessionid RAW(16);
BEGIN
   nsList := DBMS_XS_NSATTRLIST(DBMS_XS_NSATTR('ns1'), DBMS_XS_NSATTR('ns2'));
   SYS.DBMS_XS_SESSIONS.CREATE_SESSION('lwuser1', sessionid);
   SYS.DBMS_XS_SESSIONS.ATTACH_SESSION(sessionid, NULL, NULL, NULL, nsList);
END;
//
```



- ATTACH\_SESSION Procedure for information about the syntax of this PL/SQL procedure
- Oracle Database Real Application Security Java API Reference for information about the syntax of the Java attachSession method (in Javadoc format)
- Example 6-3 for information about a Java example of this task

# 3.3.2.3 Initializing a Namespace When a Named Application User Is Assigned to an Anonymous Application Session

When you assign an application user to an application session using the <code>DBMS\_XS\_SESSIONS.ASSIGN\_USER</code> procedure in PL/SQL or the <code>assignUser</code> method of the <code>XSSessionManager</code> class in Java, you can specify a list of namespaces to initialize.

If you specify namespaces, then the user is required to be granted application privileges <code>MODIFY\_NAMESPACE</code> or <code>MODIFY\_ATTRIBUTE</code> on the namespaces, or <code>ADMIN NAMESPACE</code> system privilege.

Example 3-15 shows how to initialize two namespaces, ns1 and ns2, while assigning an application user to an application session.



The namespaces used in Example 3-15 must already have corresponding namespace templates defined.

# **Example 3-15** Initializing Namespaces When Assigning an Application User to an Application Session



- ASSIGN\_USER Procedure for information about the syntax of this PL/SQL procedure
- Oracle Database Real Application Security Java API Reference for information about the syntax of the Java assignUser method (in Javadoc format)
- Example 6-5 for information about a Java example of this task

# 3.3.2.4 Initializing a Namespace When the Application User Is Switched in an Application Session

When you switch an application user in an application session using the <code>DBMS\_XS\_SESSIONS.SWITCH\_USER</code> procedure in PL/SQL or using the <code>switchUser</code> method of the <code>Session</code> interface in Java, you can specify a list of namespaces to initialize.

If you specify namespaces, then the user is required to be granted application privileges MODIFY\_NAMESPACE or MODIFY\_ATTRIBUTE on the namespaces, or the ADMIN\_NAMESPACE system privilege.



To enable the switch from lwuser1 to lwuser2 after attaching the session, you must first define lwuser2 as the target user for lwuser1, as follows:

```
exec XS_PRINCIPAL.ADD_PROXY_USER('lwuser2', 'lwuser1');
```

Example 3-16 shows how to initialize two namespaces, ns1 and ns2, while switching an application user in an application session.



The namespaces used in Example 3-16 must already have corresponding namespace templates defined.

# Example 3-16 Initializing Namespaces When Switching an Application User in an Application Session



END;

## See Also:

- SWITCH\_USER Procedure for information about the syntax of this PL/SQL procedure
- Oracle Database Real Application Security Java API Reference for information about the syntax of the Java switchUser method (in Javadoc format)
- Example 6-6 for information about a Java example of this task

## 3.3.2.5 Initializing a Namespace Explicitly

You can explicitly initialize a namespace in an application session using the <code>DBMS\_XS\_SESSIONS.CREATE\_NAMESPACE</code> procedure in PL/SQL or the <code>createNamespace</code> method of the <code>Session</code> interface in Java.

To execute the <code>DBMS\_XS\_SESSIONS.CREATE\_NAMESPACE</code> procedure, the calling user must have the <code>MODIFY\_NAMESPACE</code> application privilege on the namespace or the <code>ADMIN\_NAMESPACE</code> system privilege.

Example 3-17 shows how to explicitly initialize a namespace, ns1, in an application session.



The namespace used in Example 3-17 must already have a corresponding namespace template defined.

### **Example 3-17** Initializing a Namespace Explicitly in an Application Session

```
DECLARE
  sessionid RAW(30);
BEGIN
  SYS.DBMS_XS_SESSIONS.CREATE_SESSION('lwuser1', sessionid);
  SYS.DBMS_XS_SESSIONS.ATTACH_SESSION(sessionid);
  SYS.DBMS_XS_SESSIONS.CREATE_NAMESPACE('ns1');
END;
/
```



- CREATE\_NAMESPACE Procedure for information about the syntax of this PL/SQL procedure
- Oracle Database Real Application Security Java API Reference for information about the syntax of the Java createNamespace method (in Javadoc format)
- Example 6-10 for information about a Java example of this task

## 3.3.3 Setting Session Attributes in an Application Session

You can set the value of a specific session attribute using the DBMS\_XS\_SESSIONS.SET\_ATTRIBUTE procedure in PL/SQL or the setAttribute method of the SessionNamespace interface method in Java.

The calling user is required to be granted the <code>MODIFY\_ATTRIBUTE</code> application privilege on the namespace or the <code>ADMIN NAMESPACE</code> system privilege.



An attribute can store a string value up to 4000 characters long.

Example 3-18 shows how to set a value, val1, for an attribute, attr1, of the application session.

#### **Example 3-18** Setting a Namespace Attribute for an Application Session

```
DECLARE
sessionid RAW(16);
BEGIN
SYS.DBMS_XS_SESSIONS.CREATE_SESSION('lwuserl', sessionid);
SYS.DBMS_XS_SESSIONS.ATTACH_SESSION(sessionid);
SYS.DBS_XS_SESSIONS.CREATE_NAMESPACE('ns1');
SYS.DBMS_XS_SESSIONS.SET_ATTRIBUTE('ns1', 'attr1', 'val1');
SYS.DBMS_XS_SESSIONS.DETACH_SESSION;
SYS.DBMS_XS_SESSIONS.DESTROY_SESSION(sessionid);
END;
//
```

## See Also:

- SET\_ATTRIBUTE Procedure for more information about the syntax of this PL/SQL procedure
- Oracle Database Real Application Security Java API Reference for information about the syntax of the Java setAttribute method (in Javadoc format)
- About Setting a Session Namespace Attribute for information about this task in Java



## 3.3.4 Getting Session Attributes in an Application Session

You can retrieve the value of a specific session attribute using the DBMS\_XS\_SESSIONS.GET\_ATTRIBUTE procedure in PL/SQL or using the getAttribute method of the SessionNamespace interface method in Java.

The calling user is not required to be granted any privileges to get attributes using the DBMS XS SESSIONS.GET ATTRIBUTE procedure.



If an attribute value has not been set, and the FIRSTREAD\_EVENT has been specified for the attribute, then an attempt to read the the attribute value triggers a call to the namespace event handler. The namespace event handler procedure typically sets a value for the attribute, and performs other application-specific processing tasks.

Example 3-19 shows how to retrieve an attribute, attr1, of the application session.

#### **Example 3-19 Getting a Namespace Attribute for an Application Session**

```
DECLARE

sessionid RAW(16);
attrib_out_val VARCHAR2(4000);

BEGIN

SYS.DBMS_XS_SESSIONS.CREATE_SESSION('lwuser1', sessionid);
SYS.DBMS_XS_SESSIONS.ATTACH_SESSION(sessionid);
SYS.DBMS_XS_SESSIONS.CREATE_NAMESPACE('ns1');
SYS.DBMS_XS_SESSIONS.SET_ATTRIBUTE('ns1', 'attr1', 'val1');
SYS.DBMS_XS_SESSIONS.GET_ATTRIBUTE('ns1', 'attr1', attrib_out_val);
SYS.DBMS_XS_SESSIONS.DETACH_SESSION;
SYS.DBMS_XS_SESSIONS.DESTROY_SESSION(sessionid);
END;
```

## See Also:

- GET\_ATTRIBUTE Procedure for information about the syntax of this PL/SQL procedure
- Oracle Database Real Application Security Java API Reference for information about the syntax of the Java getAttribute method (in Javadoc format)
- Getting a Session Namespace Attribute for information about this task in Java



## 3.3.5 Creating Custom Attributes in an Application Session

You can create custom attributes in a namespace using the DBMS\_XS\_SESSIONS.CREATE\_ATTRIBUTE procedure in PL/SQL or the createAttribute method of the SessionNamespace interface method in Java.

Custom attributes differ from template attributes. Template attributes are part of the namespace template, and are automatically created in the session when the namespace is created. Custom attributes are programmatically created in a namespace, using the CREATE ATTRIBUTE procedure.

The calling application is required to be granted the MODIFY\_ATTRIBUTE application privilege on the namespace or the ADMIN NAMESPACE system privilege.

Example 3-20 shows how to create a custom attribute, customattr, in a namespace of the application session.

#### Example 3-20 Creating a Custom Namespace Attribute for an Application Session

```
DECLARE

sessionid RAW(16);
attrib_out_val VARCHAR2(4000);

BEGIN

SYS.DBMS_XS_SESSIONS.CREATE_SESSION('lwuser1', sessionid);
SYS.DBMS_XS_SESSIONS.ATTACH_SESSION(sessionid);
SYS.DBMS_XS_SESSIONS.CREATE_NAMESPACE('ns1');

SYS.DBMS_XS_SESSIONS.CREATE_ATTRIBUTE('ns1','customattr','default_value_custom',NULL);

SYS.DBMS_XS_SESSIONS.SET_ATTRIBUTE('ns1','customattr','newvalue');
SYS.DBMS_XS_SESSIONS.GET_ATTRIBUTE('ns1', 'customattr', attrib_out_val);
SYS.DBMS_XS_SESSIONS.DETACH_SESSION;
SYS.DBMS_XS_SESSIONS.DETACH_SESSION(sessionid);
END;

END;
```

### See Also:

- CREATE\_ATTRIBUTE Procedure for information about the syntax of this PL/SQL procedure
- Oracle Database Real Application Security Java API Reference for information about the syntax of the Java createAttribute method (in Javadoc format)
- Example 6-13 for information about a Java example of this task

## 3.3.6 Deleting a Namespace in an Application Session

You can delete a namespace and all attributes identified by it from an application session using the DBMS\_XS\_SESSIONS.DELETE\_NAMESPACE procedure in PL/SQL or the deleteAttribute method of the SessionNamespace interface method in Java.

The calling user must have the MODIFY\_NAMESPACE application privilege on the namespace or the ADMIN NAMESPACE system privilege.

Example 3-21 shows how to delete a namespace ns1 from an application session.

#### Example 3-21 Deleting a Namespace in an Application Session

```
DECLARE

sessionid RAW(16);
out_value VARCHAR2(4000);

BEGIN

SYS.DBMS_XS_SESSIONS.CREATE_SESSION('lwuser1', sessionid);
SYS.DBS_XS_SESSIONS.ATTACH_SESSION(sessionid);
SYS.DBMS_XS_SESSIONS.CREATE_NAMESPACE('ns1');
SYS.DBMS_XS_SESSIONS.SET_ATTRIBUTE('ns1', 'attr1', 'val1');
SYS.DBMS_XS_SESSIONS.GET_ATTRIBUTE('ns1', 'attr1', out_value);
SYS.DBMS_XS_SESSIONS.DELETE_NAMESPACE('ns1');
SYS.DBMS_XS_SESSIONS.DETACH_SESSION;
SYS.DBMS_XS_SESSIONS.DETACH_SESSION;
SYS.DBMS_XS_SESSIONS.DESTROY_SESSION(sessionid);
END;
```

## See Also:

- DELETE\_NAMESPACE Procedure for information about the syntax of this PL/SQL procedure
- Oracle Database Real Application Security Java API Reference for information about the syntax of the Java deleteNamespace method (in Javadoc format)
- Example 6-11 for information about a Java example of this task

## 3.3.7 Enabling Application Roles for a Session

You can enable only directly granted regular application roles of an application session user using the <code>DBMS\_XS\_SESSIONS.ENABLE\_ROLE</code> procedure in PL/SQL or the <code>enableRole</code> method of the <code>Session</code> interface in Java.

The DBA\_XS\_SESSION\_ROLES dynamic data dictionary view lists application roles enabled in all application sessions. The V\$XS\_SESSION\_ROLES dynamic data dictionary view lists application roles enabled in the currently attached application session.

Example 3-22 shows how to enable a role in an application session.

#### Example 3-22 Enabling a Role in an Application Session

```
DECLARE
   sessionid RAW(16);
BEGIN
   SYS.DBMS_XS_SESSIONS.CREATE_SESSION('lwuserl', sessionid);
   SYS.DBMS_XS_SESSIONS.ATTACH_SESSION(sessionid);
   SYS.DBMS_XS_SESSIONS.ENABLE_ROLE('authl_role');
   SYS.DBMS_XS_SESSIONS.DETACH_SESSION;
   SYS.DBMS_XS_SESSIONS.DETACH_SESSION;
   SYS.DBMS_XS_SESSIONS.DESTROY_SESSION(sessionid);
END;
//
```



- ENABLE\_ROLE Procedure for information about the syntax of this PL/SQL procedure
- Oracle Database Real Application Security Java API Reference for information about the syntax of the Java enableRole method (in Javadoc format)
- Example 6-7 for information about a Java example of this task

## 3.3.8 Disabling Application Roles for a Session

You can disable application roles for a specific session using the DBMS\_XS\_SESSIONS.DISABLE\_ROLE procedure in PL/SQL or the disableRole method of the Session interface in Java.

Example 3-23 shows how to disable a role in an application session.

#### Example 3-23 Disabling a Role in an Application Session

```
DECLARE
  sessionid RAW(16);
BEGIN
  SYS.DBMS_XS_SESSIONS.CREATE_SESSION('lwuserl', sessionid);
  SYS.DBMS_XS_SESSIONS.ATTACH_SESSION(sessionid);
  SYS.DBMS_XS_SESSIONS.ENABLE_ROLE('authl_role');
  SYS.DBMS_XS_SESSIONS.DISABLE_ROLE('authl_role');
  SYS.DBMS_XS_SESSIONS.DETACH_SESSION;
  SYS.DBMS_XS_SESSIONS.DESTROY_SESSION(sessionid);
  END;
//
```

### See Also:

- DISABLE\_ROLE Procedure for information about the syntax of this PL/SQL procedure
- Oracle Database Real Application Security Java API Reference for information about the syntax of the Java disableRole method (in Javadoc format)
- Example 6-8 for information about a Java example of this task

## 3.4 About Administrative APIs for External Users and Roles

This section describes the following administrative APIs that are required for external users and roles:

- CREATE\_SESSION Procedure
- ATTACH SESSION Procedure
- ASSIGN\_USER Procedure
- SAVE SESSION Procedure



# 3.5 About Real Application Security Session Privilege Scoping Through ACL

Describes session privilege scoping through an ACL allowing per principal session privilege grants through an ACL set on the principal, where the principal can be either an application user or a dynamic role.

In Oracle Database 12c Release 1 (12.1), Real Application Security session privileges are granted through <code>GRANT\_SYSTEM\_PRIVILEGE</code> procedure or revoked through <code>REVOKE\_SYSTEM\_PRIVILEGE</code> procedure in the <code>XS\_ADMIN\_UTIL</code> package. These grants are applicable system wide and allow the grantee to exercise the grants for session operations on any Real Application Security principal. This is implemented using a seeded system ACL — <code>SESSIONACL</code>. All session privilege checks are done in this ACL. For a database running multiple applications with multiple user communities this approach becomes cumbersome to administer Real Application Security session administration appropriately.

Session privilege scoping addresses two issues. The first issue is who can create, attach, detach, or destroy a user's session. The second issue is who can enable a dynamic role for a user session. Because a dynamic role can never be granted and can only be dynamically enabled, session privilege scoping through an ACL is needed. Because regular roles are granted to application users, an ACL is not needed on them.

Beginning with Oracle Database 12c Release 2 (12.2), Real Application Security supports session privilege scoping through an ACL. This feature allows per principal session privilege grants through an ACL set on the principal. The ACL containing the session privilege grant can be set on a regular Real Application Security application user or a dynamic role, but it cannot be set on a regular Real Application Security role or external principal. Because Real Application Security session scoping can be enforced per the ACL set on the application user or dynamic role involved in a session operation, you can more finely restrict session operations to specific user communities. For example, you can set up a session operation for a separate user community that can be managed by separate session managers (dispatchers), so users belonging to the same user community will have the same ACL set on them. In addition, enabling and disabling of a dynamic role can be restricted to appropriate dispatchers with the addition of a new privilege ENABLE DYNAMIC ROLE to restrict enablement and disablement of dynamic roles. This privilege is enforced even for enabling existing session scope dynamic roles from previous attach. With these features, the Real Application Security session administrator can provide a scoping for session privileges that allows a user to create or attach or modify sessions for one Real Application Security user community or a group of Real Application Security user communities for which it has been granted privilege.

The ACL can be set on the principal at creation time using the Real Application Security least system privilege PROVISION, which can create, modify, or drop application users and application roles. In addition, you can also use the CREATE\_USER procedure and CREATE\_DYNAMIC\_ROLE procedure and this requires the caller to have ALTER USER or ALTER ROLE privilege depending on the principal being created; or the ACL can be created after principal creation using the SET\_ACL procedure, which requires the caller to have the same privilege as previously mentioned depending on whether the principal is an application user or dynamic role. Both the application user or dynamic role and ACL must already exist before setting the ACL using the SET\_ACL procedure and the ACL must have been created in the SYS schema.



Session operations require either specific session privileges to be granted to the session manager depending on the session operation by either a system-wide ACL or an ACL attached to the affected user or role. The ADMINISTER\_SESSION privilege aggregates all specific session privileges. Like all session privileges, this privilege can be granted to the session manager by either a system-wide ACL or principal specific ACL. The use of new privileges is audited like other system privileges. An ACL set on a user or dynamic role overrides the system-wide ACL.

The DBA\_XS\_USERS and DBA\_XS\_DYNAMIC\_ROLES views are enhanced to show an additional column for the ACL that is set on the Real Application Security application user or dynamic role.

System level session privilege grants can coexist with the principal specific ACL, but the principal specific ACL grants have precedence. This precedence is important as Real Application Security principal specific ACLs can have negative grants. Note that the negative grant can only appear in principal specific ACLs not for the system ACL.

The following table describes the behavior of a session privilege check comparing the principal specific ACL (column 1) with the System ACL (column 2) and showing the result of the session privilege check (column 3) as being either True or False. For example, when the checked privilege is neither granted or denied in a principal specific ACL, then the System ACL is checked for the privilege.

Table 3-2 Session Privilege Checking

Principal Specific ACL	System ACL	Session Privilege Check Result
Grant	Deny, grant, or not specified	True
Deny	Deny, grant, or not specified	False
Not specified or ACL does not exist	Grant	True
Not specified or ACL does not exist	Not specified.	False

## **Enforcing Session Privilege According to the ACL Set on the Principal**

Session privilege is enforced according to the ACL set on the Real Application Security application user in the session operation. A privilege to enable a dynamic role is enforced according to the ACL set on the dynamic role. For example, a create session operation requires the caller to have the CREATE\_SESSION privilege in the ACL set on the Real Application Security application user. Similarly, the attach operation with dynamic role requires the ENABLE\_DYNAMIC\_ROLES privilege in the ACLs set on the dynamic roles. Any existing system level session privilege grants as mentioned previously can still coexist, but the principal specific ACL grants gets precedence.

Privilege check is first done in the ACL associated with the principal (if at all there are such settings). If the ACL check succeeds the operation will go through. If the check finds deny, the operation fails with insufficient privilege error. If neither grant nor deny is found, the check is done in system ACL associated with SESSION\_SC security class and operation fails or succeeds based on this privilege check result.

The following table lists Real Application Security session operations and the required session privileges to perform that operation. This information is useful for creating ACLs on principals for specific session operations. Note that the ADMINISTER\_SESSION privilege aggregates all session privileges listed in this table.



Table 3-3 Session Privilege Operations and the Required Privileges to Perform Them

Session Operations	Required Session Privilege
Create Session	CREATE_SESSION privilege in the ACL set on the Real Application Security application user (that is used for creating the session) or in the System ACL. If there is a negative grant in the principal specific ACL, the operation fails.
Attach Session	ATTACH_SESSION privilege in the ACL set on the Real Application Security application user (that is used for creating the session) or in the System ACL. ENABLE_DYNAMIC_ROLE privilege in the ACL set on the dynamic roles (that is used for creating the session) or in the
	System ACL. If there is a negative grant in the principal specific ACL, the operation fails.
Assign User	ASSIGN_SESSION privilege in the ACL set on the named Real Application Security application user to be assigned or in the System ACL. ENABLE_DYNAMIC_ROLE privilege in the ACL set on the dynamic roles (that is used for creating the session) or in the System ACL. So if there are dynamic roles enabled, these privileges are checked. If there is a negative grant in the principal specific ACL, the operation fails.
Switch User	No session privilege check, as per proxy configuration.
Enable Role	No session privilege check.
Disable Role	No session privilege check.
Namespace Operation (Create Namespace, delete namespace, create attribute, set attribute, reset attribute, delete attribute)	No session privilege check, only namespace privilege checks.
Save Session, Detach Session	No session privilege check.
Destroy Session	TERMINATE_SESSION privilege in the ACL set on the Real Application Security application user (that is used for creating the session) or in the System ACL.
Set Session Cookie	MODIFY_SESSION privilege in the ACL set on the Real Application Security application user (that is used for creating the session) or in the System ACL.
Set Inactivity Timeout	MODIFY_SESSION privilege in the ACL set on the Real Application Security application user (that is used for creating the session) or in the System ACL.
Reauthorize Session	MODIFY_SESSION privilege in the ACL set on the Real Application Security application user (that is used for creating the session) or in the System ACL.
Get SID from Cookie	No session privilege check.



Table 3-3 (Cont.) Session Privilege Operations and the Required Privileges to Perform Them

Session Operations	Required Session Privilege
Global Callback Configuration (Add global callback, delete global callback, enable global callback)	No session privilege check, only callback privilege checks.

- GRANT\_SYSTEM\_PRIVILEGE Procedure and REVOKE\_SYSTEM\_PRIVILEGE Procedure
- Security Classes
- CREATE\_USER Procedure, CREATE\_DYNAMIC\_ROLE Procedure, and SET ACL Procedure
- DBA\_XS\_USERS and DBA\_XS\_DYNAMIC\_ROLES

## 3.5.1 Granting Session Privileges on a Principal Using an ACL

Describes how to grant session privileges on a principal using an ACL while creating the user and after the user is already created.

The following examples show how to grant session privileges to principals through an ACL,  $user_{ACL}$  while creating the user and after the user is already created.

First, create the ACL USER\_ACL and grant the privilege ADMINISTER\_SESSION to user lwuser3 and grant the privileges CREATE\_SESSION, MODIFY\_SESSION, and ATTACH\_SESSION to user lwuser4 and grant the privileges CREATE SESSION and MODIFY SESSION to user lwuser5.

```
sqlplus /nolog
SQL> CONNECT SYS/password as SYSDBA
SQL> GRANT CREATE SESSION, XS_SESSION_ADMIN TO SEC_MGR IDENTIFIED BY
password;
SQL> EXEC SYS.XS_ADMIN_UTIL.GRANT_SYSTEM_PRIVILEGE('PROVISION', 'sec_mgr',
SYS.XS_ADMIN_UTIL.PTYPE_DB);

CONNECT SEC_MGR
Enter password: password
Connected.

DECLARE
ace_list XS$ACE_LIST;
BEGIN
ace_list := XS$ACE_LIST(
    XS$ACE TYPE(privilege list=>XS$NAME_LIST('"ADMINISTER_SESSION"'),
```



```
granted=>true,
                principal name=>'lwuser3'),
XS$ACE TYPE(privilege list=>XS$NAME LIST('"CREATE SESSION"','"MODIFY SE
SSION"', '"ATTACH SESSION"'),
                granted=>true,
                principal name=>'lwuser4'),
XS$ACE TYPE(privilege list=>XS$NAME LIST('"CREATE SESSION"','"MODIFY SE
SSION"'),
                granted=>true,
                principal name=>'lwuser5'));
sys.xs_acl.create_acl(name=>'USER ACL',
                     ace list=>ace list,
                     sec class=>'SESSIONPRIVS',
                     description=>'Session management');
END;
Next, create users lwuser3 and lwuser4 and grant these users the ACL, USER ACL.
BEGIN
 SYS.XS PRINCIPAL.CREATE USER(name=>'lwuser3',
                               schema=>'HR',
                               acl=>'USER ACL';
END;
     SYS.XS PRINCIPAL.SET PASSWORD('lwuser3', 'password');
EXEC
BEGIN
 SYS.XS PRINCIPAL.CREATE USER(name=>'lwuser4',
                               schema=>'HR',
                               acl=>'USER ACL';
END;
EXEC SYS.XS PRINCIPAL.SET PASSWORD('lwuser4', 'password');
Next, create user lwuser5 and set the ACL, USER ACL, for this user using the SET ACL
procedure.
sqlplus SEC MGR
Enter password: password
Connected.
 SYS.XS PRINCIPAL.CREATE USER(name=>'lwuser5',
                               schema=>'HR');
END;
EXEC SYS.XS PRINCIPAL.SET ACL('lwuser5', 'USER ACL');
```



- GRANT\_SYSTEM\_PRIVILEGE Procedure and REVOKE\_SYSTEM\_PRIVILEGE Procedure
- Security Classes
- CREATE\_USER Procedure, CREATE\_DYNAMIC\_ROLE Procedure, and SET\_ACL Procedure
- DBA\_XS\_USERS and DBA\_XS\_DYNAMIC\_ROLES



4

# Configuring Application Privileges and Access Control Lists

This chapter describes how to configure application privileges and access control lists (ACLs) in Oracle Database Real Application Security. It includes information on how to create, set, and modify ACLs, and describes how ACL security interacts with other Oracle Database security mechanisms.

This chapter contains the following sections:

- About Application Privileges
- About Configuring Security Classes
- About Configuring Access Control Lists
- Data Security
- ACL Binding

# 4.1 About Application Privileges

The database has predefined system privileges, such as CREATE TABLE, and object privileges, such as UPDATE. A large number of custom privileges that must be defined for enterprise applications are often called application-defined privileges. Real Application Security introduces the definition of these privileges, termed application privileges, in the database. For application developers, these custom application privileges are used for access control on application-level operations. These application-level operations allow fine-grained access on data at a granular level of columns, rows, or cells.

When an application privilege is explicitly bound to a resource, for example, rows and columns of a table, an application privilege can be used to protect an application-level operation on a database object. Alternatively, it may be used in the same manner as a system privilege when binding to a resource is not required.



About Checking ACLs for a Privilege

This section contains the following topic: Aggregate Privilege.

## 4.1.1 Aggregate Privilege

A Real Application Security **aggregate privilege** implies a set of other application privileges. The implied application privileges of an aggregate privilege can be any application privilege defined by the current security class or an inherited application privilege (see "About Configuring Security Classes" for more information). When an aggregate privilege is granted or denied, its implied application privileges are implicitly granted or denied.

When an aggregate privilege AG implies the application privileges p1 and p2, granting the application privilege, AG, implies that both p1 and p2 are granted. However, granting both the p1 and p2 does not imply that AG is granted.

Aggregate privileges are useful for the following purposes:

- Enabling grouping and granting a set of application privileges as a single grant, simplifying application privilege administration. A group name or an alias for a set of application privileges, where the group name itself is not an application privilege, makes checking for the set simpler as it checks for each application privilege in the group.
- Providing an efficient way to check a set of application privileges based on a single application privilege check.

Example 4-1 adds an aggregate privilege called <code>UPDATE\_INFO</code> to the <code>HRPRIVS</code> security class. The aggregate privilege contains the implied privileges <code>UPDATE</code>, <code>DELETE</code>, and <code>INSERT</code>.

When the group name itself is a first class privilege, there may be several possible semantics for the aggregate privilege based on its relationship to its members. When defining a semantic to represent an aggregate privilege, you must consider various relations between the aggregate privilege and its members, such as *imply* and *include*. For example, consider the *imply* relation in Java Security; selecting this semantic when granting an aggregate privilege implies granting all its member application privileges individually, but not the aggregate privilege. Therefore, granting all the member application privileges of an aggregate does not imply granting the aggregate privilege.

Example 4-2 adds a list of implied application privileges for the aggregate privilege UPDATE INFO.

An aggregate privilege is not an application role. An application role itself is not an application privilege that protects a resource. Application roles are used to activate and deactivate application privileges available to an application user to enforce role-based access control constraints.

Also, an aggregate privilege is not a security class. A security class is not an application privilege that can be granted to a user. A security class lists a set of application privileges including aggregate privileges that may be either granted or denied in an ACL. Within a security class, many aggregate privileges may be defined based on the application privileges available in the security class.

An aggregate privilege can have other aggregate privileges as its members. Note that the member privileges of an aggregate privilege must be defined in the same security class (or in an ancestor security class) as the aggregate privilege. An aggregate privilege definition cannot create a cycle.

#### Example 4-1 Adding an Aggregate Privilege to a Security Class

### Example 4-2 Adding Implied Privileges to an Aggregate Privilege

```
BEGIN
SYS.XS_SECURITY_CLASS.ADD_IMPLIED_PRIVILEGES(sec_class =>'HRPRIVS',
```



```
(priv=>'UPDATE_INFO',
implied_priv_list=>XS$NAME_LIST('"UPDATE"', '"DELETE"', '"INSERT"'));
END;
```

This section contains: ALL Privilege.

## 4.1.1.1 ALL Privilege

The ALL privilege is a predefined aggregate privilege. Every security class has the ALL privilege, and it contains all the application privileges of that security class. ALL is not explicitly defined in every security class, but it is internally understood by the system based on the security class associated with the ACL. The cardinality of an ALL for a security class changes whenever an application privilege is added or removed from the security class.

Use of the ALL construct enables Real Application Security to express access control policy such as "grant all the application privileges to the application user u1 defined for an application except the specific privilege p1". Example 4-3 shows an ACL in the security class, AppSecurityClass, which has all the application privileges for the application. The ordered evaluation of ACEs ensures that the ALL except p1 is granted to the application user u1.

### **Example 4-3 Using ALL Grant**

# 4.2 About Configuring Security Classes

This section contains the following topics:

- About Security Classes
- Security Class Inheritance
- Security Class as Privilege Scope
- DML Security Class
- About Validating Security Classes
- Manipulating Security Classes

## 4.2.1 About Security Classes

A **security class** is a scope for a set of application privileges. The same application privilege can be defined in multiple security classes. A security class restricts the set of application privileges that may be granted or denied within an ACL. A security class is both a place to define a collection of relevant application privileges and a way to associate an ACL with one security class.

Real Application Security supports a set of predefined application privileges and security classes and also allows applications to define their own custom application privileges using security classes. Each class of object being protected is associated with a security class that indicates the set of operations that may be performed on its objects. There are predefined security classes that define built-in application privileges.

Security classes simplify the task of managing a large number of application privileges. Each ACL is associated with one security class. This security class defines the scope of application privileges that may be granted within the ACL.

Each object type can support a large number of application privileges, and many different object types may share a common set of operations. To simplify these types of specifications, security classes support inheritance.

## 4.2.2 Security Class Inheritance

A security class can inherit application privileges from parent security classes. A child security class implicitly contains all the application privileges defined in the parent security classes. The application privileges available in a security class are the combination of the application privileges defined in the security class and the application privileges inherited from parent security classes.

A security class can specify a list of parent security classes. The application privileges available in these parent classes become available in the child class. When the same application privilege name is defined in a child and its parent security class, the application privilege in the child replaces or overrides the application privilege in the parent.

Example 4-4 shows security class inheritance by creating a security class called HRPRIVS. The HRPRIVS security class defines two application privileges, VIEW\_SENSITIVE\_INFO and UPDATE\_INFO. UPDATE\_INFO, which is an aggregate privilege that implies three other privileges: UPDATE, DELETE, and INSERT. The security class HRPRIVS inherits application privileges from DML security class as specified by the parent list parameter.

#### **Example 4-4** Showing Security Class Inheritance



## 4.2.3 Security Class as Privilege Scope

An ACL has a single security class as its **scope**. An ACL grants application privileges to principals to control access to protected data or functionality; it can grant only the application privileges that are defined in its security class. The <code>security\_class</code> parameter is used to specify the security class in an ACL. When checking an application privilege against an ACL, the security class of the application privilege is resolved based on the security class of the ACL, as the ACL always has an associated security class. If no security class is specified, then the DML Security Class is used as the default security class. Different ACLs can have as their scope the same security class.

## 4.2.4 DML Security Class

The DML security class is predefined or created during installation. The DML security class contains common application privileges for object manipulation: SELECT, INSERT, UPDATE, and DELETE. If an ACL does not specify its security class, DML is the default security class for the ACL.

Real Application Security DML application privileges are the same as database object privileges and inherently enforced by database object-level operations. However, Real Application Security DML application privileges are effective only when Real Application Security Data Security is enabled for database tables.

## 4.2.5 About Validating Security Classes

Oracle recommends that you always validate the Real Application Security objects after administrative configuration changes. The XS\_DIAG package provides a set of validation APIs to help ensure that these changes do not damage the complicated relationships among your Real Application Security objects.

See "VALIDATE\_SECURITY\_CLASS Function" for more information about validating a security class.

## 4.2.6 Manipulating Security Classes

To manipulate security classes, use the procedures in PL/SQL package XS\_SECURITY\_CLASS; it includes procedures to create, manage, and delete security classes and their application privileges. This package also includes procedures for managing security class inheritance; see "XS\_SECURITY\_CLASS\_Package".

Example 4-5 invokes ADD\_PARENTS to add the parent security class GENPRIVS to the HRPRIVS security class.

Example 4-6 invokes REMOVE\_PARENTS to remove the parent security class GENPRIVS from the HRPRIVS security class.

Example 4-7 invokes ADD\_PRIVILEGES to add an aggregate privilege called UPDATE\_INFO to the HRPRIVS security class. The aggregate privilege contains the implied privileges UPDATE, DELETE, and INSERT. Note that ADD\_PRIVILEGES may be used to add several application privileges to a security class. See "Aggregate Privilege" for more information.

Example 4-8 invokes REMOVE\_PRIVILEGES to remove the UPDATE\_INFO application privilege from the HRPRIVS security class.



Example 4-9 invokes REMOVE\_PRIVILEGES to remove all application privileges from the HRPRIVS security class.

Example 4-10 invokes ADD\_IMPLIED\_PRIVILEGES to add a list of implied application privileges for the aggregate privilege UPDATE\_INFO.

Example 4-11 invokes REMOVE\_IMPLIED\_PRIVILEGES to remove the implicit privilege DELETE from the aggregate privilege UPDATE INFO.

Example 4-12 invokes REMOVE\_IMPLIED\_PRIVILEGES to remove all implicit application privileges from the aggregate privilege UPDATE INFO.

The procedure sets a description string for the specified security class. Example 4-13 invokes SET DESCRIPTION to set a description string for the HRPRIVS security class.

Example 4-14 invokes DELETE\_SECURITY\_CLASS to delete the HRACL ACL using the default delete option DEFAULT\_OPTION. Note that this option is defined in "XS\_ADMIN\_UTIL Package".

#### Example 4-5 Adding Parent Security Classes for a Specified Security Class

```
BEGIN
   SYS.XS_SECURITY_CLASS.ADD_PARENTS('HRPRIVS','GENPRIVS');
END:
```

# Example 4-6 Removing One or More Parent Classes for a Specified Security Class

```
BEGIN
   SYS.XS_SECURITY_CLASS.REMOVE_PARENTS('HRPRIVS','GENPRIVS');
END:
```

#### **Example 4-7** Adding One or More Application Privileges to a Security Class

# Example 4-8 Removing One or More Application Privileges from a Specified Security Class

```
BEGIN
   SYS.XS_SECURITY_CLASS.REMOVE_PRIVILEGES('HRPRIVS','UPDATE_INFO');
END;
```

#### Example 4-9 Removing all Application Privileges for a Specified Security Class

```
BEGIN
   SYS.XS_SECURITY_CLASS.REMOVE_PRIVILEGES('HRPRIVS');
END;
```

# Example 4-10 Adding One or More Implied Application Privileges to an Aggregate Privilege

```
BEGIN
SYS.XS_SECURITY_CLASS.ADD_IMPLIED_PRIVILEGES(priv=>'UPDATE_INFO',
```



```
implied_priv_list=>XS$NAME_LIST('"UPDATE"', '"DELETE"', '"INSERT"'));
END;
```

# Example 4-11 Removing a Specified Implied Application Privileges from an Aggregate Privilege

```
BEGIN
SYS.XS_SECURITY_CLASS.REMOVE_IMPLIED_PRIVILEGES('UPDATE_INFO','"DELETE"');
END;
```

# Example 4-12 Removing all Implied Application Privileges from an Aggregate Privilege

```
BEGIN
   SYS.XS_SECURITY_CLASS.REMOVE_IMPLIED_PRIVILEGES('UPDATE_INFO');
END;
```

#### **Example 4-13** Setting a Description String for a Specified Security Class

```
BEGIN
   SYS.XS_SECURITY_CLASS.SET_DESCRIPTION(
    'HRPRIVS','Contains privileges required to manage HR data');
END:
```

#### Example 4-14 Deleting a Specified Security Class

```
BEGIN
   SYS.XS_SECURITY_CLASS.DELETE_SECURITY_CLASS('HRPRIVS', XS_ADMIN_UTIL.DEFAULT_OPTION);
END;
```

# 4.3 About Configuring Access Control Lists

This section contains the following topics:

- About ACLs and ACEs
- Creating ACLs and ACEs
- About Validating Access Control Lists
- Updating Access Control Lists
- About Checking ACLs for a Privilege
- About Using Multilevel Authentication
- Principal Types
- Access Resolution Results
- ACE Evaluation Order
- ACL Inheritance
- About ACL Catalog Views
- About Security Class Catalog Views

## 4.3.1 About ACLs and ACEs

Real Application Security encompasses access control lists (ACLs) and supports grants, denials, and various conflict resolution methods. ACLs are extended to support application-defined privileges, enabling applications to control privileges that are meaningful to it.

Authorization queries are of the form: "Is the application user authorized for privilege p in ACL a?" Application-defined privileges are implemented through APIs supported both in the middle tier and in the database. These APIs enable the application to protect sensitive operations, such as approval of purchase orders.

Before performing a sensitive operation, the application must determine the required application privileges. For example, if the application requires the <code>approvePO</code> application privilege, it must locate the ACL associated with the desired purchase order, <code>a1</code>, and issue a query to determine if the Real Application Security session is authorized for application privilege <code>approvePO</code> in <code>a1</code>. Note that the application must be trusted to properly carry out authorization. Data security improves this by providing a declarative method of associating ACLs with rows in a table; a data security policy allows an administrator or developer to identify a set of rows in a table using an SQL predicate and associates the set with the ACL that is used to control access to its member rows.

The data security system provides a SQL operator that returns the ACLs associated with a row. This SQL operator performs an authorization check using the ACL references associated with the row. By default, a query returns all rows the user is allowed to view; these ACL references may be used in the middle tier to determine appropriate access for a particular row, as arguments in a WHERE clause that limits the result set. Thus, the result set may be further restricted to display only those rows for some specific operations, such as approvePO, based on the user's authorization.

The Real Application Security system provides native enforcement for SQL operations in the database, limiting the scope for damage due to security errors in the application. Thus, a SQL injection attack in one part of the application will not provide access to tables outside of that component.

An ACL protects a resource by specifying application privileges of the principals on the resource. An ACL is a list of access control entries (ACEs) where each ACE maintains the mapping from a principal to a granted or denied application privileges for the resource. A principal may be a database user or Real Application Security application user or application role.

#### **Access Control Entry or ACE**

An **access control entry**, or **ACE**, represents an application privilege grant, and an ACL represents a set of application privilege grants that are bound to a resource. Here, the resource can be a database table, a column in a table, or a set of rows in a table selected using a SQL predicate. Hence when a resource is accessed, only the ACLs associated with the resource are checked for the access right.

An ACE either grants or denies access to some application function or other database data for a particular principal. The ACE does not, itself, specify which data to protect; that is done outside the ACE and the ACL, by associating the ACL with target data.

XS\$ACE\_TYPE type is provided to construct each ACE entry for the ACL. An XS\$ACE\_LIST object consists of a list of privileges and the principal to whom the privileges are granted or denied. ACEs related information can be accessed through DBA XS ACES view.

## 4.3.2 Creating ACLs and ACEs

Example 4-15 creates an ACL called HRACL. This ACL includes ACEs contained in ace list. The application privileges used in ace list are available in the HRPRIVS



security class. The st\_date and en\_date parameters specify the active start and end times for this ACL; note that only the <code>SELECT</code> and <code>VIEW\_SENSITIVE\_INFO</code> application privileges are temporary.

#### Example 4-15 Creating an Access Control List

```
DECLARE
  st date TIMESTAMP WITH TIME ZONE;
  en date TIMESTAMP WITH TIME ZONE;
  ace list XS$ACE LIST;
BEGIN
  st date := SYSTIMESTAMP;
  en_date := TO TIMESTAMP TZ('2019-06-18 11:00:00 -5:00',
                          'YYYY-MM-DD HH:MI:SS TZH:TZM');
  ace_list := XS$ACE LIST(
     XS$ACE TYPE(privilege list=>XS$NAME LIST('"SELECT"','VIEW SENSITIVE INFO'),
                  granted=>true,
                  principal name=>'HRREP',
                  start date=>st date,
                  end date=>en date),
      XS$ACE TYPE(privilege list=>XS$NAME LIST('UPDATE INFO'),
                  granted=>true,
                  principal name=>'HRMGR'),
      XS$ACE TYPE(privilege list=>XS$NAME LIST('"SELECT"'),
                  granted=>true,
                  principal_name=>'DB_HR', principal_type=>XS_ACL.PTYPE_DB));
  sys.xs_acl.create_acl(name=>'HRACL',
                    ace list=>ace list,
                    sec class=>'HRPRIVS',
                    description=>'HR Representative Access');
END;
```

Each ACE includes a principal that is the target of the grant and a list of application privileges. The grant is subject to the following attributes described in these topics:

- Deny
- Invert
- ACE Start-Date and End-Date

## 4.3.2.1 Deny

When a grant is negated, the application privileges are denied. Example 4-16 sets the value of the attribute granted to FALSE to deny application privileges to the principal. The default value is TRUE.

Real Application Security ACL supports only the ordered evaluation of ACEs. The first ACE that grants or denies the requested application privilege contributes toward the final grant or deny. See section "DBA\_XS\_ACES".

### Example 4-16 Denying a Privilege



## 4.3.2.2 Invert

When the specified application privileges are given to all principals except one, that principal is inverted; the inverted attribute is set to TRUE. The default value of the attribute inverted is FALSE. In Example 4-17, a grant made to the inverted role HRGUEST provides the application privileges to any user that does not have the role enabled.

#### Example 4-17 Inverting an Application Privilege

## 4.3.2.3 ACE Start-Date and End-Date

Each ACE can have a time constraint based on a start-date and an end-date, specifying the time when the ACE is in effect.

In Example 4-18, the optional attributes start\_date and end\_date (of datatype TIMESTAMP WITH TIME ZONE) define the time period over which an ACE is valid. The end date value must be greater than the start date value.

#### Example 4-18 Setting ACE Start-Date and End-Date

## 4.3.3 About Validating Access Control Lists

Oracle recommends that you always validate the Real Application Security objects after administrative configuration changes. The XS\_DIAG package provides a set of validation APIs to help ensure that these changes do not damage the complicated relationships among your Real Application Security objects.

See "VALIDATE\_ACL Function" for more information about validating an ACL.

## 4.3.4 Updating Access Control Lists

To manipulate ACLs, use the procedures in PL/SQL package XS\_ACL; it contains procedures that create and manage ACLs. See "XS\_ACL Package".

Example 4-19 invokes APPEND\_ACES to add an ACE, ace\_entry, to the HRACL ACL. The ACE grants the SELECT privilege to the DB HR database user.

Example 4-20 invokes REMOVE ACES to remove all ACEs from the ACL called HRACL.

The procedure sets or modifies the security class for an ACL. Example 4-21 invokes SET\_SECURITY\_CLASS procedure to associate the HRPRIVS security class with ACL HRACL.



Example 4-22 invokes SET\_PARENT\_ACL to set the AllDepACL ACL as the parent ACL for the HRACL ACL. The inheritance type is set to EXTEND.

Example 4-23 invokes REMOVE ACL PARAMETERS to remove all ACL parameters for ACL1.

Example 4-24 invokes REMOVE ACL PARAMETERS to remove the REGION parameter for ACL1.

Example 4-25 invokes SET DESCRIPTION to set a description for ACL HRACL.

Example 4-26 invokes DELETE ACL to delete ACL HRACL using the default delete option.

#### Example 4-19 Appending an ACE to an Access Control List

#### Example 4-20 Removing all ACEs from an ACL

```
BEGIN
SYS.XS_ACL.REMOVE_ACES('HRACL');
END;
```

#### Example 4-21 Modifying the Security Class for an ACL

```
BEGIN
   SYS.XS_ACL.SET_SECURITY_CLASS('HRACL','HRPRIVS');
END;
```

#### Example 4-22 Setting or Modifying the Parent ACL

```
BEGIN
   SYS.XS_ACL.SET_PARENT_ACL('HRACL', 'AllDepACL', XS_ACL.EXTENDED);
END;
```

#### Example 4-23 Removing all ACL Parameters for an ACL

```
BEGIN
   SYS.XS_ACL.REMOVE_ACL_PARAMETERS('ACL1');
END;
```

#### Example 4-24 Removing the Specified ACL Parameter for an ACL

```
BEGIN
   SYS.XS_ACL.REMOVE_ACL_PARAMETERS('ACL1', 'REGION');
END:
```

#### Example 4-25 Setting a Description String for an ACL



#### Example 4-26 Deleting an ACL

```
BEGIN
SYS.XS_ACL.DELETE_ACL('HRACL');
END.
```

## 4.3.5 About Checking ACLs for a Privilege

There are two forms of enforcement; the system enforces DML privileges on data security protected objects, and the SQL operator added by the user enforces all other application privileges.

To check an ACL for an application privilege, call the SQL operator ORA\_CHECK\_ACL:

```
ORA CHECK ACL (acls, privilege [,privilege] ...)
```

The ORA\_CHECK\_ACL SQL operator evaluates the list of application privileges with respect to an ordered list of ACLs. The evaluation process proceeds until any one of the following three events occurs:

- A grant is encountered for every application privilege specified before any potential denials of the same application privilege. The outcome is that the application privileges are granted.
- One of the application privileges specified is denied before any potential grants. The outcome is that at least one of the application privileges is denied.
- The list of ACEs is fully traversed. The outcome is that not all of the application privileges are granted.

To evaluate the application privilege, Oracle checks the ACEs (which are kept in order), and the evaluation stops when it finds an ACE that grants or denies the requested application privileges.

To find the ACLs associated with rows of a table or view, call the SQL operator ORA\_GET\_ACLIDS: ORA\_GET\_ACLIDS (table, ...). For example, to enforce an application privilege, priv, on a table, tab, the user query adds the following check:

```
ORA CHECK ACL(ORA GET ACLIDS(tab), priv)
```

This function answers the question whether application privileges were granted, denied, or neither. A corresponding Java API is also available.

## 4.3.6 About Using Multilevel Authentication

**Multilevel authentication** enables the user to specify, through system-constraining ACLs, application privileges based on levels of authentication. A **system-constraining ACL** specifies a minimum application-wide set of application privileges on objects, based on dynamic roles that reflect an application user's level of authentication. When attempting to access an object, an application user may be either strongly or weakly authenticated, either inside or outside the firewall, with the following four possible levels of authentication:

- Strongly authenticated, inside firewall
- Strongly authenticated, outside firewall
- Weakly authenticated, inside firewall
- Weakly authenticated, outside firewall



A system-constraining ACL can specify application privileges that apply to application users at each level of authentication in an application. Based on application requirements, the administrator may grant additional application privileges to specific users based on any necessary criteria; such additional application privileges are independent of any system-constraining ACL. Example 4-28 and Example 4-29 implement a system-constraining ACL.

## 4.3.7 Principal Types

In addition to Real Application Security principals, application users and application roles, Real Application Security supports grants based on database users and roles. When the system evaluates an ACL in a context of a Real Application Security session, it ignores grants that are based on a database schema, but honors grants that are based on database role because they are part of Real Application Security user's role list. Within an ACL, multiple ACEs can grant privileges to a principal.

## 4.3.8 Access Resolution Results

Requests for access can have two possible results: true or false.

- A result of true means that the requested application privilege is granted
- A result of false means that the requested application privilege is either not granted or denied.

## 4.3.9 ACE Evaluation Order

ACEs are evaluated in the order they appear in the ACL. The outcome of evaluating a particular ACE may be one of the following:

- The application privilege is granted.
- The application privilege is denied.
- The application privilege is neither granted nor denied.

Note that if an ACE grants an application privilege that a previous ACE denies, the result is a deny because the ACEs are evaluated in order.

## 4.3.10 ACL Inheritance

ACLs can explicitly inherit from a single parent ACL, enabling the application to share policies across multiple objects. When the request for an application privilege involves two ACLs, the final result of the access-resolution algorithm may be based on semantics of individual access-resolution results of the ACLs. Real Application Security supports two types of inheritance semantics: extending ACL inheritance (OR with ordered evaluation), and constraining ACL inheritance (AND).

This section contains:

- Extending ACL Inheritance
- Constraining ACL Inheritance

## 4.3.10.1 Extending ACL Inheritance

Extending ACL inheritance (OR with ordered evaluation) dictates that the ACEs are evaluated from the bottom of the inheritance tree to its top, from child to parent. In extending ACL

inheritance, an application privilege is granted if either child or parent ACL grants the privilege, and denied if either the child or parent ACL denies the privilege. In fact, the first ACL that explicitly grants or denies the requested application privilege determines the final result. After the first grant or deny, further evaluations of the remaining ACLs are not attempted. Note that this evaluation rule is the same as the ordered evaluation of ACEs within an ACL.

The following example sets the AllDepACL ACL as the parent ACL for the HRACL ACL. The inheritance type is set to EXTENDED.

#### **Example 4-27 Extending ACL Inheritance**

```
BEGIN
   SYS.XS_ACL.SET_PARENT_ACL('HRACL','AllDepACL',XS_ACL.EXTENDED);
END:
```

## 4.3.10.2 Constraining ACL Inheritance

Constraining ACL inheritance (AND) requires that both the child and the parent ACL grant the application privilege so that the ACL check evaluates to true.

Application-wide security policies can be enforced if all the ACLs for an application are constrained by the same parent ACL. For example, imagine a sample policy where users who are authenticated as being inside the corporate firewall can have application privileges in addition to the SELECT privilege. Example 4-28 shows the constraining ACL for this policy (inheritance type is set to CONSTRAINED), where all application users with XSPUBLIC application role are granted the SELECT privilege. Note that only the application users who are inside the corporate firewall have the dynamic application role FIREWALL enabled. Therefore, application users inside the firewall are granted all the application privileges in HRPRIVS security class. As this ACL constrains all the ACLs, such as guestacl, Example 4-29 shows that the application privilege grants of these ACLs are constrained by FIREWALL ACL.

# Example 4-28 Constraining ACL Inheritance: Firewall-Specific Authentication Privilege

```
DECLARE
 ace list XS$ACE LIST;
BEGIN
  ace list := XS$ACE LIST(
                 XS$ACE TYPE(privilege list=>XS$NAME LIST('"SELECT"'),
                             granted=>true,
                             principal name=>'XSPUBLIC'),
                 XS$ACE TYPE(privilege list=>XS$NAME LIST('ALL'),
                             granted=>true,
                             principal name=>'FIREWALL'));
  sys.xs acl.create acl(name=>'FIREWALL ACL',
                    ace list=>ace list,
                    sec class=>'HRPRIVS',
                    description=>'Only select privilege if not inside firewall');
END;
/
SYS.XS ACL.SET PARENT ACL('GuestACL', 'FIREWALL ACL', XS ACL.CONSTRAINED);
END;
```



#### Example 4-29 Using a Constraining Application Privilege

```
SQL> select ACE_ORDER, GRANT_TYPE, PRINCIPAL, PRIVILEGE
    from DBA_XS_ACES
    where ACL='FIREWALL ACL';
```

ACE_ORDER	GRANT_TYPE	PRINCIPAL	PRIVILEGE
1	GRANT	XSPUBLIC	SELECT
2	GRANT	FIREWALL	ALL

## 4.3.11 About ACL Catalog Views

ACLs have the following catalog views:

- DBA XS ACLS catalog view, described in section "DBA\_XS\_ACLS"
- DBA XS ACES catalog view, described in section "DBA\_XS\_ACES"

## 4.3.12 About Security Class Catalog Views

Security classes have the following catalog views:

- DBA\_XS\_SECURITY\_CLASSES, described in section "DBA\_XS\_SECURITY\_CLASSES"
- DBA XS SECURITY CLASS DEP, described in "DBA\_XS\_SECURITY\_CLASS\_DEP"
- DBA XS PRIVILEGES, described in "DBA\_XS\_PRIVILEGES"

## 4.4 Data Security

Data security associates ACLs with a logical group of rows, known as a data realm.

This enables applications to define and enforce application-specific privileges at the database layer, through policies that define data realms and their access. These data realms include both a SQL predicate that identifies a set of rows and an ACL that protects the identified rows. The ACL evaluation is based on the application user, not the schema owner.

This section includes the following topics:

- Data Realms
- Parameterized ACL

### 4.4.1 Data Realms

Real Application Security's Data Security policy **data realms** associate ACLs with rows in a table. A data realm has two parts:

- 1. A rule expressed as a SQL predicate, which selects a set of rows.
- 2. A set of ACLs, which specify access policies on the rows.

Data Security manages DML Real Application Security application privileges granted by the associated ACLs. The <code>DataSecurity</code> module does not inherently enforce other (non-DML) Real Application Security application privileges. Such application privilege may be enforced programmatically as part of a DML operation, when invoking the <code>CHECK\_PRIVILEGE</code> operator inside either the SQL operator or data realm predicate.



## 4.4.2 Parameterized ACL

Because each data realm defines a rule that uses a set of parameters, different values for these parameters select different rows. These sets of rows may require different ACLs. Therefore, association between an ACL and a set of rows depends on the data realm rule and its parameter names and values.

## 4.5 ACL Binding

In the database, a privilege may be bound to a resource in the following manner:

- It can be explicitly bound as part of a privilege grant. For example, database object privileges are bound to a resource as part of a privilege grant, such as GRANT user Nupdate ON table M.
- It may also be globally bound as part of the privilege definition, such as a system privileges ALTER SYSTEM or CREATE ANY TABLE, which do not require the resource name as part of their grant statement.

Similarly, a Real Application Security application privilege can be one of these types:

- Explicitly bound through an ACL and data realms as part of Data Security policies;
   see Configuring Data Security
- Globally bound to a resource as part of its definition



5

# **Configuring Data Security**

#### This chapter contains:

- About Data Security
- · About Validating the Data Security Policy
- Understanding the Structure of the Data Security Policy
- About Designing Data Realms
- Applying Additional Application Privileges to a Column
- About Enabling Data Security Policy for a Database Table or View
- About Creating Real Application Security Policies on Master-Detail Related Tables
- About Managing Application Privileges for Data Security Policies
- Using BEQUEATH CURRENT\_USER Views
- Real Application Security: Putting It All Together
- About Schema Level Real Application Security Policy Administration

## 5.1 About Data Security

Data security refers to the ability to control application user access to data in an Oracle database throughout all components of an Oracle Enterprise, using a uniform methodology. In Oracle Database Real Application Security, to secure a database table or view, you must specify the rows that you want to secure by creating a data realm (see also, data realm).

To restrict access to the data realm, you associate one or more access control lists (ACLs) that list the application users or application roles and their application privileges for each data realm. A data realm together with its associated ACL is known as a data realm constraint.

You can further restrict access to specific columns by applying one or more application privileges to each column. This is useful in situations where you want only privileged application users to see the data in that column.

Data security is an extension of Oracle Virtual Private Database (VPD). VPD adds a WHERE predicate to restrict data access each time an application user selects or modifies a database table. For more information about VPD, see *Oracle Database Security Guide*. Oracle Database Real Application Security extends VPD concepts further by implementing an authorization model that can further restrict access at both the row and column by means of associating ACLs to these objects. In addition, the application session and session context (through user roles and session namespace) are made more secure. Furthermore Real Application Security provides its own data dictionaries.

To configure data security in Oracle Database Real Application Security, you must follow these steps:

1. Create a data security policy. The data security policy defines one or more data realms and associates ACLs for each data realm to create data realm constraints. The data security policy can also contain column-specific attributes to further control data access.

Multiple tables or views can share the same data security policy. This lets you create a uniform security strategy that can be used across a set of tables and views

Example 5-1 Shows the structure a data security policy.

2. Associate the data security policy with the table or view you want to secure.

You can run the <code>XS\_DATA\_SECURITY.APPLY\_OBJECT\_POLICY</code> PL/SQL procedure to enable the data security policy for the table or view that contains the data realms and columns that you want to secure.

Note that if your application security requires that you update table rows and also restrict read access to certain columns in the same table, you must use two APPLY\_OBJECT\_POLICY procedures to enforce both data security policies. For example, one APPLY\_OBJECT\_POLICY procedure would enforce the DML statement\_types required for updating table rows (for example, INSERT, UPDATE, DELETE), while the other APPLY\_OBJECT\_POLICY procedure would enforce only the statement types of SELECT for the column constraint.

Example 5-5 shows how to use the APPLY\_OBJECT\_POLICY procedure. See "APPLY\_OBJECT\_POLICY Procedure" for more information.

Validate the data security policy. See "About Validating the Data Security Policy" for more information.

## 5.2 About Validating the Data Security Policy

Oracle recommends that you should always validate the Real Application Security objects after administrative configuration changes. The XS\_DIAG package provides a set of validation APIs to help ensure that the complicated relationships among your Real Application Security objects are not damaged unintentionally by these changes.

See "VALIDATE\_DATA\_SECURITY Function" for more information about validating a data security policy.

# 5.3 Understanding the Structure of the Data Security Policy

You can create a data security policy using the <code>XS\_DATA\_SECURITY.CREATE\_POLICY</code> PL/SQL procedure.

Figure 5-1 shows the structure of a Real Application Security data security policy named HR.EMPLOYEES\_DS that is created from a data realm constraint and a column constraint, both of which are to be applied to the EMPLOYEES table. The data realm constraint defines the rows (DEPARTMENT\_ID with a value of 60 or 100) on which the data security policy applies and the ACL (HRACL) that is associated with these rows. The column constraint defines a constraint for the sensitive column data in the SALARY column of the EMPLOYEES table by using the VIEW\_SENSITIVE\_INFO privilege that is required to view this sensitive data.



Security Classes **EMPLOYEES Data Security Policies** HR.EMPLOYEES\_SC HR.EMPLOYEES\_DS DEPARTMENT\_ID **Privileges** SALARY **Realm Constraints** SELECT Data Realm Constraints VIEW\_SENSITIVE\_INFO DEPARTMENT\_ID in (60, 100) **ACLs** Column Constraints HR.HRACL 'SALARY' Grant SELECT to Employee\_Role Grant SELECT, VIEW\_SENSITIVE\_INFO to Manager\_Role

Figure 5-1 Real Application Security Data Security Policy Created on the EMPLOYEES Table

Example 5-1 creates the data security policy shown in Figure 5-1.

```
See Also:

"CREATE_POLICY Procedure"
```

You should validate the data security policy after you create it. See "VALIDATE DATA SECURITY Function" for more information.

The main parameters of a data security policy are as follows:

- Policy Name: This defines the name of the data security policy.
   Example 5-1 uses the name EMPLOYEES DS for the data security policy that it creates.
- **Data Realm Constraints:** The data realm constraints define the data realms, or the rows, on which the data security policy applies, together with the ACLs to be associated with these data realms.

Example 5-1 uses the realm\_cons list to define the data realm constraint for the EMPLOYEES\_DS policy. realm\_cons comprises of rows that have a DEPARTMENT\_ID value of 60 or 100. These rows are associated with the HRACL access control list.

• **Column Constraint:** Column constraint defines additional constraint for sensitive column data in the data realm constraint.

Example 5-1 associates the column\_cons column constraint with the EMPLOYEES\_DS policy. column cons protects the SALARY column with the VIEW SENSITIVE INFO privilege.

#### Example 5-1 Structure of a Data Security Policy

-- Create the ACL HRACL. DECLARE



```
ace list XS$ACE LIST;
BEGIN
ace list := XS$ACE LIST(
XS$ACE TYPE(privilege list => XS$NAME LIST('SELECT'),
granted => true,principal name => 'Employee Role'),
XS$ACE_TYPE(privilege_list => XS$NAME_LIST('SELECT', 'VIEW SENSITIVE INFO'),
granted => true, principal_name => 'Manager_Role'));
sys.xs_acl.create_acl(name => 'HRACL',ace_list => ace_list, sec_class =>
'HR.EMPOLYEES SC');
END:
-- Create variables to store the data realm constraints and the column
DECLARE
 realm cons XS$REALM CONSTRAINT LIST;
-- Create a data realm constraint comprising of a data realm (rule) and
-- an associated ACL.
 realm cons :=
    XS$REALM CONSTRAINT LIST(
     XS$REALM CONSTRAINT TYPE (realm=> 'DEPARTMENT ID in (60, 100)',
                               acl list=> XS$NAME LIST('HRACL')));
-- Create the column constraint.
  column cons :=
    XS$COLUMN CONSTRAINT LIST(
      XS$COLUMN_CONSTRAINT_TYPE(column_list=> XS$LIST('SALARY'),
                            privilege=> 'VIEW SENSITIVE INFO'));
 -- Create the data security policy.
  SYS.XS DATA SECURITY.CREATE POLICY(
          name=>'HR.EMPLOYEES DS',
          realm constraint list=>realm cons,
          column_constraint_list=>column_cons);
-- Enforce the data security policy to protect READ access of the EMPLOYEES table
-- and restrict access to the SALARY column using the VIEW SENSITIVE INFO
-- privilege.
  sys.xs data security.apply object policy(
           policy => 'HR.EMPLOYEES DS',
           schema => 'HR',
           object => 'EMPLOYEES',
           statement types => 'SELECT',
           owner bypass => true);
END;
```

# 5.4 About Designing Data Realms

This section includes the following topics:

- About Understanding the Structure of a Data Realm
- About Using Static Data Realms
- Using Trace Files to Check for Policy Predicate Errors

## 5.4.1 About Understanding the Structure of a Data Realm

A data realm is a collection of one or more object instances. An object instance is associated with a single row in a table or view and is identified by the primary key value of the row in the storage table of the object. A table can have both static and dynamic data realms defined for it at the same time. As described earlier, an ACL defines the application privilege grants for the data realm.

A data realm constraint is used to associate a data realm with an ACL. Example 5-2 creates a data realm constraint called  $realm\_cons$ . The data realm constraint includes a membership rule to create a data realm. The data realm includes rows where <code>DEPARTMENT\_ID</code> is 60 or 100. realm cons also declares an ACL, called <code>HRACL</code>, to associate with the data realm.

The membership of the object instances within a data realm is determined by a rule in the form of a SQL predicate, which must be applicable to the WHERE clause of a single-table query against the storage table of the object. The SQL predicate in Example 5-2 is DEPARTMENT\_ID in (60, 100).

If the SQL you write causes errors, such as ORA-28113: policy predicate has error, then you can use trace files to find cause of the error. See "Using Trace Files to Check for Policy Predicate Errors" for more information.

Example 5-2 uses a single ACL called HRACL. A data realm can be associated with multiple ACLs, and the same ACL can be used across multiple data realms.

Consider the following columns from the <code>ORDERS</code> purchase order table in the <code>OE</code> sample schema:

ORDER_ID	CUSTOMER_ID	ORDER_STATUS	SALES_REP_ID	ORDER_TOTAL
2354	104	0	155	46257
2355	104	8	NULL	94513.5
2356	105	5	NULL	29473.8
2357	108	5	158	59872.4
2358	105	2	155	7826

Each row in the ORDERS table is an object instance in the purchase order object. The number listed in the ORDER\_ID column is the primary key used to uniquely identify a particular purchase order object instance. For example:

- A data realm comprised of one object instance, that is, one row. For example, you could use the <code>where predicate of ORDER\_ID=2354</code>.
- A data realm comprised of multiple object instances. For example, you could have multiple rows using the WHERE predicate of CUSTOMER ID=104.
- A data realm comprised of the entire contents of the table, defined by the WHERE predicate
  of 1=1.

Examples of ways to define data realms are as follows:

Use valid SQL attributes such as columns in a table.
 In this case, you are using WHERE predicates such as the following:



```
CUSTOMER ID=104
```

Changes made to the data in the rows and columns are automatically reflected in the data collected by the data realm.

Use parameters in the WHERE predicate.

You can parameterize an data realm, for example:

```
CUSTOMER ID=&PARAM
```

This example assumes that the parameter PARAM has been associated with different customer IDs. When you grant permissions in this situation, you need to grant the permission to the specific parameter value. You must specify the values of the parameters in the ACL associated with the data realm that contains this type of WHERE predicate. This enables you to create the grant based on customer IDs without having to create many customer ID-specific data realms.

 Use a membership rule based on runtime application session variables or subqueries.

An example of this type of membership rule is:

```
CUSTOMER ID=XS SYS CONTEXT('order', 'cust id')
```

However, be careful about creating membership rules that are based on session variables or subqueries. For example, suppose you wanted to use the session variable USER, which reflects the current application user, in the membership rule col=USER. Oracle Database cannot pre-compute the resultant row set because the result is not deterministic. Application user SCOTT and application user JSMITH may have a different result for the same row. However, the membership rule col='SCOTT' works because the rule is always evaluated to the same result for any given row.

See "About Using Static Data Realms" for more information about creating data realms. See also "XS\_SYS\_CONTEXT Function" for more information about XS\_SYS\_CONTEXT.

#### Example 5-2 Components of a Data Realm Constraint

## 5.4.2 About Using Static Data Realms

In a static data realm, Oracle Database evaluates changes to data affected by a data realm when the data is updated. You can use static data realms with tables, but not with views.

To set an data realm to be static, set its <code>is\_static</code> attribute to <code>true</code>. The following example creates a static data realm:

Materialized Views (MVs) will be used to maintain the binding between rows in the protected table and the ACLs that protect them. They will be generated automatically whenever static data realms are included in the data security policy. These MVs will

support complete refresh only and will allow up to 125 ACLs to be associated with any single row.

The MV that is generated will be of the form mv (TABLEROWID, ACLIDLIST) where TABLEROWID refers to a row in the table being protected and ACLIDLIST is a list of ACLID values stored in a RAW type column. The individual 16-byte values will be concatenated to form the list.

Oracle Database evaluates dynamic data realms each time the application user performs a query on the data realm data. You can use dynamic data realms to protect rows for both tables and views. A dynamic data realm has the most flexibility, because it is not bound by the requirements needed for static data realms. Be aware that an overly complex rule within the dynamic data realm definition may affect performance.

If the base table update is infrequent or the data realm member evaluation rule is complex, then you should consider using static data realms to protect the base table. A frequently updated base table may be constantly out of sync with the ACLIDS storage MV, unless the MV is refreshed accordingly. The administrator should make the decision based on the base table statistics and performance requirements of the system.

To set a data realm constraint to be dynamic, set its <code>is\_static</code> attribute to <code>FALSE</code>, or omit the is static attribute. The following example creates a dynamic data realm:

## 5.4.3 Using Trace Files to Check for Policy Predicate Errors

If the SQL defined in the <code>realm</code> element causes an <code>ORA-28113</code>: policy <code>predicate</code> has <code>error</code> or similar message, then you can use trace files to find the cause of the error. The trace file shows the actual error, along with the VPD view showing the reason for the problem. Often, the syntax of the view has a trivial error, which you can solve by analyzing the SQL text of the view.

To enable tracing, log into SQL\*Plus as a user who has the ALTER SESSION privilege.

If you want to dump all the data realm constraint rules (with their parameter values resolved) into the trace file, enter the following statement:

```
ALTER SESSION SET EVENTS 'TRACE[XSXDS] disk=high';
```

If you want to dump the VPD views of the XDS-enabled table during the initial (hard) parse of a guery, enter the following statement:

```
ALTER SESSION SET EVENTS 'TRACE[XSVPD] disk=high';
```

Alternatively, you can enable tracing by adding the following lines to the initialization file for the database instance:

```
event="TRACE[XSXDS] disk=high"
event="TRACE[XSVPD] disk=high"
```

You can find the location of this trace file by issuing the following SQL command:

```
SHOW PARAMETER USER_DUMP_DEST;
```

If you need to disable tracing, issue the following statements:

```
ALTER SESSION SET EVENTS 'TRACE[XSVPD] off';
ALTER SESSION SET EVENTS 'TRACE[XSXDS] off';
```



#### See Also:

- "About Data Security (XSXDS and XSVPD) Event-Based Tracing"
- Oracle Database Administrator's Guide for more information about using trace files

## 5.5 Applying Additional Application Privileges to a Column

By default, access to rows is protected by the ACL associated with the data realm. In addition, you can protect a particular column with custom application privileges.

To protect a column for table  $\mathbb{T}$ , add a list of column constraints to the data security policy that will be applied to table  $\mathbb{T}$ .

#### Note:

Starting from Oracle Database 23c, the absolute maximum number of columns in a table is 1000 or 4096, depending on the value of the MAX\_COLUMNS initialization parameter.

- When this parameter is set to STANDARD, the maximum number of columns allowed in a database table or view is 1000.
- When this parameter is set to EXTENDED, the maximum number of columns allowed in a database table or view is 4096.

For tables approaching 4096 columns (1000 in case of STANDARD), there is a limitation on the number of columns that can be protected as Real Application Security uses an internal virtual column to compute and store the authorization indicator. The sum of the number of columns and the number of protected columns should not exceed 4096 (1000 in case of STANDARD). Number of table columns + Number of protected table columns <=4096. For example, if a table has 4094 columns, up to and including two protected columns are allowed; If the number of columns to be protected exceeds the number allowed, an ORA-28113: policy predicate has error is returned.

For example, the PRODUCT\_INFORMATION table in the OE schema contains the LIST\_PRICE column. If you want to restrict the display of product prices to specific categories, you can apply an additional application privilege to the LIST\_COLUMN table, so that only the sales representative who has logged in can see the product list prices for the categories they manage.

**Example 5-3 shows a column constraint that protects the LIST\_PRICE column with the** ACCESS PRICE application privilege.

Before you add the column constraint, a <code>SELECT</code> statement on the following columns from the <code>OE.PRODUCT\_INFORMATION</code> table for products in categories 13 and 14 shows the following output:



PRODUCT_ID	PRODUCT_NAME	CATEGORY_ID	LIST_PRICE
3400	HD 8GB /SE	13	389
3355	HD 8GB /SI	13	NULL
2395	32MB Cache /M	14	123
1755	32MB Cache /NM	14	121

After the column constraint is applied, the sales representatives who are responsible for category 13 products see the following output:

PRODUCT_ID	PRODUCT_NAME	CATEGORY_ID	LIST_PRICE
3400	HD 8GB /SE	13	389
3355	HD 8GB /SI	13	NULL
2395	32MB Cache /M	14	NULL
1755	32MB Cache /NM	14	NULL

Conversely, sales representatives responsible for category 14 products see this output:

PRODUCT_ID	PRODUCT_NAME	CATEGORY_ID	LIST_PRICE
3400	HD 8GB /SE	13	NULL
3355	HD 8GB /SI	13	NULL
2395	32MB Cache /M	14	123
1755	32MB Cache /NM	14	121

In these examples, the list price for product 3355 is <code>NULL</code>. To enable a mid-tier application to distinguish between the true value of authorized data, which could include <code>NULL</code>, and an unauthorized value that is always <code>NULL</code>, use the <code>COLUMN\_AUTH\_INDICATOR</code> SQL function to check if the column value in a row is authorized. You can mask the unauthorized data with a value different from <code>NULL</code> by modifying the <code>SELECT</code> statement to include a <code>DECODE</code> or <code>CASE</code> function that contains the <code>COLUMN\_AUTH\_INDICATOR</code> SQL function.

Example 5-4 shows a SELECT statement that uses the COLUMN\_AUTH\_INDICATOR function to check authorized data and the DECODE function to replace NULL with the value restricted.

Afterward, the masked value appears in place of NULL. For example, if our category 13 sales representative logs on and searches for product list prices, they see the following output:

PRODUCT_ID	PRODUCT_NAME	CATEGORY_ID	LIST_PRICE
3400	HD 8GB /SE	13	389
3355	HD 8GB /SI	13	NULL
2395	32MB Cache /M	14	restricted
1755	32MB Cache /NM	14	restricted
<b></b>			



#### See Also:

- Oracle Database Real Application Security Data Dictionary Views for information about the column constraints data dictionary views, which list existing tables that use column level security
- "COLUMN\_AUTH\_INDICATOR Function"
- Example 5-1 for an example of a column constraint element within a data security policy.
- Configuring OCI and JDBC Applications for Column Authorization if your applications use either Oracle Call Interface (OCI) or JDBC

# Example 5-3 Column with an Additional Application Privilege That Has Been Applied

#### **Example 5-4 Checking Authorized Data and Masking NULL Values**

```
SELECT PRODUCT_ID, PRODUCT_NAME, CATEGORY_ID
DECODE(COLUMN_AUTH_INDICATOR(LIST_PRICE), 0, 'restricted', 1, LIST_PRICE)
LIST_PRICE
FROM PRODUCT_INFORMATION
WHERE CATEGORY ID = 13;
```

# 5.6 About Enabling Data Security Policy for a Database Table or View

The XS\_DATA\_SECURITY.APPLY\_OBJECT\_POLICY procedure applies a data security policy on a table or view.

This section includes the following topics:

- Enabling Real Application Security Using the APPLY\_OBJECT\_POLICY Procedure
- About How the APPLY\_OBJECT\_POLICY Procedure Alters a Database Table
- About How ACLs on Table Data Are Evaluated

# 5.6.1 Enabling Real Application Security Using the APPLY\_OBJECT\_POLICY Procedure

Use the XS\_DATA\_SECURITY.APPLY\_OBJECT\_POLICY procedure to enable Real Application Security for a database table or view. Example 5-5 enables the ORDERS\_DS data security policy for the OE.ORDERS table. See "APPLY\_OBJECT\_POLICY Procedure" for more information.



#### Example 5-5 Using XS\_DATA\_SECURITY.APPLY\_OBJECT\_POLICY

This section includes the following topic: About Applying Multiple Policies for a Table or View.

### 5.6.1.1 About Applying Multiple Policies for a Table or View

You can apply multiple data security policies for a table or view. When a table or view is protected by multiple data security policies, an application user has access to only those rows that are allowed by all the policies. So, for example, if the data realm for Policy 1 includes a row, but the data realm for Policy 2 does not include the same row, the application user would be unable to access the row.

Column security works similarly. Consider the case where column Col1 is protected by multiple policies: Policy1 protects it with Priv1, Policy2 protects it with Priv2, and so forth. Then an application user must have been granted all application privileges (Priv1, Priv2, and so forth) to access Col1. Thus, for columns protected by column policies, an application user must have been granted access by all policies protecting the column.

# 5.6.2 About How the APPLY\_OBJECT\_POLICY Procedure Alters a Database Table

The following table, OE.ORDERS, shown earlier under "About Understanding the Structure of a Data Realm", has been enabled with XS\_DATA\_SECURITY.APPLY\_OBJECT\_POLICY. It shows the addition of the hidden SYS\_ACLOID column. This column, whose data type is NUMBER, lists application user-managed ACL identifiers. The following table contains the application user-managed ACL identifier 500, which is a direct grant on the object instance identified by the order ID 2356.



The SYS\_ACLOID hidden column can be enabled by passing the value XS\_DATA\_SECURITY.APPLY\_ACLOID\_COLUMN for the apply\_option parameter when invoking the XS\_DATA\_SECURITY procedure. Real Application Security allows only one ACLID to be added to the SYS\_ACLOID column.

ORDER_ID	CUSTOMER_ID	ORDER_STATUS	SALES_REP_ID	ORDER_TOTAL	SYS_ALCOID
2354	104	0	155	46257	
2355	104	8	NULL	94513.5	
2356	105	5	NULL	29473.8	500
2357	108	5	158	59872.4	
2358	105	2	155	7826	

The system-managed static ACL identifiers, are stored in a Materialized View (MV).



TABLEROWID	ACLIDLIST
AAAO/8AABAAANrCABJ	60FB8AAA40D46C9EE040449864653987
AAAO/8AABAAANrCABL	60FB8AAA40D46C9EE040449864653987

To find detailed information on the data realms or data realm constraints associated with a table, query the <code>DBA\_XS\_REALM\_CONSTRAINTS</code> data dictionary view. See "DBA\_XS\_REALM\_CONSTRAINTS" for more information.

### 5.6.3 About How ACLs on Table Data Are Evaluated

When Oracle Database evaluates a set of ACLs, it stops the evaluation when it finds the first grant or deny. For this reason, it is important to plan the order of ACLs carefully. The ACLs associated with each row in a table are evaluated in the following order:

- The ACLs from grants directly on object instances (that is, application usermanaged ACL identifiers) are evaluated first. See "About Configuring Access Control Lists" for more information about creating an ACL and adding it to the object instance.
- 2. The ACLs from static data realm constraint grants are evaluated next, after application user-managed ACLs. If you have multiple static data realms, they are evaluated in the order of their physical appearance in the data security policy. See "About Using Static Data Realms" for more information about static data realms.
- 3. The ACLs from dynamic data realm constraint grants are evaluated last. If you have multiple dynamic data realms, they are evaluated in the order of their physical appearance in the policy. See "About Using Static Data Realms" for more information about dynamic data realms.

# 5.7 About Creating Real Application Security Policies on Master-Detail Related Tables

This section includes the following topics:

- About Real Application Security Policies on Master-Detail Related Tables
- About Understanding the Structure of Master Detail Data Realms
- Example of Creating a Real Application Security Policy on Master-Detail Related Tables

For more information about master-detail tables, see the chapter about creating a master-detail application using JPA and Oracle ADF in *Oracle Database Get Started with Java Development*.

# 5.7.1 About Real Application Security Policies on Master-Detail Related Tables

You can create a data security policy that can be used for master-detail related tables. Typically, you may want the same policy that protects the master table to protect its



detail tables. Creating a Real Application Security policy for master-detail tables enables anyone accessing these tables to do so under a uniform policy that can be inherited from master table to detail table.

The possible inheritance paths for policies and master-detail tables are as follows:

- Multiple detail tables can inherit policies from one master table.
- Detail tables can inherit policies from other detail tables.
- One detail table can inherit policies from multiple master tables.

If any one of the policies in the master table is satisfied, then application users can access the corresponding rows in the detail table.

## 5.7.2 About Understanding the Structure of Master Detail Data Realms

To create a Real Application Security policy for master-detail related tables, you must create a data security policy for each table. In each data security policy for the detail tables, you indicate the master table from which the detail table inherits by including master detail data realms. Steps 4, 6 and 7 in the procedure under "Example of Creating a Real Application Security Policy on Master-Detail Related Tables" shows examples of creating and using master-detail data realms and creating and applying master-detail data security policies to master-detail tables.

Example 5-6 shows a sample master detail data realm.

In this specification:

- when\_condition specifies a predicate for the detail table, similar to a WHERE clause, to
  filter data. If when\_condition evaluates to true, then Oracle Database applies the master
  policy. This element is optional.
- parent schema specifies the name of the schema that contains the master table.
- parent object specifies the name of the master table.
- primary key specifies the primary key from the master table.
- foreign key specifies the foreign key of the detail table.

#### Example 5-6 A Master Detail Data Realm

## 5.7.3 Example of Creating a Real Application Security Policy on Master-Detail Related Tables

This example uses the SH sample schema. The SH schema has a table called CUSTOMERS, which is the master table. The master table CUSTOMERS has a detail table called SALES, and another detail table called COUNTRIES. The following example demonstrates how to enforce a Real Application Security policy that virtually partitions the customer and sales data along their regional boundary defined in the COUNTRIES table for read access of the CUSTOMERS and



SALES tables. In addition, there is a requirement to mask out data on the columns CUST\_INCOME\_LEVEL and CUST\_CREDIT\_LIMIT to users, except for those users who need full table access for business analysis, such as the business analyst.



All administrative commands in this example can be performed by a database user, such as the SYSTEM account who has the DBA roles in the database, because the DBA role has been granted appropriate privilege for Real Application Security administrative tasks. In addition, because security classes, ACLs, and data security policies are schema qualified objects, you must explicitly use the intended schema name when these objects are specified in the APIs, so they will not be resolved to objects under the database session default schema of SYSTEM.

The descriptions for the three tables, which are all in the same schema (SH), are as follows:

SH.CUSTOMERS in the master table. Name	Null?	Туре
CUST ID	NOT NULL	NUMBER
CUST FIRST NAME	NOT NULL	VARCHAR2(20)
CUST LAST NAME		VARCHAR2 (40)
CUST GENDER		CHAR(1)
CUST YEAR OF BIRTH		NUMBER (4)
CUST MARITAL STATUS		VARCHAR2(20)
CUST STREET ADDRESS	NOT NULL	VARCHAR2(40)
CUST POSTAL CODE	NOT NULL	VARCHAR2(10)
CUST CITY	NOT NULL	VARCHAR2(30)
CUST STATE PROVINCE		VARCHAR2(40)
COUNTRY ID	NOT NULL	CHAR(2)
CUST MAIN PHONE NUMBER		VARCHAR2 (25)
CUST_INCOME_LEVEL		VARCHAR2(30)
CUST_CREDIT_LIMIT		NUMBER
CUST_EMAIL		VARCHAR2(30)
SH.SALES is a detail table. Name	Null?	Type
Name		
Name  PROD_ID	NOT NULL	NUMBER(6)
Name  PROD_ID CUST_ID	NOT NULL	NUMBER (6) NUMBER
Name PROD_ID CUST_ID TIME_ID	NOT NULL NOT NULL NOT NULL	NUMBER (6) NUMBER DATE
Name PROD_ID CUST_ID TIME_ID CHANNEL_ID	NOT NULL NOT NULL NOT NULL	NUMBER (6) NUMBER DATE CHAR (1)
Name PROD_ID CUST_ID TIME_ID	NOT NULL NOT NULL NOT NULL NOT NULL	NUMBER (6) NUMBER DATE
Name PROD_ID CUST_ID TIME_ID CHANNEL_ID PROMO_ID	NOT NULL NOT NULL NOT NULL NOT NULL NOT NULL NOT NULL	NUMBER (6) NUMBER DATE CHAR (1) NUMBER (6)
Name PROD_ID CUST_ID TIME_ID CHANNEL_ID PROMO_ID QUANTITY_SOLD	NOT NULL NOT NULL NOT NULL NOT NULL NOT NULL NOT NULL	NUMBER (6) NUMBER DATE CHAR (1) NUMBER (6) NUMBER (3) NUMBER (3)
Name PROD_ID CUST_ID TIME_ID CHANNEL_ID PROMO_ID QUANTITY_SOLD AMOUNT_SOLD SH.COUNTRIES is a detail table.	NOT NULL	NUMBER (6) NUMBER DATE CHAR (1) NUMBER (6) NUMBER (3) NUMBER (3)
Name PROD_ID CUST_ID TIME_ID CHANNEL_ID PROMO_ID QUANTITY_SOLD AMOUNT_SOLD SH.COUNTRIES is a detail table.	NOT NULL	NUMBER (6) NUMBER DATE CHAR (1) NUMBER (6) NUMBER (3) NUMBER (10,2)  Type CHAR (2)
Name PROD_ID CUST_ID TIME_ID CHANNEL_ID PROMO_ID QUANTITY_SOLD AMOUNT_SOLD SH.COUNTRIES is a detail table. Name	NOT NULL	NUMBER (6) NUMBER DATE CHAR (1) NUMBER (6) NUMBER (3) NUMBER (10,2)  Type CHAR (2) VARCHAR2 (40)
Name PROD_ID CUST_ID TIME_ID CHANNEL_ID PROMO_ID QUANTITY_SOLD AMOUNT_SOLD  SH.COUNTRIES is a detail table. Name COUNTRY_ID COUNTRY_NAME COUNTRY_SUBREGION	NOT NULL	NUMBER (6) NUMBER DATE CHAR (1) NUMBER (6) NUMBER (3) NUMBER (10,2)  Type  CHAR (2) VARCHAR2 (40) VARCHAR2 (30)
Name PROD_ID CUST_ID TIME_ID CHANNEL_ID PROMO_ID QUANTITY_SOLD AMOUNT_SOLD  SH.COUNTRIES is a detail table. Name	NOT NULL	NUMBER (6) NUMBER DATE CHAR (1) NUMBER (6) NUMBER (3) NUMBER (10,2)  Type CHAR (2) VARCHAR2 (40)



Figure 5-2 shows an overview of the completed Real Application Security data security policies created and applied to the master-detail related tables (CUSTOMERS - SALES - COUNTRIES) that are described as an overview in the following steps and in more detail in the steps that follow this figure.

- Create the principals, an application role and an application user, for each of four geographic regions: Europe, Americas, Asia, and Africa, in addition to a business analyst role and an associated application user.
- Create the VIEW\_SENSITIVE\_INFO privilege and create the SH.CUST\_SEC\_CLASS in which to scope the privilege.
- 3. Grant the VIEW SENSITIVE INFO privilege to the business analyst role.
- 4. Define a data realm constraint with a rule that parameterizes regions in order for the system to recognize the string &REGION, which will later be used in a policy.
- 5. Create a column constraint to secure the two columns, CUST\_INCOME\_LEVEL and CUST CREDIT LEVEL using the VIEW SENSITIVE INFO privilege.
- 6. Create the data security policy SH.CUSTOMER\_DS specifying the data realm constraint and the column constraint that was previously created.
- Register the name and data type of the parameter in the rule for the SH.CUSTOMER\_DS
  data security policy.
- 8. Create the ACLs for each region to authorize read access to the respective roles needing read access. For example for the Europe region, you grant SELECT privilege to the Europe\_sales role and grant SELECT and VIEW\_SENSITIVE\_INFO privileges to the Business Analyst role.
- 9. Associate each ACL in each region with the rows that satisfy the rule where the value of the parameter REGION is equal to region name, for example, Europe. You do this for each of the four regions, and then add this ACL to the SH.CUSTOMER DS data security policy.
- 10. Create the data realm constraint for the master-detail tables, so users can access a record in the SALES detail table only if a user is authorized to access its parent row in the CUSTOMERS master table.
- 11. Create the SH. SALES DS data security policy to enforce this data realm constraint.

In Figure 5-2, the master-detail tables also show the primary key (PK) fields and foreign key (FK) fields and a number of additional fields that are used in creating the data realm constraints and column constraints. Using these PK and FK relationships, the same data security policies that apply to the master table also apply to the detail tables. In this particular case, for example, all ACLs granting SELECT privilege to the CUSTOMERS master table and enforced by the SH.CUSTOMER DS data security policy, also applies to the SALES detail table.



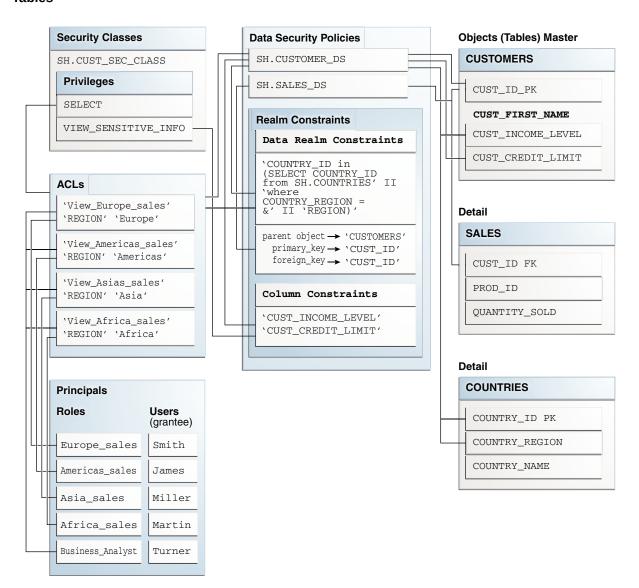


Figure 5-2 Real Application Security Data Security Policy Created on Master-Detail Related Tables

To create a Real Application Security policy for these master-detail tables, follow these steps:

1. Create the roles and users needed for each country, (role Europe\_sales, user SMITH), (role Americas\_sales, user JAMES), (role Asia\_sales, user MILLER), (role Africa\_sales, user MARTIN), and (role Business\_Analyst, user TURNER), who is the only user who will have full table access.

```
segin
sys.xs_principal.create_role(name => 'Europe_sales', enabled => TRUE);
sys.xs_principal.create_role(name => 'Americas_sales', enabled => TRUE);
sys.xs_principal.create_role(name => 'Asia_sales', enabled => TRUE);
sys.xs_principal.create_role(name => 'Africa_sales', enabled => TRUE);
sys.xs_principal.create_role(name => 'Business_Analyst', enabled => TRUE);
sys.xs_principal.create_user(name => 'SMITH', schema => 'SH');
```



```
sys.dbms xs principals.set password(username => 'SMITH',
                                       password => 'password',
                                       type => XS PRINCIPAL.XS SHA512);
   sys.xs principal.grant roles(grantee => 'SMITH', role => 'Europe sales');
   sys.xs principal.create user(name =>' JAMES', schema => 'SH');
   sys.dbms xs principals.set password(username => 'JAMES',
                                       password => 'password',
                                       type => XS PRINCIPAL.XS SHA512);
   sys.xs principal.grant roles(grantee => 'JAMES', role => 'Americas sales');
   sys.xs principal.create user(name => 'MILLER', schema => 'SH');
   sys.dbms xs principals.set password(username => 'MILLER',
                                       password => 'password',
                                       type => XS PRINCIPAL.XS SHA512);
   sys.xs principal.grant roles(grantee => 'MILLER', role => 'Asia sales');
   sys.xs_principal.create_user(name => 'MARTIN', schema => 'SH');
   sys.dbms xs principals.set password(username => 'MARTIN',
                                       password => 'password',
                                       type => XS PRINCIPAL.XS SHA512);
   sys.xs principal.grant roles(grantee => 'MARTIN', role => 'Africa sales');
   sys.xs principal.create user(name => 'TURNER', schema=> 'SH');
   sys.dbms xs principals.set password(username => 'TURNER',
                                       password => 'password',
                                       type => XS PRINCIPAL.XS SHA512);
   sys.xs principal.grant roles(grantee => 'TURNER', role => 'Business Analyst');
END;
```

2. Define the SH.CUST\_SEC\_CLASS security class for the privilege, VIEW\_SENSITIVE\_INFO to protect the sensitive columns.

The row level privileges to access data security protected objects for query and DML are predefined in the Security Class DML under the SYS schema.

3. Define the data realm constraint with a rule that parameterizes regions, then define the column constraint and specify the name of the two columns, <code>CUST\_INCOME\_LEVEL</code> and <code>CUST\_CREDIT\_LIMIT</code>, to be secured by the <code>VIEW\_SENSITIVE\_INFO</code> privilege. Then, create a <code>SH.CUSTOMER\_DS</code> data security policy and register the name and data type of the parameter in the rule.

The security policy requires that regional customers and sales data be partitioned with different ACLs. One way to achieve this is to define as many data realms as regions and do this for both tables. However, in this example, another way is shown. That is, to parameterize the region in a data realm with a single rule and use the master-detail relationship to simplify the administrative tasks.



So, instead of creating many constraints for the policy, it is more efficient to create only one constraint with the following rule that parameterizes the region:

```
COUNTRY_ID in (select COUNTRY ID from SH.COUNTRIES where COUNTRY REGION = &REGION)
```

In order for the system to recognize that the string &REGION in the rule is indeed a parameter, you must invoke the <code>xs\_data\_security.create\_acl\_parameter</code> procedure to register the parameter name after the policy is created. In addition, you must specify the data type of the parameter value. Since regions are stored as character string data, the <code>XS\_ACL.TYPE\_VARCHAR</code> macro is used for this example. Another supported data type is <code>XS\_ACL.TYPE\_NUMBER</code> for numbers.

```
DECLARE
 rows secs XS$REALM CONSTRAINT LIST;
 cols secs XS$COLUMN CONSTRAINT LIST;
-- Define the realm constraint with a rule that parameterizes regions.
 rows secs := xs$REALM CONSTRAINT LIST(
         XS$REALM_CONSTRAINT TYPE(
            realm => 'COUNTRY ID in (select COUNTRY ID from SH.COUNTRIES ' ||
                     'where COUNTRY REGION = &' || 'REGION)'));
-- Define the column constraint to secure CUST INCOME LEVEL and
-- CUST CREDIT LIMIT columns by using the VIEW SENSITIVE INFO privilege.
 cols secs := XS$COLUMN CONSTRAINT LIST(
        XS$COLUMN CONSTRAINT TYPE(
         column list => XS$LIST('CUST INCOME LEVEL', 'CUST CREDIT LIMIT'),
         privilege => 'VIEW SENSITIVE INFO'));
-- Create the data security policy.
 sys.xs data security.create policy(
         name => 'SH.CUSTOMER DS',
         realm constraint list => rows secs,
         column constraint list => cols secs,
         description => 'Policy to protect sh.customers table');
-- Register the name and data type of the parameter in the rule.
 sys.xs data security.create acl parameter(
          policy => 'SH.CUSTOMER DS',
          parameter => 'REGION',
          param type => XS ACL.TYPE VARCHAR);
END;
```

4. Create ACLs to authorize read access for each region. For the Europe region, grant SELECT to the Europe\_sales role. In addition, SELECT and VIEW\_SENSITIVE\_INFO privileges are granted to the Business\_Analyst role so that the grantee of the role has full table access and is able to see data in the columns of CUST\_INCOME\_LEVEL and CUST\_CREDIT\_LIMIT as well.



Create ACLs to authorize read access for the other three regions, Americas, Asia, and Africa.

```
DECLARE
 ace_list XS$ACE_LIST;
BEGIN
 ace list := XS$ACE LIST(
              XS$ACE TYPE(privilege list => XS$NAME LIST('SELECT'),
                          granted => true,
                          principal name => 'Americas sales'),
              XS$ACE TYPE(privilege list =>
                            XS$NAME LIST('SELECT', 'VIEW SENSITIVE INFO'),
                          granted => true,
                          principal name => 'Business Analyst'));
 sys.xs acl.create acl(name => 'View Americas sales',
                ace list => ace list,
                sec class => 'SH.CUST SEC CLASS',
                description => 'Authorize read access for the Americas region');
 sys.xs acl.add acl parameter(acl => 'View Americas sales',
                           policy => 'SH.CUSTOMER DS',
                           parameter => 'REGION',
                           value => 'Americas');
END;
DECLARE
 ace list XS$ACE LIST;
BEGIN
 ace list := XS$ACE LIST(
              XS$ACE TYPE(privilege list => XS$NAME LIST('SELECT'),
                          granted => true,
                          principal name => 'Asia sales'),
              XS$ACE TYPE(privilege list =>
                            XS$NAME LIST('SELECT', 'VIEW SENSITIVE INFO'),
                          granted => true,
                          principal_name => 'Business_Analyst'));
  sys.xs acl.create acl(name => 'View Asia sales',
                ace list => ace list,
                sec class => 'SH.CUST SEC CLASS',
                description => 'Authorize read access for the Asia region');
  sys.xs acl.add acl parameter(acl => 'View Asia sales',
                           policy => 'SH.CUSTOMER DS',
```



```
parameter => 'REGION',
                           value => 'Asia');
END;
DECLARE
 ace list XS$ACE LIST;
BEGIN
 ace list := XS$ACE LIST(
              XS$ACE TYPE(privilege list => XS$NAME LIST('SELECT'),
                          granted => true,
                          principal name => 'Africa sales'),
              XS$ACE TYPE(privilege list =>
                            XS$NAME LIST('SELECT', 'VIEW SENSITIVE INFO'),
                          granted => true,
                          principal name => 'Business Analyst'));
 sys.xs acl.create acl(name => 'View Africa sales',
                ace list => ace list,
                sec_class => 'SH.CUST_SEC CLASS',
                description => 'Authorize read access for the Africa
region');
 sys.xs acl.add acl parameter(acl => 'View Africa sales',
                           policy => 'SH.CUSTOMER DS',
                           parameter => 'REGION',
                           value => 'Africa');
END;
```

Apply the SH.CUSTOMER\_DS policy created in Step 3 to protect read access to the CUSTOMERS table.

```
BEGIN
    sys.xs_data_security.apply_object_policy(
        policy => 'SH.CUSTOMER_DS',
        schema => 'SH',
        object => 'CUSTOMERS',
        statement_types => 'SELECT',
        owner_bypass => true);
END;
```

7. Create the data realm master-detail constraint to protect the SALES table. This master-detail constraint utilizes the same regional partitioning policy as previously described in Steps 3 through 6. This means that a user can access a record in the SALES detail table only if that user is authorized to access its parent row in the CUSTOMERS master table.



8. Grant object level SELECT privilege to PUBLIC for users to perform a query.

```
GRANT SELECT ON sh.customers TO PUBLIC;
GRANT SELECT ON sh.countries TO PUBLIC;
GRANT SELECT ON sh.sales TO PUBLIC;
```

9. Connect as user MARTIN and perform a query to display user MARTIN's sales data for the Africa region and to show the masking of the sensitive sales information for the CUST\_INCOME\_LEVEL and CUST\_CREDIT\_LIMIT columns.

```
CONNECT MARTIN/welcome

SELECT c.COUNTRY_NAME, c.COUNTRY_ID, ct.CUST_FIRST_NAME, PROD_ID, QUANTITY_SOLD FROM sh.customers ct, sh.sales s, sh.countries c

WHERE ct.CUST_ID = s.CUST_ID AND ct.COUNTRY ID = c.COUNTRY ID;
```

COUNTRY_NAME	CO	CUST_FIRST_NAME	PROD_ID	QUANTITY_SOLD
South Africa	ZA	Forrest	8050	2
South Africa	ZA	Mitch	17505	11
South Africa	ZA	Murry	32785	7
South Africa	ZA	Heath	3585	12

# 5.8 About Managing Application Privileges for Data Security Policies

This section includes the following topics:

- About Bypassing the Security Checks of a Real Application Security Policy
- Using the SQL\*Plus SET SECUREDCOL Command

# 5.8.1 About Bypassing the Security Checks of a Real Application Security Policy

The following database users can bypass the security checks of a Real Application Security Policy:

- User sys
- Database users who have the EXEMPT ACCESS POLICY system privilege
- The owner of the object to which the policy is applied.

If the data security policy is applied to an object with the owner bypass specification, the owner of the object may bypass such policy. By default, owner bypass is not allowed.

The object owner also can create another view on the same table and assign this view a different Real Application Security policy.

## 5.8.2 Using the SQL\*Plus SET SECUREDCOL Command

The SQL\*Plus SET SECUREDCOL command enables you to customize how secure column values are displayed in SQL\*Plus output for users without permission to view a column and for columns with unknown security. You can choose either the default text or specify the text that is displayed. The default is OFF.

When column level security is enabled, and SET SECUREDCOL is set ON, output from SQL\*Plus for secured columns or columns of unknown security level is replaced with either your customized text or the default indicators. This only applies to scalar data types. Complex object data output is not affected.

#### **Syntax**

SET SECUREDCOL {OFF | ON} [UNAUTH[ORIZED] text][UNK[NOWN] text]

#### **Parameters**

Parameter	Description
ON	Displays the default indicator asterisks (****) in place of column values for users without authorization to view the column, and displays question marks (?????) in place of column values where the security level is unknown for the column (when the specific privileges applied to the column are not known). The indicators "*" and "?" are filled to the defined column length or the column length defined by a current COLUMN command.
	By default this command will be OFF.
OFF	Displays null values in place of column values for application users without authorization to view the column, and in place of column values where the security level is unknown for the column.
UNAUTH[ORIZED]	Text enables you to specify the text to be displayed in a secured column for application users without authorization to view the column. This text appears instead of the default *****.
	You can specify any alphanumeric text up to the column length or a maximum of 30 characters. Longer text is truncated. Text containing spaces must be quoted.
UNK[NOWN]	Text enables you to specify the text to be displayed in a column of unknown security level (when the specific privileges applied to the column are not known). This text appears instead of the default ???????.
	You can specify any alphanumeric text up to the column length or a maximum of 30 characters. Longer text is truncated. Text containing spaces must be quoted.

#### **Example 1**

SET SECUREDCOL ON SELECT empno, ename, sal FROM emp ORDER BY deptno;

#### The output of the example will be as follows:

EMPNO ENAME	DEPTNO SAL
7539 KING	10 ******
7369 SMITH	20 800
7566 JONES	20 2975
7788 SCOTT	20 3000



```
7521 WARD 30 *******
7499 ALLEN 30 *******
```

6 rows selected.

#### **Example 2**

```
SET SECUREDCOL ON UNAUTH notallowed SELECT empno, ename, sal FROM emp ORDER BY deptno;
```

#### The output of the example will be as follows:

EMPNO ENAME	DEPTNO SAL
7539 KING	10 notallowed
7369 SMITH	20 800
7566 JONES	20 2975
7788 SCOTT	20 3000
7521 WARD	30 notallowed
7499 ALLEN	30 notallowed

6 rows selected.

## 5.9 Using BEQUEATH CURRENT USER Views

Traditionally, views in Oracle Database use definer's rights. This means that if you invoke an identity or privilege-sensitive SQL function or an invoker's rights PL/SQL or Java function, then current schema, and current user, are set to the view owner and currently enabled roles is set to the view owner plus PUBLIC within the functions's execution.

If you need background information on invoker's rights and definer's rights, see *Oracle Database PL/SQL Language Reference*.



Certain built-in SQL functions, such as SYS\_CONTEXT() and USERENV() are exceptions to the preceding rule. These functions always use the current application user's environment, even when called from definer's rights views.

Oracle Database 12c Release 1 (12.1) and later enables you to create views with the BEQUEATH clause, which lets you configure this behavior. The BEQUEATH clause determines whether identity or privilege-sensitive SQL functions, invoker's rights PL/SQL program units, and Java functions referenced in the view inherit the current schema, current user, and currently enabled roles from the querying user's environment. This is especially useful for Real Application Security applications, which often need to run code in the invoking application user's environment.

Using BEQUEATH CURRENT\_USER in the view definition creates a view that allows privilege-sensitive, and invoker's rights functions referenced in the view to inherit current schema, current user, and currently enabled roles from the querying user's environment. See *Oracle Database SQL Language Reference* for the syntax of the CREATE OR REPLACE VIEW statement.



Example 5-7 illustrates how a BEQUEATH CURRENT\_USER view enables invoker right's program units to run in the invoking application user's environment. When USER2 selects from USER1's view, the invoker's rights function is invoked in USER2's environment.

Using BEQUEATH DEFINER in the view definition creates a view that causes privilegesensitive, and invoker's rights functions referenced in the view to inherit current schema, current user, and currently enabled roles from the view definer's environment. If no BEQUEATH clause is specified, then BEQUEATH DEFINER is assumed.

If a BEQUEATH\_DEFINER view contains a reference to a BEQUEATH CURRENT\_USER view, then invoker's rights functions in the referenced view would use the parent view owner's rights.

Example 5-8 illustrates how a BEQUEATH DEFINER view defines a boundary for nested invoker right's program units to run in the view owner's environment. When USER2 selects from USER1's view, the view's invoker's rights function is invoked in USER1's environment.



Oracle Database Security Guide for the use of invoker's rights and definer's rights in VPD and FGA policies

#### Example 5-7 How a BEQUEATH CURRENT USER View Works

```
SOL> CONNECT USER1/USER1
Connected.
SOL>
SQL> -- You first create an invoker's rights function to determine who the
current SQL> -- user really is.
SQL> CREATE OR REPLACE FUNCTION CALLED AS USER RETURN VARCHAR2 AUTHID
CURRENT USER IS
2 BEGIN
3 RETURN SYS CONTEXT ('USERENV', 'CURRENT USER');
4 END;
5 /
Function created.
SQL> -- Note that you do not need to grant EXECUTE to called as user, because
SQL> -- BEQUEATH CURRENT USER views do name resolution and privilege checking on
SQL> -- the references present in the view body using definer's rights.
SQL> CREATE OR REPLACE VIEW BEQUEATH INVOKER VIEW BEQUEATH CURRENT USER AS
2 SELECT CALLED AS USER FROM DUAL;
View created.
SQL> GRANT SELECT ON BEQUEATH INVOKER VIEW TO PUBLIC;
Grant succeeded.
SOL> CONNECT USER2/USER2
Connected.
SQL> SELECT * FROM USER1.BEQUEATH INVOKER VIEW;
CALLED AS USER
```



\_\_\_\_\_USER2

#### **Example 5-8 How a BEQUEATH DEFINER View Works**

```
SOL> CONNECT USER1/USER1
Connected.
SQL>
SQL> -- You first create an invoker's rights function to determine who the current
SQL> -- user really is.
SQL> CREATE OR REPLACE FUNCTION CALLED AS USER RETURN VARCHAR2 AUTHID CURRENT USER IS
3 RETURN SYS CONTEXT ('USERENV', 'CURRENT USER');
4 END:
5 /
Function created.
SQL> -- Note that you do not need to grant EXECUTE to called as user, because even
SQL> -- BEQUEATH CURRENT USER views do name resolution and privilege checking on
SQL> -- the references present in the view body using definer's rights.
SQL> CREATE OR REPLACE VIEW BEQUEATH DEFINER VIEW BEQUEATH DEFINER AS
2 SELECT CALLED AS USER FROM DUAL;
View created.
SQL> GRANT SELECT ON BEQUEATH_DEFINER_VIEW TO PUBLIC;
Grant succeeded.
SQL> CONNECT USER2/USER2
Connected.
SQL> SELECT * FROM USER1.BEQUEATH DEFINER VIEW;
CALLED AS USER
```

This section includes the following topic: Using SQL Functions to Determine the Invoking Application User.

## 5.9.1 Using SQL Functions to Determine the Invoking Application User

SQL functions, such as  $SYS\_CONTEXT()$  and USERENV(), and  $XS\_SYS\_CONTEXT()$ , always return the current application user's environment, even when called from definer's rights views. Sometimes, applications need to determine the invoking application user based on the security context (BEQUEATH property) of views referenced in the statement.

The following new functions introduced in Oracle Database 12c Release 1 (12.1) enable you to figure out the invoking application user taking into account the BEQUEATH property of views referenced in the statement:

- ORA\_INVOKING\_USER: Use this function to return the name of the database user whose context is currently used. If the function is invoked from within a definer's rights boundary, then the name of the database object owner is returned. If the invoking user is a Real Application Security application user, then the constant XS\$USER is returned.
- ORA\_INVOKING\_USERID: Use this function to return the identifier (ID) of the database user whose context is currently used. If the function is invoked from within a definer's rights boundary, then the ID of the database object owner is returned.



If the invoking user is a Real Application Security application user, then the function returns an identifier common to all Real Application Security application users, but distinct from the identifier for any database user.

• ORA\_INVOKING\_XS\_USER: Use this function to return the name of the Real Application Security application user whose context is currently used.

If the invoking user is a database user, then the value  ${\tt NULL}$  is returned.

• ORA\_INVOKING\_XS\_USER\_GUID: Use this function to return the identifier (ID) of the Real Application Security application user whose context is currently used.

If the invoking user is a database user, then the value NULL is returned.

The following example shows a database user <code>user1</code> querying <code>ora\_invoking\_user</code> and <code>ora\_invoking\_xs\_user</code>. <code>ora\_invoking\_xs\_user</code> returns <code>null</code>, as the user is not a Real Application security application user.

#### See Also:

- Oracle Database SQL Language Reference for detailed information on the preceding SQL functions and other functions like SYS CONTEXT
- "XS SYS CONTEXT Function"

# 5.10 Real Application Security: Putting It All Together

This section puts all the Real Application Security concepts together in order to define a basic data security policy. It builds upon the HR scenario example introduced in "Scenario: Security Human Resources (HR) Demonstration of Employee Information".

The section includes the following topic that discusses each implementation task described in the scenario with the help of an example.

This section includes the following topics:

- Basic HR Scenario: Implementation Tasks
- Running the Security HR Demo

### 5.10.1 Basic HR Scenario: Implementation Tasks

The following implementation tasks are discussed:



- Connecting as User SYS to Create Real Application Security Users and Roles
- Creating Roles and Application Users
- Creating the Security Class and ACLS
- Creating the Data Security Policy
- Validating the Real Application Security Objects
- Disabling a Data Security Policy for a Table

# 5.10.1.1 Connecting as User SYS to Create Real Application Security Users and Roles

To create Real Application Security users and roles, you need only to connect as user SYS.

#### Example 5-9 Connecting as User SYS

SQL> connect sys/&passwd as sysdba Connected.

### 5.10.1.2 Creating Roles and Application Users

#### **Creating the Database Role**

Create the database role DB\_EMP and grant this role the necessary table privileges. This role is used to grant the required object privileges to application users.

#### **Creating the Application Roles**

#### Grant the DB\_EMP Database Role to the Application Roles

Grant the  $DB\_EMP$  database role to the three application roles, so they each have the required object privilege to access the table.

#### **Create the Application Users**

Create application user <code>DAUSTIN</code> (in the IT department) and grant this user application roles EMPPLOYEE and II ENGINEER.

In this example:



To make logins easier, you can create the name in upper case. That way, the user can omit the quotation marks when logging in or connecting to SQL\*Plus. For example:

sqlplus DAUSTIN



#### See Also:

"Creating a Simple Application User Account" for information about how case sensitivity affects database logins for application users

Create application user SMAVRIS (in the HR department) and grant this user application roles EMPLOYEE and HR REPRESENTATIVE.

# Grant the HR User the Policy Administration Privilege ADMIN ANY SEC POLICY

Grant the HR user the ADMIN\_ANY\_SEC\_POLICY privilege.

#### Example 5-10 Creating the DB\_EMP Role

```
SQL> create role db_emp;
Role created.
SQL> grant select, insert, update, delete on hr.employees to db_emp;
Grant succeeded.
```

# Example 5-11 Creating the Application Role EMPLOYEE for Common Employees

```
SQL> exec sys.xs_principal.create_role(name => 'employee', enabled => true);
PL/SQL procedure successfully completed.
```

# Example 5-12 Creating the Application Role IT\_ENGINEER for the IT Department

```
SQL> exec sys.xs_principal.create_role(name => 'it_engineer', enabled => true);
PL/SQL procedure successfully completed.
```

# Example 5-13 Creating the Application Role HR\_REPRESENTATIVE for the HR Department

```
SQL> exec sys.xs_principal.create_role(name => 'hr_representative', enabled =>
true);
PL/SQL procedure successfully completed.
```

#### Example 5-14 Granting DB\_EMP Database Role to Each Application Role

```
SQL> grant db_emp to employee;
Grant succeeded.
SQL> grant db_emp to it_engineer;
Grant succeeded.
SQL> grant db_emp to hr_representative;
Grant succeeded.
```



#### **Example 5-15 Creating Application User DAUSTIN**

```
SQL> exec sys.xs_principal.create_user(name => 'daustin', schema => 'hr');
PL/SQL procedure successfully completed.

SQL> exec sys.xs_principal.set_password('daustin', 'password');
PL/SQL procedure successfully completed.

SQL> exec sys.xs_principal.grant_roles('daustin', 'XSCONNECT');
PL/SQL procedure successfully completed.

SQL> exec sys.xs_principal.grant_roles('daustin', 'employee');
PL/SQL procedure successfully completed.

SQL> exec sys.xs_principal.grant_roles('daustin', 'it_engineer');
PL/SQL procedure successfully completed.
```

#### Example 5-16 Creating Application User SMAVRIS

```
SQL> exec sys.xs_principal.create_user(name => 'smavris', schema => 'hr');
PL/SQL procedure successfully completed.
SQL> exec sys.xs_principal.set_password('smavris', 'password');
PL/SQL procedure successfully completed.
SQL> exec sys.xs_principal.grant_roles('daustin', 'XSCONNECT');
PL/SQL procedure successfully completed.
SQL> exec sys.xs_principal.grant_roles('smavris', 'employee');
PL/SQL procedure successfully completed.
SQL> exec sys.xs_principal.grant_roles('smavris', 'hr_representative');
PL/SQL procedure successfully completed.
```

# Example 5-17 Granting the HR User the Policy Administration Privilege ADMIN\_ANY\_SEC\_POLICY

```
SQL> exec sys.xs_admin_util.grant_system_privilege('ADMIN_ANY_SEC_POLICY','HR');
PL/SQL procedure successfully completed.
```

## 5.10.1.3 Creating the Security Class and ACLS

#### **Creating the Security Class**

Create a security class  $\mbox{HR\_PRIVILEGES}$  based on the predefined DML security class.  $\mbox{HR\_PRIVILEGES}$  has a new privilege  $\mbox{VIEW\_SALARY}$ , which controls access to the  $\mbox{SALARY}$  column.

#### **Creating the ACIs**

Create three ACLs, EMP\_ACL, IT\_ACL, and HR\_ACL to grant privileges for the data security policy to be defined later.

In this example:

- Lines 11 through 13: Creates the EMP\_ACL and grants EMPLOYEE the SELECT and VIEW SALARY privileges.
- Lines 21 through 23: Creates the IT\_ACL and grants IT\_ENGINEER the SELECT privileges.
- Lines 30 through 33: Creates the HR\_ACL and grants HR\_REPRESENTATIVE the
   SELECT, INSERT, UPDATE, and DELETE database privileges to view and update all
   employee's records, and granting the VIEW\_SALARY application privilege to view the
   SALARY column.

#### **Example 5-18 Creating the HRPRIVS Security Class**

PL/SQL procedure successfully completed.

#### Example 5-19 Creating ACLs: EMP\_ACL, IT\_ACL, and HR\_ACL

```
SQL> declare
 2
     aces xs$ace list := xs$ace list();
 3 begin
     aces.extend(1);
     -- EMP ACL: This ACL grants EMPLOYEE role the privileges to view an
employee's
                 own record including SALARY column.
    aces(1) := xs$ace type(privilege list =>
xs$name list('select','view salary'),
                            principal name => 'employee');
10
                                    => 'emp_acl',
11
    sys.xs_acl.create_acl(name
12
                        ace list => aces,
13
                        sec class => 'hr privileges');
14
15
      -- IT ACL: This ACL grants IT ENGINEER the privilege to view the employee
16
                  records in IT department, but it does not grant the
VIEW SALARY
17
                  privilege that is required for access to SALARY column.
18
      aces(1) := xs$ace type(privilege list => xs$name list('select'),
 19
                             principal name => 'it engineer');
 20
                                    => 'it_acl',
 21
     sys.xs_acl.create_acl(name
 22
                        ace list => aces,
 23
                        sec_class => 'hr_privileges');
 24
 25
      -- HR ACL: This ACL grants HR REPRESENTATIVE the privileges to view and
```



```
update all
26
                  employees' records including SALARY column.
      aces(1):= xs$ace type(privilege list => xs$name list('select', 'insert',
                                                 'update', 'delete', 'view salary'),
 29
                            principal name => 'hr representative');
 30
                                     => 'hr_acl',
 31
     sys.xs_acl.create_acl(name
32
                        ace list => aces,
 33
                        sec class => 'hr privileges');
 34 end;
35 /
```

PL/SQL procedure successfully completed.

### 5.10.1.4 Creating the Data Security Policy

Create the data security policy for the EMPLOYEES table. The policy defines three data realm constraints and a column constraint that protects the SALARY column.

In this example:

- Lines 7 through 23: Defines the three data realm constraints.
- Lines 27 through 30: Defines the column constraint requiring the VIEW\_SALARY
  application privilege to view the SALARY column.
- Lines 32 through 35: Creates the EMPLOYEES\_DS data security policy encompassing the three data realm constraints and the column constraint.

#### **Applying the Data Security Policy to the Table**

Apply the data security policy to the EMPLOYEES table.

#### Example 5-20 Creating the EMPLOYEES\_DS Data Security Policy

```
SOL> declare
 2
     realms xs$realm_constraint_list := xs$realm_constraint_list();
              xs$column constraint list := xs$column constraint list();
 3
 4 begin
     realms.extend(3);
 7
     -- Realm #1: Only the employee's own record.
                 EMPLOYEE role can view the realm including SALARY column.
    realms(1) := xs$realm constraint type(
10
      realm => 'email = xs sys context(''xs$session'',''username'')',
      acl list => xs$name list('emp acl'));
      -- Realm #2: The records in the IT department.
13
                IT ENGINEER role can view the realm excluding SALARY column.
14
     realms(2) := xs$realm constraint type(
15
      realm => 'department id = 60',
16
17
       acl_list => xs$name_list('it_acl'));
18
19
      -- Realm #3: All the records.
20
                  HR REPRESENTATIVE role can view and update the realm including
SALARY column.
21 realms(3) := xs$realm constraint type(
22
      realm => '1 = 1',
23
      acl_list => xs$name_list('hr_acl'));
24
     -- Column constraint protects SALARY column by requiring VIEW SALARY
```

```
26 -- privilege.
27 cols.extend(1);
28 cols(1) := xs$column constraint type(
     column list => xs$list('salary'),
     privilege => 'view salary');
30
31
32
    sys.xs_data_security.create_policy(
                           => 'employees_ds',
33
     name
34
      realm constraint list => realms,
35
      column constraint list => cols);
36 end;
37 /
```

PL/SQL procedure successfully completed.

# Example 5-21 Applying the EMPLOYEES\_DS Security Policy to the EMPLOYEES Table

```
SQL> begin
2    sys.xs_data_security.apply_object_policy(
3     policy => 'employees_ds',
4     schema => 'hr',
5     object =>'employees');
6    end;
7    /
```

PL/SQL procedure successfully completed.

### 5.10.1.5 Validating the Real Application Security Objects

After you create these Real Application Security objects, validate them to ensure they are all properly configured.

#### **Example 5-22 Validating the Real Application Security Objects**

```
SQL> set serveroutput on;
SQL> begin
2    if (xs_diag.validate_workspace()) then
3        dbms_output.put_line('All configurations are correct.');
4    else
5        dbms_output.put_line('Some configurations are incorrect.');
6    end if;
7    end;
8    /
All configurations are correct.

PL/SQL procedure successfully completed.

SQL> -- XS$VALIDATION_TABLE contains validation errors if any.
SQL> -- Expect no rows selected.
SQL> select * from xs$validation_table order by 1, 2, 3, 4;
no rows selected
```

### 5.10.1.6 Disabling a Data Security Policy for a Table

Example 5-23 shows the complementary operation of disabling data security for table HR.EMPLOYEES.

#### **Example 5-23** Disabling a Data Security Policy for a Table

```
BEGIN
   SYS.XS_DATA_SECURITY.DISABLE_OBJECT_POLICY(policy => 'EMPLOYEES_DS', schema => 'HR',
object => 'EMPLOYEES');
END;
//
```

## 5.10.2 Running the Security HR Demo

The Security HR Demo is run in two ways:

- Using direct logon first as application user DAUSTIN and later as application user SMAVRIS.
  In each case, each user performs queries on the HR.EMPLOYEES table to demonstrate what each can access or cannot access to view employee records and the SALARY column. See "Running the Security HR Demo Using Direct Logon" for a description of this demonstration.
- Attached to a Real Application Security session
   In this demonstration, the Real Application Security Administrator creates a Real Application Security session for an application user to attach to. See "Running the Security HR Demo Attached to a Real Application Security Session" for a description of this demonstration.

# 5.11 About Schema Level Real Application Security Policy Administration

Describes introduction of schema level privileges for Real Application Security policy administration across different applications within the same schema.

Beginning with Oracle Database 12c Release 2 (12.2), Real Application Security introduces schema level privileges, which allows a policy administrator to create, update, and apply a policy in only the granted schema and administer policy enforcement within one application, thereby achieving separate management and enforcement of a policy across different applications within the same schema. This level of policy administration is essential in a Cloud computing scenario where each application may be running under one or more schemas. It then becomes highly desirable for a policy administrator to have the ability to manage and apply data security policies for each individual application in that environment.

#### **Achieving Schema Level Data Security Policy Administration**

To achieve schema level data security policy administration, the following new and changed features were introduced:

The GRANT\_SYSTEM\_PRIVILEGE and REVOKE\_SYSTEM\_PRIVILEGE procedures were extended
with the addition of the schema parameter to allow granting and revoking Real Application
Security privileges on a particular schema to a database or application user as shown in
the following syntax descriptions:



Where the schema parameter is the schema on which the privilege is granted. The value is NULL if the privilege is a system privilege.

- The system security class ADMIN\_SEC\_POLICY privilege is extended to schemas for policy management (Create, Read, Update, and Delete) operations. So a policy administrator can grant ADMIN\_SEC\_POLICY privilege on a particular schema to a user to manage policy artifacts within granted schemas and apply policy management for individual applications. The APIs that are affected by this enhancement include the Real Application Security administrator packages: XS\_ACL, XS\_DATA\_SECURITY, and XS\_SECURITY\_CLASS
- A new system security class APPLY\_SEC\_POLICY privilege is added for policy
  enforcement to allow a policy administrator to enforce a policy within granted
  schemas within one application. The following data security APIs are checked
  before enforcing data security policies:

```
XS_DATA_SECURITY.APPLY_OBJECT_POLICY
XS_DATA_SECURITY.REMOVE_OBJECT_POLICY
XS_DATA_SECURITY.ENABLE_OBJECT_POLICY
XS_DATA_SECURITY.DISABLE_OBJECT_POLICY
```

- Auditing of GRANT\_SYSTEM\_PRIVILEGE procedure is provided with the audit action AUDIT GRANT PRIVILEGE.
- Auditing of REVOKE\_SYSTEM\_PRIVILEGE procedure is provided with the audit action AUDIT REVOKE PRIVILEGE.
- A new data dictionary view DBA\_XS\_PRIVILEGE\_GRANTS is added to show all the Real Application Security system or schema level privilege grants in the database.
- In addition, the following views are added: ALL\_XS\_SECURITY\_CLASSES,
   ALL\_XS\_SECURITY\_CLASS\_DEP, ALL\_XS\_PRIVILEGES,
   ALL\_XS\_IMPLIED\_PRIVILEGES, ALL\_XS\_ACLS, ALL\_XS\_ACES,
   ALL\_XS\_POLICIES, ALL\_XS\_REALM\_CONSTRAINTS,
   ALL\_XS\_INHERITED\_REALMS, ALL\_XS\_ACL\_PARAMETERS,
   ALL\_XS\_COLUMN\_CONSTRAINTS, ALL\_XS\_APPLIED\_POLICIES, and
   DBA\_XS\_PRIVILEGE\_GRANTS

This section includes the following topic: Setting Up and Enabling a Schema Level Data Security Policy.



## 5.11.1 Setting Up and Enabling a Schema Level Data Security Policy

Describes how to set up and enable a schema level data security policy for two application administrators.

The following set of examples describe how to set up and enable a schema level data security policy for two application administrators to administer two different schemas. Later, it will show how to disable the data security policy and revoke system privileges from these two application administrator users.

Create the application administrator users, then grant them the roles they need.

```
EXEC SYS.XS_PRINCIPAL.CREATE_USER(NAME => 'app_admin_user1', SCHEMA => 'HR');
EXEC SYS.XS_PRINCIPAL.SET_PASSWORD('app_admin_user1', 'PASSWORD');
EXEC SYS.XS_PRINCIPAL.GRANT_ROLES('app_admin_user1', 'XSCONNECT');

EXEC SYS.XS_PRINCIPAL.CREATE_USER(NAME => 'app_admin_user2', SCHEMA => 'SH');
EXEC SYS.XS_PRINCIPAL.SET_PASSWORD('app_admin_user2', 'PASSWORD');
EXEC SYS.XS_PRINCIPAL.GRANT_ROLES('app_admin_user2', 'XSCONNECT');
```

The Real Application Security Administrator with either SYS or a user granted GRANT ANY PRIVILEGE grants the system privileges ADMIN\_SEC\_POLICY and APPLY\_SEC\_POLICY to each application administrator user on the respective HR and SH schemas to the Real Application Security user, PTYPE XS.

```
EXEC SYS.XS_ADMIN_UTIL.GRANT_SYSTEM_PRIVILEGE('ADMIN_SEC_POLICY', 'app_admin_user1', SYS.XS_ADMIN_UTIL.PTYPE_XS, 'HR');

EXEC SYS.XS_ADMIN_UTIL.GRANT_SYSTEM_PRIVILEGE('APPLY_SEC_POLICY', 'app_admin_user1', SYS.XS_ADMIN_UTIL.PTYPE_XS, 'HR');

EXEC SYS.XS_ADMIN_UTIL.GRANT_SYSTEM_PRIVILEGE('ADMIN_SEC_POLICY', 'app_admin_user2', SYS.XS_ADMIN_UTIL.PTYPE_XS, 'SH');

EXEC SYS.XS_ADMIN_UTIL.GRANT_SYSTEM_PRIVILEGE('APPLY_SEC_POLICY', 'app_admin_user2', SYS.XS_ADMIN_UTIL.PTYPE_XS, 'SH');
```

Next, the policy administrator applies the desired object policy to a particular table in an application and enables it. Once created, then the policy administrator applies the data security policy EMPLOYEES DS to the EMPLOYEES table in the HR schema.



# Disabling the Data Security Policy and Revoking the System Privileges from the User

Describes how to disable data security policy and revoke the system privileges from the user.

# How to Disable the Data Security Policy and Revoke the System Privileges from the User

To disable the EMPLOYEES\_DS data security policy for the EMPLOYEES table in the HR schema, the policy administrator does the following:

To disable the CUSTOMERS\_DS data security policy for the CUSTOMERS table in the SH schema, the policy administrator does the following:

To revoke the system privileges from application administrator users <code>app\_admin\_user1</code> and <code>app\_admin\_user2</code> not from the role <code>policy\_admin\_role</code> because there may be other policy administrators with this same role enabled, the Real Application Security Administrator with either <code>sys</code> privilege or a user granted <code>GRANT ANY PRIVILEGE</code> privilege revokes the system privileges <code>ADMIN\_SEC\_POLICY</code> and <code>APPLY\_SEC\_POLICY</code> from application users <code>app\_admin\_user1</code> and <code>app\_admin\_user2</code> as follows:

```
EXEC SYS.XS_ADMIN_UTIL.REVOKE_SYSTEM_PRIVILEGE('APPLY_SEC_POLICY', 'app_admin_user1', SYS.XS_ADMIN_UTIL.PTYPE_XS, 'HR');
```



```
EXEC SYS.XS_ADMIN_UTIL.REVOKE_SYSTEM_PRIVILEGE('ADMIN_SEC_POLICY', 'app_admin_user1', SYS.XS_ADMIN_UTIL.PTYPE_XS, 'HR');

EXEC SYS.XS_ADMIN_UTIL.REVOKE_SYSTEM_PRIVILEGE('APPLY_SEC_POLICY', 'app_admin_user2', SYS.XS_ADMIN_UTIL.PTYPE_XS, 'SH');

EXEC SYS.XS_ADMIN_UTIL.REVOKE_SYSTEM_PRIVILEGE('ADMIN_SEC_POLICY', 'app_admin_user2', SYS.XS_ADMIN_UTIL.PTYPE_XS, 'SH');
```



6

# Using Real Application Security in Java Applications

This chapter describes how to use Real Application Security in Java applications. This chapter contains the following sections:

- About Initializing the Middle Tier
- About Managing Real Application Security Sessions
- Authenticating Application Users Using Java APIs
- About Authorizing Application Users Using ACLs
- Human Resources Administration Use Case: Implementation in Java

# 6.1 About Initializing the Middle Tier

The XSSessionManager class manages the life cycle of the session. It provides methods to create, attach, assign, detach, and destroy sessions. It also provides methods to perform cache activities.

This section describes the following topics:

- About Mid-Tier Configuration Mode
- Using the getSessionManager Method
- · About Changing the Middle-Tier Cache Setting

# 6.1.1 About Mid-Tier Configuration Mode

You can use one mid-tier configuration mode:

Dispatcher mode - get a session manager with dispatcher connections

In dispatcher mode, the dispatcher user must have session administration and cache access privileges. The application user does not need any session or cache privilege. The two predefined database roles,  $xs_session_admin$  and  $xs_cache_admin$ , can be granted to the dispatcher.

For best security practices, the application user should be given the least amount of privilege, therefore dispatcher mode is the recommended mid-tier configuration.

## 6.1.2 Using the getSessionManager Method

There is one way to get a session manager following the mid-tier configuration mode described in "About Mid-Tier Configuration Mode":

 Pass a connection or a pool of connections of the dispatcher user. In this way, the needed privileges are granted to the dispatcher. The two predefined roles, xs\_session\_admin and xs\_cache\_admin, should be granted to the dispatcher user. The dispatcher user is a direct logon Real Application Security user.

Using the dispatcher mode, you can initiate the Real Application Security middle tier by getting an instance of the session manager (see Example 6-1). Use the getSessionManager method (in bold typeface) of the XSSessionManager class to get an instance of the session manager. This method initializes a Real Application Security session manager by using either a single connection or a pool of connections. The caller of the getSessionManager method should have the Java Authentication and Authorization Service (JAAS) permission

XSSecurityPermission("initSecurityManager").

## **Privileges for the Session Manager**

Real Application Security session manager is initialized with a connection of a privileged user, who authorizes the session operations on behalf of the regular Real Application Security application users. If the session manager has the session operation privileges, then, each application user under the session manager does not need to have session operation privileges, and the application user's session operations can be performed as a trusted party. The session manager authorizes session operations for a connection, so you do not need to grant the createSession and attachToSession privileges directly to the regular Real Application Security application user. This session manager must have the following privileges:

- Real Application Security database object privileges to manage cached data in the middle tier.
- Session life cycle management privileges for the session manager to create or attach sessions on behalf of Real Application Security application user and external users.

#### **Roles for the Session Manager**

The session manager needs the following two roles to have the privileges mentioned in "Privileges for the Session Manager":

- A database role xs cache admin with the following privileges:
  - Privilege to query Real Application Security entities and to synchronize metadata
  - Privilege to execute code for the key exchange
- A Real Application Security role, xs session admin, with ADMIN SESSION privilege

These roles are predefined in the system.

# Example 6-1 How to Get an Instance of the Session Manager in Java Using a Single Connection

```
static XSSessionManager manager;
static Connection dispatcherConn = null;
int cacheMaxIdleTime=30;
int cacheMaxsize=2048000;
String host;
String port;
String sid;
...
dispatcherConn = DriverManager.getConnection("jdbc:oracle:thin:@" + host + ":" + port + ":" + sid, dispatcherUser, dispatcherPassword);
...
```



manager = XSSessionManager.getSessionManager(dispatcherConn, cacheMaxIdleTime, cacheMaxsize);

# 6.1.3 About Changing the Middle-Tier Cache Setting

Once the session manager is initialized, it starts to add some data like the ACL and Security class information to the cache. This cache data can be reused. The cache is initialized with its default settings that can be changed later.

This section describes the following topics:

- About Setting the Maximum Cache Idle Time
- About Setting the Maximum Cache Size
- About Getting the Maximum Cache Idle Time
- About Getting the Maximum Cache Size
- About Removing Entries from the Cache
- About Clearing the Cache

## 6.1.3.1 About Setting the Maximum Cache Idle Time

To set the maximum cache idle time, use the <code>setCacheMaxIdleTime</code> method of the <code>XSSessionManager</code> class. The <code>setCacheMaxIdleTime</code> method sets the maximum number of minutes that the cache can go without updating.

If an attempt is made to fetch objects from the cache and the XSSessionManager has not called the updateCache method for a period of time equal to the value set by the setCacheMaxIdleTime method, then, before returning any objects, the updateCache method is invoked forcefully to check that all the cached objects are still valid. The caller of the setCacheMaxIdleTime method must have the JAAS permission

XSSecurityPermission("setCacheMaxIdleTime").

## 6.1.3.2 About Setting the Maximum Cache Size

To set the maximum cache size, use the <code>setCacheMaxSize</code> method of the <code>XSSessionManager</code> class. This method sets the size of the cache on the middle tier.

The default size of the cache is 10MB. The minimum cache size is 1MB. The caller of the setCacheMaxSize method must have the JAAS permission

XSSecurityPermission("setCacheMaxSize").

## 6.1.3.3 About Getting the Maximum Cache Idle Time

To get the maximum cache idle time, use the getCacheMaxIdleTime method of the XSSessionManager class. This method returns the maximum number of minutes for which the cache does not have an updateCache call to update the cache. The caller of the getCachemaxIdleTime method must have the JAAS permission 
XSSecurityPermission("getCacheMaxIdleTime").

## 6.1.3.4 About Getting the Maximum Cache Size

To get the maximum cache size, use the <code>getCacheMaxSize</code> method of the <code>XSSessionManager</code> class. This method returns the maximum size of the cache in bytes. The caller of the



getCacheMaxSize method must have the JAAS permission XSSecurityPermission("getCacheMaxSize").

## 6.1.3.5 About Removing Entries from the Cache

To remove entries from the cache, a cache eviction algorithm is used, along with watermark levels. A watermark level determines how long data should stay in memory cache before being removed. When the cache size reaches the high watermark, then the cache eviction algorithm removes entries until the cache size reaches the low watermark.

This section describes the following activities for removing entries from the cache:

- About Setting the WaterMark
- About Getting the High WaterMark
- About Getting the Low WaterMark

## 6.1.3.5.1 About Setting the WaterMark

To set the watermark, use the setWaterMark method from the XSSessionManager class. The caller of the setWaterMark method must have the JAAS permission XSSecurityPermission ("setWaterMark").

## 6.1.3.5.2 About Getting the High WaterMark

To get the high watermark for cache, use the <code>getHighWaterMark</code> method from the <code>XSSessionManager</code> class.

## 6.1.3.5.3 About Getting the Low WaterMark

To get the low watermark for cache, use the <code>getLowWaterMark</code> method from the <code>XSSessionManager class</code>.

## 6.1.3.6 About Clearing the Cache

To clear the cache explicitly from the middle tier, use the clearCache method of the XSSessionManager class. This method explicitly clears the shared cache from the middle tier. The caller of the clearCache method must have the JAAS permission XSSecurityPermission ("clearCache").

# 6.2 About Managing Real Application Security Sessions

This section describes the following topics:

- Creating a Real Application Security User Session
- Attaching an Application Session
- Assigning or Switching an Application User
- Enabling Real Application Security Application Roles
- About Performing Namespace Operations as Session User
- About Performing Miscellaneous Session-Related Activities



- Detaching an Application Session
- Destroying A Real Application Security Application Session

## 6.2.1 Creating a Real Application Security User Session

To create a Real Application Security user session, for example, lws, for the application user lwuser, use the createSession method of the XSSessionManager class (see Example 6-2). The createSession method (in bold typeface). creates a session on the server with the specified parameters passed. A database round-trip is required to perform this operation.

To create an anonymous Real Application Security application session, use the createAnonymousSession method of the XSSessionManager class. The application user for this session is a predefined anonymous user, so no user parameter is passed in this method.

Both methods support using a cookie and a namespace.

The cookie, passed as the parameter, can be used to identify the newly created Real Application Security application session in future calls, until the cookie value is changed or the session is destroyed.

The namespace, passed as the parameter, can be used to create a namespace in the session. For details, see "About Performing Namespace Operations as Session User".

It is possible to reassign a specific application user to take over this session. In this case, some of the state of the session for the anonymous user is still preserved. For details, see "Assigning or Switching an Application User".

## Example 6-2 How to Create a Real Application Security Session in Java

```
Session lws = null;
static XSSessionManager manager;
static Connection lwsConn = null;
static String user = "lwuser";
String cookie="nst";
...
lws = manager.createSession(lwsConn, user, cookie, null);
```

## 6.2.2 Attaching an Application Session

To attach an application session, use the attachSession method of the XSSessionManager class (see Example 6-3). The attachSession method (in bold typeface) attaches the JDBC connection to the specified Real Application Security application session object. It also enables or disables the dynamic application roles, creates namespaces of the session, and sets the authentication time.

You can also attach to a session by using either ID or cookie as shown in Example 6-4. See Example 7-2 for another example of attaching to a session by using a cookie.

#### Example 6-3 How to Attach a Real Application Security Session in Java

```
Session lws = null;
static Connection lwsConn = null;
static XSSessionManager manager;
static String user = "lwuser";
String cookie = "lwscookie";
List <String> edynamicRoles = new ArrayList <String>();
edynamicRoles.add("EDYNROLE001");
```



```
edynamicRoles.add("EDYNROLE002");
List <String> ddynamicRoles = new ArrayList <String>();
ddynamicRoles.add("DDYNROLE001");
ddynamicRoles.add("DDYNROLE002");
...
lws = manager.createSession(lwsConn, user, cookie, null);
manager.attachSession(lwsConn, lws, edynamicRoles, ddynamicRoles, null, new
Timestamp(System.currentTimeMillis()));
```

### Example 6-4 How to Attach Using a Cookie

```
Session lws = null;
static Connection lwsConn = null;
static XSSessionManager manager;
...
lws = manager.attachSessionByCookie(lwsConn, "myCookie", null, null, null, null);
```

## 6.2.3 Assigning or Switching an Application User

If you have an anonymous session, you can reassign it to another application user later. Otherwise, if your session is assigned to an application user already, you can switch the session to another application user. In either case, the session must be attached first, before assigning or switching an application user.

To assign a name to a previously anonymous application user, use the <code>assignUser</code> method of the <code>XSSessionManager</code> class (see <code>Example 6-5</code>). The <code>assignUser</code> method (in bold typeface) changes the session context (user and roles) to the given user, for example, <code>lwuser</code>, but keeps the existing namespace. It can also change the session at the same time, by any given dynamic roles and namespace parameters, in the same way as the <code>attachSession</code> method. The associated session attributes remain in effect unless they are removed through another call.

To change a session user from a named user (non-anonymous) to another named user, use the <code>switchUser</code> method of the Session object.

Any request for retaining the dynamic application roles, which were assigned while attaching the session, is disabled. The dynamic application roles are retained for the new application user only when they are also included in the dynamic application roles list for the new application user. The associated session attributes remain in effect unless the session attributes list is reset.

This method changes the session context (user and roles) to the target user (see "Switching a Current Application User to Another Application User in the Current Application Session" for details about roles change), but *not* keeping the existing namespace by default. If you want to retain the existing namespace, you can use the switchUserKeepState method of the Session object. It can also change the session at the same time, by any given dynamic roles and namespace parameters, in the same way as the attachSession method.

Example 6-6 demonstrates how to switch the application user from lwuser to lwuser1. The switchUser method is in bold typeface.

### Example 6-5 How to Assign an Application User to a Session in Java

```
Session lws = null;
static XSSessionManager manager;
static String user = "lwuser";
...
```



```
manager.assignUser(lws, user, null, null, null, new
Timestamp(System.currentTimeMillis()));
```

## Example 6-6 How to Switch an Application User in a Session in Java

```
Session lws = null;
Vector<String> listOfNamespaces;
static String user = "lwuser";
List<String> nslist1 = new ArrayList<String>();
...
manager.assignUser(lws, user, nslist1, nslist2, nslist3, new
Timestamp(System.currentTimeMillis()));
...
lws.switchUser("lwuser1",listOfNamespaces);
```

## 6.2.4 Enabling Real Application Security Application Roles

A Real Application Security application role is a role that can be granted only to a Real Application Security application user or to another Real Application Security application role. Real Application Security application roles are granted database privileges through database roles. The database privileges are granted to a database role, which in turn is granted to a Real Application Security application role. For more information about Real Application Security application users and application roles, refer to "Principals: Users and Roles".

This section describes the following operation associated with application roles:

- Enabling a Real Application Security Application Role
- Disabling a Real Application Security Application Role
- Checking If a Real Application Security Application Role Is Enabled

## 6.2.4.1 Enabling a Real Application Security Application Role

To enable a Real Application Security application role granted to the current application user for the session, use the enableRole method of the Session interface (see Example 6-7).

The enableRole method (in bold typeface) has no effect if the particular application role is currently disabled. This operation requires a database round-trip.

## Example 6-7 How to Enable a Real Application Security Application Role in Java

```
static Session lws;
static Roles r1;
...
r1=new Role("HROLE1", null, 0);
lws.enableRole(r1);
```

## 6.2.4.2 Disabling a Real Application Security Application Role

To disable a Real Application Security application role granted to the current user for the session, use the disableRole method of the Session interface (see Example 6-8). This operation requires a database round-trip. The disableRole method is in bold typeface.

### Example 6-8 How to Disable a Real Application Security Application Role in Java

```
static Session lws;
static Roles r1;
...
r1=new Role("HROLE1", null,0);
```



```
lws.enableRole(r1);
...
lws.disableRole(r1);
```

## 6.2.4.3 Checking If a Real Application Security Application Role Is Enabled

To test if the specified application role is enabled in the Real Application Security application session, use the <code>isRoleEnabled</code> method of the <code>Session</code> interface (see <code>Example 6-9</code>). The <code>isRoleEnabled</code> method is in bold typeface.

This method does not have an associated database operation. You must have the administerSession Real Application Security application privilege to call this method.

# Example 6-9 How to Test If a Real Application Security Application Role Is Enabled in Java

```
static Session lws;
...
lws.enableRole("HROLE1");
...
boolean b = lws.isRoleEnabled("HROLE1");
```

## 6.2.5 About Performing Namespace Operations as Session User

A namespace is a group of additional attributes of the session context. An application uses a namespace to store application defined attribute-value pairs. The current session user should have <code>MODIFY\_NAMESPACE</code> (for namespace) and <code>MODIFY\_ATTRIBUTE</code> (for attribute) application privileges. For more information about namespaces, refer to "About Using Namespace Templates to Create Namespaces".

This section describes how to perform the following activities:

- Creating Namespaces
- Deleting Namespaces
- Implicitly Creating Namespaces
- About Using Namespace Attributes

## 6.2.5.1 Creating Namespaces

To create a namespace in Java, use the <code>createNamespace</code> method of the <code>Session</code> interface (see Example 6-10). The <code>createNamespace</code> method (in bold typeface) creates a new session namespace using the namespace template document, whose name matches with the specified name. If an event handler is specified in the template document, then the specified event handler applies to all the namespaces created using that template.



You can also create a namespace by passing a namespace name as a parameter with the <code>createSession</code> and <code>attachSession</code> methods discussed in the previous sections.



#### Example 6-10 How to Create a Namespace in Java

```
Session lws = null;
...
SessionNamespace ns = lws.createNamespace("TESTNS1");
```

## 6.2.5.2 Deleting Namespaces

To delete a namespace in Java, use the deleteNamespace method of the Session interface (see Example 6-11). The deleteNamespace method (in bold typeface) removes a namespace from a session.

## Example 6-11 How to Delete a Namespace in Java

```
Session lws = null;
...
SessionNamespace ns = lws.createNamespace("TESTNS1");
...
lws.deleteNamespace("TESTNS1");
```

# 6.2.5.3 Implicitly Creating Namespaces

To implicitly create the namespace object to represents the session namespace, use the getNamespace method of the Session interface (see Example 6-12). The getNamespace method is in bold typeface. If the namespace specified already exists, an error is thrown.

To retrieve a String representation of the namespace, use the toString method of the SessionNamespace interface.

#### Example 6-12 How to Implicitly Create the Namespace in Java

```
Session lws = null;
...
SessionNamespace ns2 = lws.getNamespace("TESTNS1");
```

## 6.2.5.4 About Using Namespace Attributes

A session namespace manages the attributes that a single application module stores for the duration of the session. The session namespace stores the attributes in a single namespace, a single set of access control restrictions, or a single event handler procedure that dispatches the attribute change events for that namespace.

This section describes how to perform the following activities:

- Creating a Session Namespace Attribute
- About Setting a Session Namespace Attribute
- Getting a Session Namespace Attribute
- Listing Attributes
- Resetting Attributes
- Deleting Attributes



## 6.2.5.4.1 Creating a Session Namespace Attribute

To create a session namespace attribute in Java, use the createAttribute method of the SessionNamespace interface (see Example 6-13). The createAttribute method (in bold typeface) creates a new attribute in the namespace.

### Example 6-13 How to Create a Session Namespace Attribute in Java

```
String name1="empid';
String value1="JB007";
SessionNamespace ns;
...
SessionNamespaceAttribute sal=ns.createAttribute(name1,value1);
```

## 6.2.5.4.2 About Setting a Session Namespace Attribute

To set a session namespace attribute in Java, use the <code>setAttribute</code> method of the <code>SessionNamespace</code> interface.

## 6.2.5.4.3 Getting a Session Namespace Attribute

To retrieve a session namespace attribute in Java, use the <code>getAttribute</code> method of the <code>SessionNamespace</code> interface (see Example 6-14). The <code>getAttribute</code> method (in bold typeface) returns the attribute whose name is specified as the parameter.

#### Example 6-14 How to Retrieve a Session Namespace Attribute in Java

```
String name="empid';
String value="JB007";
SessionNamespace ns;
...
SessionNamespaceAttribute sa=ns.createAttribute(name,value);
...
String attrvalue = ns.getAttribute("empid").getValue();
ns.getAttribute("empid").setValue("newValue");
```

## 6.2.5.4.4 Listing Attributes

To list the attributes in the namespace, use the <code>listAttributes</code> method of the <code>SessionNamespace</code> interface (see Example 6-15). The <code>listAttributes</code> method (in bold typeface) returns a collection of the attribute names in the namespace

#### Example 6-15 How to List Attributes in Java

```
String namel="empid';
String valuel="JB007";
SessionNamespace ns;
...
SessionNamespaceAttribute sal=ns.createAttribute(name1,value1);
...
for (Enumeration e = ns.listAttributes() ; e.hasMoreElements() ;) {
    System.out.println(" -- " + e.nextElement());
}
...
```



## 6.2.5.4.5 Resetting Attributes

To reset an attribute in Java, use the resetAttribute method of the SessionNamespace interface (see Example 6-16). The resetAttribute method (in bold typeface) resets the attribute in the namespace to its default value.

## Example 6-16 How to Reset an Attribute in Java

```
String namel="empid';
String valuel="JB007";
SessionNamespace ns;
...
SessionNamespaceAttribute sal=ns.createAttribute(name1,value1);
...
ns.resetAttribute("empid");
```

## 6.2.5.4.6 Deleting Attributes

To delete an attribute in Java, use the deleteAttribute method of the SessionNamespace interface (see Example 6-17). The deleteAttribute method (in bold typeface) deletes the particular attribute in the namespace.

#### Example 6-17 How to Delete an Attribute in Java

```
String name1="empid';
String value1="JB007";
SessionNamespace ns;
...
SessionNamespaceAttribute sal=ns.createAttribute(name1,value1);
...
ns.deleteAttribute("empid");
```

## 6.2.6 About Performing Namespace Operations as Session Manager

Each namespace has an associated ACL to determine who can manipulate the namespace and its attributes. If an application does not want the current session user to manipulate the namespace, but allows a session manager to do it, this can be done as session manager XSSessionManager.

XSSessionManager has a set of overloaded methods as Session, to manage the namespace. The usage is similar to that described for session user in "About Performing Namespace Operations as Session User".

Note that the session manager instance <code>XSSessionManager</code> may not be available to the application code; only the trusted infrastructure layer can use the session manager to manipulate such a secured namespace.

# 6.2.7 About Performing Miscellaneous Session-Related Activities

This section describes the following topics:

- About Getting the Oracle Connection Associated with the Session
- About Getting the Application User ID for the Session
- Getting the Session ID for the Session



- About Getting a String Representation of the Session
- · Getting the Session Cookie
- Setting Session Inactivity Timeout as Session Manager
- Setting the Session Cookie as Session Manager

## 6.2.7.1 About Getting the Oracle Connection Associated with the Session

To get the Oracle connection associated with the session, if it is currently bound to one, use the getConnection method of the Session interface.

## 6.2.7.2 About Getting the Application User ID for the Session

To get the application user identifier (ID) for a particular session, use the getUserId method of the Session interface.

To check if the application user for the session is anonymous, use the <code>isAnonymous</code> method of the <code>Session</code> interface.

## 6.2.7.3 Getting the Session ID for the Session

To get the session identifier (ID) for a particular session, use the getId method of the Session interface (see Example 6-18). The getId method is in bold typeface.

#### **Example 6-18** How to Get the Session ID for the Session in Java

```
Session lws=null;
...
System.out.println( "The Session ID is" + lws.getId());
```

## 6.2.7.4 About Getting a String Representation of the Session

To get a String representation of the session, use the toString method of the Session interface.

## 6.2.7.5 Getting the Session Cookie

To get the secure session cookie used for the session, use the <code>getSessionCookie</code> method of the <code>Session</code> interface (see Example 6-19). The <code>getSessionCookie</code> method is in bold typeface.

### Example 6-19 How to Get the Secure Session Cookie in Java

```
static Session lws;
...
System.out.println(lws.getSessionCookie());
```

## 6.2.7.6 Setting Session Inactivity Timeout as Session Manager

To set the timeout on the session, use the setInactivityTimeout method of the SessionManager interface. This method sets the session timeout in minutes.

The setInactivityTimeout method overrides the normal session timeout configuration. The method is:

```
sessionManager.setInactivityTimeout(Session session, int minutes);
```

## 6.2.7.7 Setting the Session Cookie as Session Manager

To set the secure session cookie used for the session, use the setCookie method of the SessionManager interface (see Example 6-20). The setCookie method (in bold typeface) returns the secure session cookie used for this session. The method is:

```
sessionManager.setCookie(lws,"newCookieValue");
```

#### **Example 6-20** How to Set the Secure Session Cookie in Java

```
static XSSessionManager manager;
...
manager.sessionManager.setCookie(lws,"chocolate chip");
```

## 6.2.8 Detaching an Application Session

To detach a Real Application Security application session in Java, use the detachSession method of the XSSessionManager class (see Example 6-21). The detachSession method (in bold typeface) detaches the session whose object it accepts as a parameter. The detachSession method call commits all changes in the request at the database level. A database round-trip is required to perform this operation.

## Example 6-21 How to Detach a Real Application Security Session in Java

```
Session lws = null;
static XSSessionManager manager;
static Connection lwsConn = null;
static String user = "lwuser";
String cookie;
...
lws = manager.createSession(lwsConn, user, cookie, null);
manager.attachSession(lwsConn, lws, null, null, null, new
Timestamp(System.currentTimeMillis()));
...
manager.detachSession(lws);
```

# 6.2.9 Destroying A Real Application Security Application Session

To destroy a Real Application Security application session in Java, use the <code>destroySession</code> method of the <code>XSSessionManager</code> class (see Example 6-22). The <code>destroySession</code> method (in bold typeface) accepts the database connection object and a session object as parameters. After you call this method, the destroyed session can no longer be accessed from any JVM. A database round-trip is required to perform this operation and for create session as well.

#### Example 6-22 How to Destroy a Real Application Security Session in Java

```
Session lws = null;
static Connection lwsConn = null;
static XSSessionManager manager;
static String user = "lwuser";
String cookie;
...
lws = manager.createSession(lwsConn, user, cookie, null);
manager.attachSession(lwsConn, lws, null, null, new
Timestamp(System.currentTimeMillis()));
```



```
manager.detachSession(lws);
manager.destroySession(lwsConn, lws);
```

# 6.3 Authenticating Application Users Using Java APIs

Authenticating application users is a main security function needed by applications. The XSAuthenticationModule class is used for authenticating application users. The authenticate method of the XSAuthenticationModule class is used to verify the application user credentials (see Example 6-23). The authenticate method is in bold typeface.

## **Example 6-23** How to Authenticate Application Users in Java

```
boolean authOk = false;
String dbUser;
String passwd;
String host;
String port;
String sid;
...
authOk = XSAuthenticationModule.authenticate(host + ":" + port + ":" + sid,
dbUser, passwd);
...
```

# 6.4 About Authorizing Application Users Using ACLs

Authorization is another main security feature needed by applications. In Real Application Security, the authorization policy comprises of the Access Control Lists (ACLs) and the application privileges. They are defined in the Real Application Security database and managed in a cache in the middle tier. The application privileges are data privileges. Data privileges are used to define the access of a function or operation to data. Once a function attaches a connection to the session, any query passed through the connection is automatically enforced by the database server.

The Aclid class provides various methods to perform the following:

- Constructing an ACL Identifier
- Using the checkAcl Method
- About Getting Data Privileges Associated with a Specific ACL

# 6.4.1 Constructing an ACL Identifier

To construct an Access Control List (ACL) identifier, use one of the overloaded parameterized constructors of the Aclid class (see Example 6-24). If you want to construct an ACL identifier from raw binary data, then use the following constructor:

```
public AclId(byte[] raw)
```

When you invoke this constructor, an ACL identifier, using raw binary returned from the ora get aclids operator of a query, is created.



If you want to construct an ACL identifier from internal ACL identifiers, then use the following constructor:

```
public AclId(java.util.List<java.lang.Long> ids)
```

When you invoke this constructor, it creates an ACL identifier using internal ACL identifiers.

#### Example 6-24 How to Construct an ACL Identifier

```
Session lws = null;
static byte[] aclRaw;
...
AclId id = new AclId(aclRaw);
boolean ret = lws.checkAcl(aclRaw, "UPDATE_INFO");
...
```

## 6.4.2 Using the checkAcl Method

To check one or more ACLs for specified data privileges, use the <code>checkAcl</code> method of the <code>XSAccessController</code> class. The data privileges are checked against one or more ACLs defined in the <code>Aclid</code> object. The <code>checkAcl</code> method returns <code>true</code> only when all the data privileges are granted in the ACLs. It is important to note that all privileges need not be granted in a single ACL. A session is needed for using the <code>checkAcl</code> method as <code>Example 6-25</code> indicates.

Example 6-25 demonstrates how to get the ACL associated with data privilege privileges22.

#### Example 6-25 How to get an ACL for a Specified Data Privilege

```
boolean ret;
Session lws = null;
AclId id2 = new AclId(ids);
List <String> privileges22 = new ArrayList<String>();
...
ret = XSAccessController.checkAcl(lws, id2, privileges22);
```

# 6.4.3 About Getting Data Privileges Associated with a Specific ACL

To get a collection of data privileges that are granted in the given ACL, for the given session, use the getPrivileges method of the Session class.



You use the <code>checkAcl</code> method for data security and the <code>checkPrivilege</code> method for function security.

# 6.5 Human Resources Administration Use Case: Implementation in Java

This section describes how to verify data security related application privileges at the middle tier. This Java example is based on the Security Human Resources (HR) scenario described in "Real Application Security: Putting It All Together". It uses the EMPLOYEES table in the

sample HR schema. The example uses two Real Application Security application users  ${\tt DAUSTIN}$  and  ${\tt SMAVRIS}$  to illustrate Real Application Security concepts. The example can be divided into the following modules:

- Setting Up the Mid-Tier Related Configuration
- Setting up the Connection and Initializing the Middle Tier
- Setting up the Session and Authorizing with Middle-Tier API
- Running a Query on the Database
- Performing Cleanup Operations
- The main Method

### **Setting Up the Mid-Tier Related Configuration**

To set up the mid-tier configuration involves creating a DISPATCHER user and password and granting this user the xscacfeadmin and xsessionadmin Real Application Security administrator privileges.

```
exec xs_principal.create_user(name=>'dispatcher', schema=>'HR');
exec sys.xs_principal.set_password('dispatcher', 'password');
exec xs_principal.grant_roles('dispatcher', 'xscacheadmin');
exec xs_principal.grant_roles('dispatcher', 'xssessionadmin');
```

## Setting up the Connection and Initializing the Middle Tier

This example uses the setupConnection method to create the connection to the database. The setupConnection method accepts a String array as argument, where:

```
args[0]=Database user
args[1]=Password
args[2]=Host
```

This method also initializes the middle tier by calling the <code>getSessionManager</code> method of the <code>oracle.security.xs.XSSecurityManager</code> class.

```
public static void setupConnection(String[] args) throws Exception {
    mgrConnection =
        DriverManager.getConnection(args[2], "dispatcher", "password");

    mgr = XSSessionManager.getSessionManager(mgrConnection, 30, 2048000);

    appConnection = DriverManager.getConnection(args[2], args[0], args[1]);
}
```

### Setting up the Session and Authorizing with Middle-Tier API

This example uses <code>queryAsUser</code> method to set up the session and authorize with the middle-tier <code>checkAcl</code> method. This example creates a session and attaches the session, and then calls the <code>queryEmployees</code> method. The <code>queryEmployees</code> method in "Running a Query on the <code>Database</code>" checks the ACL for the <code>UPDATE</code> privilege, and if <code>TRUE</code>, it allows the update; it checks the ACL again for the <code>VIEW\_SALARY</code> application privilege, and if <code>TRUE</code>, it allows access to the <code>SALARY</code> column and displays all the employees records including the sensitive data in the <code>SALARY</code> column. Then after displaying the employees records, it detaches the session, and destroys the session.



```
private static void queryAsUser(String user) throws SQLException {
    System.out.println("\nQuery HR.EMPLOYEES table as user \"" + user + "\"");
    try {
        Session lws = mgr.createSession(appConnection, user, null, null);
        mgr.attachSession(appConnection, lws, null, null, null, null, null);
        queryEmployees(lws);
        mgr.detachSession(lws);
        mgr.destroySession(appConnection, lws);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

#### Running a Query on the Database

This example uses the queryEmployees method to run a query on the HR database.

```
public static void queryEmployees(Session lws) throws SQLException {
    Connection conn = lws.getConnection();
    String query =
      " select email, first name, last name, department id, salary,
ora get aclids(emp) from hr.employees emp where department id in (40, 60, 100) order
by email";
    Statement stmt = null;
    ResultSet rs = null;
    System.out.printf(" EMAIL | FIRST NAME | LAST NAME | DEPT | SALARY | UPDATE |
VIEW SALARY\n");
    try {
      stmt = conn.createStatement();
      rs = stmt.executeQuery(query);
      while (rs.next()) {
        String email = rs.getString("EMAIL");
        String first name = rs.getString("FIRST NAME");
        String last name = rs.getString("LAST NAME");
        String department id = rs.getString("DEPARTMENT ID");
        String salary;
        if (((OracleResultSet)rs).getAuthorizationIndicator("SALARY") ==
AuthorizationIndicator.NONE) {
          salary = rs.getString("SALARY");
        else {
          salary = "*****";
        byte[] aclRaw = rs.getBytes(6);
        String update, viewSalary;
        if (XSAccessController.checkAcl(lws, aclRaw, "UPDATE")) {
          update = "true";
        }
```



```
else {
          update = "false";
        if (XSAccessController.checkAcl(lws, aclRaw, "VIEW SALARY")) {
          viewSalary = "true";
        else {
         viewSalary = "false";
        System.out.printf(\$9s|\$12s|\$12s|\$6s|\$8s|\$8s|\$8s|\$n", email,
                          first name, last name, department id,
                          salary, update, viewSalary);
       }
    } catch (Exception e) {
     e.printStackTrace();
    } finally {
     try { if (rs != null) rs.close(); } catch (Exception e) {};
     try { if (stmt != null) stmt.close(); } catch (Exception e) {};
 }
}
```

The queryEmployees method is run for both application users DAUSTIN and SMAVRIS.

## **Performing Cleanup Operations**

This examples uses the cleanup method for system cleanup operations.

```
public static void cleanupConnection() throws Exception {
   mgrConnection.close();
   appConnection.close();
```

#### The main Method

This section contains the main method for the Java example discussed. This section also contains the different packages that you must import to run the program.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import oracle.jdbc.OracleDriver;
import oracle.jdbc.OracleResultSet;
import oracle.jdbc.OracleResultSet.AuthorizationIndicator;

import oracle.security.xs.Role;
import oracle.security.xs.Session;
import oracle.security.xs.XSAccessController;
import oracle.security.xs.XSSessionManager;

/**
    * HR demo java version, check data security related privilege at mid-tier
```



```
* /
public class HRDemo {
  static Connection mgrConnection = null;
  static Connection appConnection = null;
  static XSSessionManager mgr = null;
  static String user = null;
  public static void main(String[] args) {
    try {
      DriverManager.registerDriver(new OracleDriver());
      if (args.length >=3) {
       user = args[0];
      } else {
        System.out.println("Usage HRDemo user pwd dbURL");
        System.exit(1);
      }
      setupConnection(args);
      queryAsUser("DAUSTIN");
      queryAsUser("SMAVRIS");
      cleanupConnection();
    } catch (Exception e1) {
      e1.printStackTrace();
  }
```

- Running the Security HR demo in Java assumes that the set up script described in "Setting Up the Security HR Demo Components" has been run to set up the Real Application Security components.
- 2. Compile the Java code.

```
$ORACLE_HOME/jdk6/bin/javac -classpath $ORACLE_HOME/rdbms_ho/jlib/
xs.jar:$ORACLE HOME/dbjava/lib/ojdbc6.jar HRdemo.java
```



You must use JDK 6 with xs.jar and ojdbc6.jar, which are located in the Oracle home directory. Different jars and JDK may not work.

3. Run the Java code.

```
$ORACLE_HOME/jdk6/bin/java -classpath $ORACLE_HOME/rdbms_ho/jlib/
xs.jar:$ORACLE_HOME/dbjava/lib/ojdbc6.jar
HRdemo db hr db hr jdbc:oracle:thin:@myserver:myport:mysid
```



## Output

Running the Security HR demo in Java assumes that the set up script described in "Setting Up the Security HR Demo Components" has been run to set up the Real Application Security components. When you run the Security HR demo, results of two queries are returned.

The first query runs with application user <code>DAUSTIN</code>, who has application roles <code>EMP\_ROLE</code> and <code>IT\_ROLE</code>, so he can view employee records in the IT department, but he cannot view the <code>SALARY</code> column except for his own salary record. The results of the query are as follows:

Query HR.EMF	LOYEES table	as user "DAU	JSTIN"			
EMAIL   F	'IRST_NAME   1	LAST_NAME	DEPT	SALARY	UPDATE	VIEW_SALARY
AHUNOLD	Alexander	Hunold	60	*****	false	false
BERNST	Bruce	Ernst	60	*****	false	false
DAUSTIN	David	Austin	60	4800	false	true
DLORENTZ	Diana	Lorentz	60	*****	false	false
VPATABAL	Valli	Pataballa	60	****	false	false

Note that application user  ${\tt DAUSTIN}$  can only view the  ${\tt SALARY}$  column data for his own record, and no others.

The second query runs with application user SMAVRIS, who has application roles  ${\tt EMP\_ROLE}$  and  ${\tt HR\_ROLE}$ , so she can view and update all the employee records. The results of the query are as follows:

Query HR.E	MPLOYEES table	e as user "SM	AVRIS"			
EMAIL	FIRST_NAME	LAST_NAME	DEPT	SALARY	UPDATE	VIEW_SALARY
AHUNOLD	Alexander	Hunold	60	9000	true	true
BERNST	Bruce	Ernst	60	6000	true	true
DAUSTIN	David	Austin	60	4800	true	true
DFAVIET	Daniel	Faviet	100	9000	true	true
DLORENTZ	Diana	Lorentz	60	4200	true	true
ISCIARRA	Ismael	Sciarra	100	7700	true	true
JCHEN	John	Chen	100	8200	true	true
JMURMAN	Jose Manuel	Urman	100	7800	true	true
LPOPP	Luis	Popp	100	6900	true	true
NGREENBE	Nancy	Greenberg	100	12008	true	true
SMAVRIS	Susan	Mavris	40	6500	true	true
VPATABAL	Valli	Pataballa	60	4800	true	true

Note that application user  ${\tt SMAVRIS}$  can view all the employee records, including all data in the  ${\tt SALARY}$  column.



7

# Oracle Fusion Middleware Integration with Real Application Security

Real Application Security adds external user and role support for application integration, that can be used, for example, with Oracle Fusion Middleware. For Oracle Fusion Middleware, the users and roles are also externalized to a common, single repository with centralized management and single authentication of the user interface using the Authorization Policy Manager. From a Real Application Security perspective, the integrated users and roles (including application roles) are externalized principals because Oracle Fusion Middleware manages them externally. The mid-tier initialization and authorization operations are the same as those described in Using Real Application Security in Java Applications.

This chapter describes the following topics:

- About External Users and External Roles
- Session APIs for External Users and Roles

## 7.1 About External Users and External Roles

An external user is an end-user accessing a service. User information is stored in the identity store, typically instantiated by the WebLogic Authenticator. This user is neither a database user nor a Real Application Security application user. An external user does not have any footprint in the database. But, an external user needs to access the database for application data. Therefore, a Real Application Security context (session) is established for such a user to control the user's access to the required data.

An anonymous user is an unauthenticated user, or a user whose credentials have not been validated. An anonymous user is permitted to access only unprotected resources such as public data from a database. An application can enable or disable the use of anonymous users.

An external role or group is a collection of users and other groups, which can be hierarchical. For example, a group can include arbitrarily nested groups.

An external application role is a collection of users, groups, and application roles, which can be hierarchical. This role is specific to the application, defined by the application policy, and may not be known to the J2EE container. Application roles are scoped because they are visible only when the application runs. They can be mapped to other application roles defined in the same application scope and also to enterprise users or groups. Application roles are used in authorization decisions.

Similar to external users, external roles and application roles have no footprint in the Real Application Security system. They are used to control the way the Real Application Security ACLs grant data access to an application.

External roles and application roles also enforce the details of data access. External users need some basic database privileges, typically the object privilege to run SELECT on an application table. These privileges can be granted through a Real Application Security dynamic application role, which is enabled when a user session is attached. For example, to grant privileges to an external user or role, specify the principal type as

XS\_ACL.PTYPE\_EXTERNAL in an ACE list when creating an ACL. See the "CREATE\_ACL Procedure" for more information.

#### Session Modes for External Users

Real Application Security supports the following two modes of operation for sessions:

Secure Mode

In secure mode, data security is enforced at the database server. By default, a session is created in a secure mode for all users.

Trusted mode

A trusted mode is a mode in which data security is enforced at the middle tier and not at the database server. In such a mode, the data security implemented by Real Application Security is bypassed. So, creating a session in trusted mode is a privileged operation.

Trusted mode is allowed only for external users, and only when the dispatcher has CREATE\_TRUSTED\_SESSION privilege. This privilege can be granted to the dispatcher user as follows:

```
XS_ADMIN_UTIL.grant_system_privilege('CREATE_TRUSTED_SESSION','dispatcher',
XS_ADMIN_UTIL.PTYPE_XS);
```

## 7.2 Session APIs for External Users and Roles

This section describes the following topics for external users and roles:

- Namespace for External Users
- Creating a Session
- Attaching a Session
- Assigning a User to a Session
- Saving a Session and Terminating a Session

## 7.2.1 Namespace for External Users

The namespaces for external users are enhanced with attribute manipulation features during creating, attaching, and assigning a session. External users are able to perform the following activities:

- Creating namespace with attributes while creating a session
- Setting namespace attributes while attaching a session and assigning a user
- Saving a session and leaving it as attached

# 7.2.2 Creating a Session

To create a Real Application Security application session, use the <code>createSession</code> method of the <code>XSSessionManager</code> class.

For external users, this method creates a Session object on the server as well as its corresponding middle-tier representation with namespaces and attributes. This method also creates the Namespaces and sets corresponding attributes given in the Namespace/AttributeValue. The cookie can be used to identify the newly created Real



Application Security application session in future calls, until the cookie value is changed or the session is destroyed.

#### **Syntax**

```
public abstract Session createSession(java.sql.Connection conn,
                                      ExternalUser eUser,
                                      java.lang.String cookie,
                                      java.util.Collection<NamespaceValue> nav)
                                      throws InvalidXSUserException,
                                             AccessDeniedException,
                                              java.sql.SQLException,
                                             XSSessionException,
                                              InvalidXSNamespaceException
public abstract Session createSessionTrusted(java.sql.Connection conn,
                             ExternalUser externalUser,
                             java.lang.String cookie,
                             java.util.Collection<NamespaceValue> nameSpaceValues)
                             throws InvalidXSUserException,
                             AccessDeniedException,
                             java.sql.SQLException,
                             SQLException,
                             XSException,
                             InvalidXSNamespaceException
```

#### **Parameter**

Parameter	Description
conn	The JDBC connection for database server roundtrip
eUser <b>or</b> externalUser	The external user associated with the session
cookie	The session cookie used to identify the external user
nav <b>or</b> nameSpaceValues	A list of namespaces with corresponding attributes to be created for the namespaces

#### **Example**

Example 7-1 demonstrates how to create a Real Application Security session for external users. The createSession method is in bold typeface.

#### Example 7-1 Creating a Real Application Security Session for External Users

```
.
.
.
static Connection lws_conn =null;
static XSSessionManager sm = null;
lws_conn = DriverManager.getConnection(lws_conn_string, username, password);
sm = XSSessionManager.getSessionManager(privConn, 20, 29999999);
.
.
.
String trituser = "TUSER01";
String cookie = "some_cookie";
String extuser = "ExtPrincp";
```



```
String extuuid = "ExtPrincp";
Session lws = null;
List<AttributeValue> nsavList = new ArrayList<AttributeValue>();
AttributeValue nsav1 = new AttributeValue("ATTR01", "value1");
nsavList.add(nsav1):
AttributeValue nsav2 = new AttributeValue("ATTR02", "value2");
nsavList.add(nsav2);
NamespaceValue nav = new NamespaceValue("NST01", nsavList);
List<NamespaceValue> nsList = new ArrayList();
nsList.add(nav);
/* create session with external user name in secure mode with namespace attr-
vals and cookie */
lws = sm.createSession(lws conn, new ExternalUser(extuser, extuuid), cookie,
nsList);
sm.destroySession(lws_conn, lws);
/*Create external user session in secure mode*/
lws = sm.createSession(lws_conn, new ExternalUser(extuser, extuuid), null, null);
sm.destroySession(lws_conn, lws);
/*Create external user session in secure mode with namespace attribute values */
lws = sm.createSession(lws_conn, new ExternalUser(extuser, extuuid), null,
nsList);
sm.destroySession(lws conn, lws);
/* create session with external user name in secure mode with cookie */
lws = sm.createSession(lws_conn, new ExternalUser(extuser, extuuid), cookie,
sm.destroySession(lws conn, lws);
/* create trusted session with only external user name */
lws = sm.createSessionTrusted(lws conn, new ExternalUser(extuser, extuuid),
null, null);
sm.destroySession(lws_conn, lws);
/* create session with RAS user name in secure mode with namespace and cookie */
lws = sm.createSession(lws_conn, trituser, cookie, nsList);
sm.destroySession(lws conn, lws);
```

## 7.2.3 Attaching a Session

To attach an application session, use the attachSession method of the XSSessionManager class.

For external users, this method attaches the JDBC connection to the specified session object. This method also sets the dynamic application roles, external roles, authentication time, and creates namespaces for the session. It also gives a list of a namespace and its corresponding namespace attributes to be created and set. If the namespace does not exist, then this method creates the namespace, and then sets the corresponding attributes.

### **Syntax**

```
java.util.Collection<java.lang.String> disabledDynamicRoles,
             java.util.Collection<ExternalRole> externalRoles,
             java.util.Collection<NamespaceValue> nav,
             java.sql.Timestamp authenticationTime)
             throws java.sql.SQLException,
                    AccessDeniedException,
             InvalidSessionException,
             XSSessionException,
             InvalidXSNamespaceException
public abstract Session attachSessionByCookie(
          java.sql.Connection conn,
          java.lang.String cookie,
          java.util.Collection<java.lang.String> enabledDynamicRoles,
          java.util.Collection<java.lang.String> disabledDynamicRoles,
          java.util.Collection<oracle.security.xs.ExternalRole> externalRoles,
          java.util.Collection<oracle.security.xs.NamespaceValue> namespaceValues,
          java.sql.Timestamp authenticationTime)
          throws java.sql.SQLException,
                 AccessDeniedException,
                 InvalidSessionException,
                 XSException,
                 InvalidXSNamespaceException
public abstract Session attachSessionByID(
          java.sql.Connection conn,
          java.lang.String id,
          java.util.Collection<java.lang.String> enabledDynamicRoles,
          java.util.Collection<java.lang.String> disabledDynamicRoles,
          java.util.Collection<oracle.security.xs.ExternalRole> externalRoles,
          java.util.Collection<oracle.security.xs.NamespaceValue> namespaceValues,
          java.sql.Timestamp authenticationTime)
          throws java.sql.SQLException,
                 AccessDeniedException,
                 InvalidSessionException,
                 XSException,
                 InvalidXSNamespaceException
```

#### **Parameters**

Parameter	Description
conn	The database connection to be attached to the application session
session	The Session object to be attached
cookie	The session cookie
id	The session identifier
enabledDynamicRoles	A collection of dynamic application role names to be enabled
disabledDynamicRoles	A collection of dynamic application role names to be disabled
externalRoles	A collection of external roles to be enabled
nav <b>or</b> namespaceValues	A list of namespaces with corresponding attributes to be set
authenticationTime	The authentication time to be sent to the database server

#### **Example**

Example 7-2 demonstrates how to attach a Real Application Security session for external users. The attachSession method is in bold typeface.

#### External Role Behavior while Attaching a Session

- After an external role is enabled for a session, it is stored as part of the session context as an ID. This role ID is used in access control, when you call the checkAcl method on both middle tier and database server. This is same as regular Real Application Security application role or dynamic application role.
- A Real Application Security ID is assigned for every external role passed while attaching a session, whether the role is referred by ACL or not.
- The scope of the external role is within the boundary of attaching or detaching a
  session. An external role cannot be enabled for attaching multiple sessions, and it
  does not need to be explicitly disabled. So, the roles assigned for attaching the
  first session will not be automatically enabled while attaching the next session,
  unless the roles are assigned again.

This behavior is completely different from the behavior of regular Real Application Security application roles or dynamic application roles, where the application roles assigned for attaching the first session are automatically enabled while attaching the next session.

 After a session is attached, the external role remains consistent till detaching and reattach the session. The role may even be revoked for the user.

#### Example 7-2 Attaching a Real Application Security Session for External Users

```
static Connection lws conn =null;
static XSSessionManager sm = null;
lws conn = DriverManager.getConnection(lws conn string, username, password);
sm = XSSessionManager.getSessionManager(privConn, 20, 29999999);
String cookie = "some cookie";
String extuser = "ExtPrincp";
String extuuid = "ExtPrincp";
Session lws = null;
Session lws2 = null
List<AttributeValue> nsavList = new ArrayList<AttributeValue>();
AttributeValue nsav1 = new AttributeValue("ATTR01", "value1");
nsavList.add(nsav1);
AttributeValue nsav2 = new AttributeValue("ATTR02", "value2");
nsavList.add(nsav2);
NamespaceValue nav = new NamespaceValue("NST01", nsavList);
List<NamespaceValue> nsList = new ArrayList();
nsList.add(nav);
List <String> dynamicRoles = new ArrayList <String>();
dynamicRoles.add("DYNROLE001");
dynamicRoles.add("DYNROLE002");
```



```
List <ExternalRole> extRoles = new ArrayList <ExternalRole>();
extRoles.add(new ExternalRole("EXTPRIN01"));
extRoles.add(new ExternalRole("MYEXTPRIN02"));
lws = sm.createSession(lws_conn, new ExternalUser(extuser, extuuid), cookie +
"secure", nsList, false);
sm.attachSession(lws conn, lws, enabledDynamicRoles, disabledDynamicRoles, extRoles,
null, null);
sm.detachSession(lws);
sm.attachSession(lws conn, lws, enabledDynamicRoles, disabledDynamicRoles, extRoles,
null, new Timestamp(System.currentTimeMillis()));
sm.detachSession(lws);
sm.attachSession(lws conn, lws, enabledDynamicRoles, disabledDynamicRoles, extRoles,
nsList, null);
sm.detachSession(lws);
sm.attachSession(lws conn, lws, enabledDynamicRoles, disabledDynamicRoles, extRoles,
nsList, new Timestamp(System.currentTimeMillis()));
sm.detachSession(lws);
lws2 = sm.createSession(lws conn, new ExternalUser(extuser, extuuid), cookie +
"trusted", nsList, true);
lws2 = sm.attachSessionByCookie(lws conn, lws.getSessionCookie(), null,
enabledDynamicRoles, disabledDynamicRoles, extRoles, null, null);
sm.detachSession(lws2);
lws2 = sm.attachSessionByCookie(lws conn, lws.getSessionCookie(), null,
enabledDynamicRoles, disabledDynamicRoles, extRoles, nsList, new
Timestamp(System.currentTimeMillis()));
sm.detachSession(lws2);
```

## 7.2.4 Assigning a User to a Session

To assign a name to a previously anonymous user, use the <code>assignUser</code> method of the <code>XSSessionManager</code> class.

For external users, this method assigns a named user to a previously anonymous user, sets the dynamic application roles, external role, and authentication time. If a list of Namespace/ Attribute values is given, this method creates each namespace that does not exist, and sets the corresponding attributes.

#### **Syntax**



#### **Parameters**

Parameters	Description
session	The session object to assign the user to
targetUser	An ExternalUser object initialized based on authentication
enabledDynamicRoles	A list of dynamic application role names to be enabled
disabledDynamicRoles	A list of dynamic application role names to be disabled
externalRoles	A collection of external roles to be enabled
namespaceValues	A list of namespaces with corresponding attributes to be set
authenticationTime	The a timestamp indicated when the user authenticated

### **Example**

Example 7-3 demonstrates how to assign a Real Application Security session to external users. The assignUser method is in bold typeface.

# Example 7-3 How to Assign a Real Application Security Session to External Users

```
static Connection lws conn =null;
static XSSessionManager sm = null;
lws_conn = DriverManager.getConnection(lws_conn_string, username, password);
sm = XSSessionManager.getSessionManager(privConn, 20, 29999999);
String cookie = "some cookie";
String extuser = "ExtPrincp";
String extuuid = "ExtPrincp";
Session lws = null;
List<AttributeValue> nsavList = new ArrayList<AttributeValue>();
AttributeValue nsav1 = new AttributeValue("ATTR01", "value1");
nsavList.add(nsav1);
AttributeValue nsav2 = new AttributeValue("ATTR02", "value2");
nsavList.add(nsav2);
NamespaceValue nav = new NamespaceValue("NST01", nsavList);
List<NamespaceValue> nsList = new ArrayList();
nsList.add(nav);
List <String> dynamicRoles = new ArrayList <String>();
dynamicRoles.add("DYNROLE001");
dynamicRoles.add("DYNROLE002");
List <ExternalRole> extRoles = new ArrayList <ExternalRole>();
```



```
extRoles.add(new ExternalRole("EXTPRIN01"));
extRoles.add(new ExternalRole("MYEXTPRIN02"));

lws = sm.createAnonymousSession(lws_conn, cookie + "trusted", nsList, true);
sm.attachSession(lws_conn, lws, null, null, null, null, null);
sm.assignUser(lws, euser, dynamicRoles, dynamicRoles, extRoles, null, null);
sm.detachSession(lws);

lws = sm.createAnonymousSession(lws_conn, cookie + "secure", nsList, false);
sm.attachSession(lws_conn, lws, null, null, null, null), null);
sm.assignUser(lws, euser, dynamicRoles, dynamicRoles, extRoles, null, new
Timestamp(System.currentTimeMillis()));
sm.detachSession(lws);

lws = sm.createAnonymousSession(lws_conn, cookie + "trusted", nsList, true);
sm.attachSession(lws_conn, lws, null, null, null, null);
sm.assignUser(lws, euser, dynamicRoles, dynamicRoles, null, nsList, null);
sm.assignUser(lws, euser, dynamicRoles, dynamicRoles, null, nsList, null);
sm.detachSession(lws);
```

## 7.2.5 Saving a Session and Terminating a Session

To save the changes of a session at the database server and keep the session still attached, use the saveSession method of the XSSessionManager class.

For external users, this method saves the current session. Similar to the <code>detachSession</code> method, this method commits all session changes to the back end and a database roundtrip is required to perform this operation. But, unlike the <code>detachSession</code> method, this method keeps the session attached. This method is mainly used to save an application context (namespace).

To terminate the changes of a session at the database server and detach from the session, use the abortSession method of the XSSessionManager class.

## **Syntax**

#### **Example**

Example 7-4 demonstrates how to save a Real Application Security external user session. The saveSession method is in bold typeface.

#### Example 7-4 How to Save a Real Application Security External User Session

```
.
.
static Connection lws_conn =null;
static XSSessionManager sm = null;
.
```



```
lws conn = DriverManager.getConnection(lws conn string, username, password);
sm = XSSessionManager.getSessionManager(privConn, 20, 29999999);
String cookie = "some_cookie";
String extuser = "ExtPrincp";
String extuuid = "ExtPrincp";
Session lws = null;
List<AttributeValue> nsavList = new ArrayList<AttributeValue>();
AttributeValue nsav1 = new AttributeValue("ATTR01", "value1");
nsavList.add(nsav1);
AttributeValue nsav2 = new AttributeValue("ATTR02", "value2");
nsavList.add(nsav2);
NamespaceValue nav = new NamespaceValue("NST01", nsavList);
List<NamespaceValue> nsList = new ArrayList();
nsList.add(nav);
List <String> dynamicRoles = new ArrayList <String>();
dynamicRoles.add("DYNROLE001");
dynamicRoles.add("DYNROLE002");
List <ExternalRole> extRoles = new ArrayList <ExternalRole>();
extRoles.add(new ExternalRole("EXTPRIN01"));
extRoles.add(new ExternalRole("MYEXTPRIN02"));
lws = sm.createAnonymousSession(lws conn, cookie + "trusted", nsList, true);
sm.attachSession(lws conn, lws, null, null, null, null, null);
sm.assignUser(lws, euser, dynamicRoles, dynamicRoles, extRoles, null, null);
lws.deleteNamespace("NST01");
sm.saveSession(lws);
```



8

# Application Session Service in Oracle Fusion Middleware

Real Application Security provides an application session service in Oracle Fusion Middleware to set up an application session transparently and securely that supports existing application users, roles, and security context. This application session service is a servlet filter that is responsible for application session setup and a set of APIs that the application can use with the application session. This application session service supports user and roles managed externally by Oracle Fusion Middleware.

Beginning with Oracle Database 12c Release 1 (12.1.0.2), this application session service supports a Java EE Web application using Oracle Platform Security Service (OPSS) as the application security provider. This application session service can be deployed to the Java EE container that OPSS can support, together with the application.

This chapter describes the following topics:

- About Real Application Security Concepts
- About Application Session Service in Oracle Fusion Middleware
- About the Application Session Filter
- About Deployment
- About Application Configuration of the Application Session Filter
- Domain Configuration: Setting Up an Application Session Service to Work with OPSS and Oracle Fusion Middleware
- About Application Session APIs
- · Human Resources Demo Use Case: Implementation in Java

# 8.1 About Real Application Security Concepts

As an Oracle Database authorization system, Real Application Security supports application security by enforcing who (application user) can do what application-level operations (ApprovePurchaseOrder, ViewSSN) on which database resource (purchase order records of employees under my report, my SSN). An application session is used to enforce application security. Typically, the users and roles are provisioned externally, that is, enterprise users are provisioned in an identity store and application roles are managed in a policy store, such as, Oracle Identity Management and Oracle Entitlement Server (OES).

#### **Application Users and Roles Managed Externally**

Real Application Security supports users and roles that are provisioned by an external party, such as Oracle Entitlement Server for managing application users and roles provisioning, while OPSS provides a runtime security framework for enforcing security for application roles. Theses are referred to as external application users and application roles (see Oracle Fusion Middleware Integration with Real Application Security for more information.)

Real Application Security also has users and roles for the application natively managed in the database, and these are referred to as Real Application Security application users and application roles (see Configuring Application Users and Application Roles for more information).

For external application users and application roles, Real Application Security does not manage user provisioning including users' role assignment. However, for native application users and application roles in the database, grants of application roles to application users, database roles to application roles, and application roles to application roles are managed in the database. Both Real Application Security application users and application roles, and external application users and application roles are supported in an application session, and can be used in a data security policy. An application privilege can be granted to users managed both in the identity store externally or in the database natively.

### **Application Session in Oracle Fusion Middleware**

An application session represents an application user's runtime security context, which includes the user identity, database and application roles, and namespace attribute values. The application session here in Oracle Fusion Middleware is using externally managed user and roles. See Configuring Application Sessions for more information about configuring an application session.

#### **Session Manager in Oracle Fusion Middleware**

In Real Application Security, the session manager authorizes the application session operation and has the necessary privileges to create or modify the application session. The application code or application database connection should *not* have these privileges. To the database, the session manager is a Real Application Security direct logon user (see "About Creating a Direct Login Application User Account"). It communicates with the database at the beginning of application session service initialization to build a trust relation with the database server based on authorization credentials. This mechanism is used subsequently to further authorize the application session operations on behalf of the application.

## **Dynamic Roles in Oracle Fusion Middleware**

Other than the application roles, an application session supports a dynamic role. This is a type of Real Application Security role that must be defined natively in the database (see "Dynamic Application Roles"). This role is not granted to the user or other roles. It must be enabled programmatically in the application session at run time. This can be done by the Real Application Security filter automatically or by the trusted application code explicitly.

The dynamic role can be defined as request scope or session scope. Session scope means the enabled dynamic role is still enabled in the next attach, unless you explicitly specify that it is disabled in the next attach. Request scope means that the role is disabled after the application session is detached from the connection.

Dynamic role serves two general purposes:

#### Object privilege

An application user is not a database user. These object privileges can be granted to a Real Application Security dynamic role when application users and roles are provisioned in external identity stores. When the Real Application Security filter sets up the application session for the application user, it enables the dynamic role



in every application session accessing the current application. The dynamic role is specific to the current application only.

Application Session privilege elevation

Certain trusted application code must temporarily have higher privileges in order to do some database operations. This is supported by enabling a Real Application Security dynamic role during application session attach from the trusted code declared using a Java code based policy. The role should be disabled upon detach.

One use case is application namespace setup where session namespace attributes are secured in Real Application Security in a fine grained manner. The namespace must be predefined at the database as a namespace template. Upon definition, in the associated ACL of the namespace authorization policy can be specified, that is, who (user/role) can do what (modify\_namespace, modify\_attribute) on the namespace. To ensure that only trusted application code can modify the namespace attributes, the privileges are granted to a dynamic role. Also, the dynamic role can only be programmatically enabled by certain trusted application code identified by Java code permission. This supports the use case that only the trusted code can set up certain namespaces.

# 8.2 About Application Session Service in Oracle Fusion Middleware

Figure 8-1 shows application session service as it is implemented in Oracle Fusion Middleware.



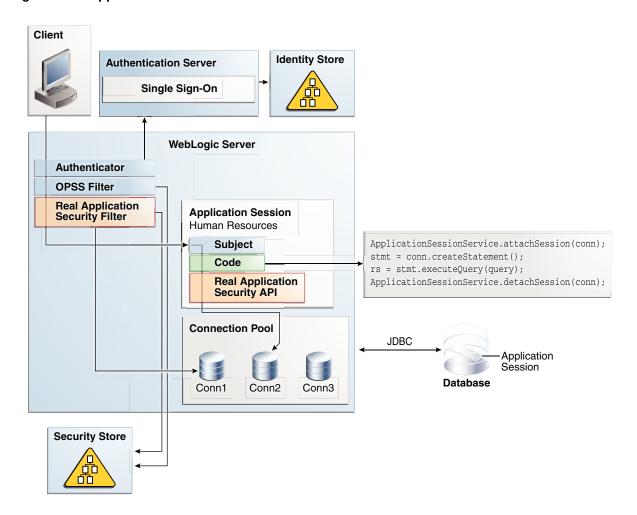


Figure 8-1 Application Session Service in Oracle Fusion Middleware

An application session service is an integrated solution with Oracle Fusion Middleware, to leverage Oracle Fusion Middleware to provide an application session at the database. In Oracle Fusion Middleware:

- The application user is authenticated by the container. In WLS, typically the authenticator works with the SSO server to authenticate the user.
- The application user and group are managed by the Identity Store.
- OPSS is an application security framework to set up the application security context based on the container's security context. See *Oracle Application Server Containers for J2EE Security Guide* for more information about application security with OPSS.

The Real Application Security servlet filter sets up the application session transparently and synchronizes the application session with the OPSS subject. The server filter code consists of a set of APIs that function in the application session to:

- Attach, detach, and destroy the session (see "About Application Session APIs")
- Provide privilege elevation (see "About the Privilege Elevation API")
- Provide namespace operations (see "About Namespace APIs")



Provide authorization (see "About the Check Privilege API")

Real Application Security provides:

- APIs that support external users and roles in the application session
- Authorizes the session operation through the session manager
- Support for fine-grained access control on namespace

## 8.3 About the Application Session Filter

The Real Application Security application session filter is a standard Java EE servlet filter that implements the <code>javax.servlet.Filter</code> interface. The basic function of this filter is to set up an application session transparently according to the authenticated user's security context (OPSS Subject).

This application session filter allows the application session to be continuously shared among applications. It cannot be created for every request, but must be tied to a stateful context and reused for the same user until logout. For web applications, the http session is such a context. It is maintained by the container for the same user's continuous access from logon until logout, across multiple single sign-on applications or containers.

The http session object is always accessible from the ServletFilter, but may not be accessible from the generic application code.

This section includes the following topic: About the Application Session Filter Operation.

## 8.3.1 About the Application Session Filter Operation

The application session filter sets up the application session in the following manner:

- It creates an application session at the user's first access.
  - If the user has been logged in, it creates the application session as the user in the authentication context (OPSS Subject).
  - If the user has not been logged in, it creates the application session as an anonymous user.
- It reuses the existing application session instance for the user's subsequent access to the same application.
- It shares the same application session among multiple applications when multiple applications access the same Real Application Security database.
- It synchronizes the application session at the beginning of each http request to make sure the user and roles in the current application session are always synchronized with the authentication context (OPSS Subject), and only the configured dynamic roles are enabled for every application session.

The synchronization is done by pushing the OPSS Subject values to the server and getting back the server computed values for the current application session.

User and roles in the application session are fixed once the filter is fired before application code execution. The filter is responsible for synchronizing the user and roles, not application code.

Application code is responsible for the namespace setup. The filter can only help to bring back the previous namespace. See "About Namespace APIs" for more information about namespace setup.



The application session is cached locally based on the http session ID. The http session is managed by the container. Real Application Security has an application session listener to listen for the container's application session event. When the http session is invalidated by the container, the application session is removed from the local cache by the Real Application Security listener.

## 8.4 About Deployment

Real Application Security application session service is delivered in one jar file, xsee.jar. Oracle recommends that you deploy the xsee.jar jar file to a common directory, not together with the web application (WAR file inside web-inf/lib). In this way, you can separate the jar from application code, and grant some special code based permissions to only the xsee.jar jar file, and not to the application code.

For the xsee.jar jar file to get the session manager's credential from the CSF store, you must grant code based permission <code>CredentialAccessPermission</code> to the <code>xsee.jar</code> jar file. The filter internally uses Real Application Security session manager to authorize the session operation.

In Example 8-1, the xsee.jar jar file is deployed to WLS's domain /lib directory. The java policy file (system-jazn-data.xml) has the CredentialAccessPermission grant, assuming that the session manager's key/map is using the default value.

For deployment instructions, see the section about standard Java EE deployment in *Understanding Oracle WebLogic Server*.

For a simple and quick method of deploying an application for testing or evaluation, use Auto-Deployment. This is an easier way to deploy the application session service by packaging everything (class, web.xml) in to one WAR file, and copying it to the Weblogic autodeploy directory. See the section about auto-deploying applications in development domains in *Deploying Applications to Oracle WebLogic Server*.

To create the session manager's credential, see Step 2 in "Manual Configuration" for more information.

## Example 8-1 Granting the Code-Based Permission CredentialAccessPermission to the xsee.jar File



# 8.5 About Application Configuration of the Application Session Filter

The filter is configured in the application's web.xml configuration file in a standard way. It can be configured to apply to only specific URLs. This avoids unnecessary application session setup for certain pages for which it does not need database access. The filter assumes that user authentication has been done and an authentication context has been established. In OPSS, the user's application context is computed at the OPSS filter, so the OPSS filter must be deployed ahead of the application session filter in the filter chain. The application session filter uses the following web.xml parameters:

application.datasource

The application uses this application.datasource parameter. The application session filter requires this parameter for initialization, application session setup and namespace operations.

dynamic.roles

A list of Real Application Security dynamic roles to be used are separated by a comma(,). The dynamic roles must already be created at the database as session scope; otherwise, the following exception is thrown: ORA-46055: invalid role specified.

The roles are enabled for every application session in the current application, and automatically disabled in other applications. Note that these dynamic roles are enabled for the anonymous session. You should not over grant any privileges to dynamic roles if they are not needed for every application session. Normally, only object privileges should be granted to the dynamic roles.

For any tables not protected by Real Application Security, the application still has the flexibility to use the database connection pool user for access, not the application user. In that case, no attach application session API call is needed and no object privilege is granted to the dynamic roles.

session.manager.pwd.key and session.manager.pwd.map

The session.manager.pwd.key parameter and the session.manager.pwd.map parameter (fixed as oracle.rdbms.ras) point to a credential (user ID and password) in the credential store. The session.manager.pwd.key parameter is used to retrieve the session manager's credential. Currently, the OPSS CSF credential store is used to store the credential, and the CSF API is used to retrieve the credential at run time. In addition, both the session manager's user ID and password can be retrieved from the store.

The default value is default for the session.manager.pwd.key parameter. If the application is using the default credential, then this parameter can be omitted.

If an application wants to use a specific session manager, not the default credential, the application's administrator must create the credential with a different key name, and configure it using this parameter. See configuring the OPSS security store in *Oracle Application Server Containers for J2EE Security Guide* for more information.

session.manager.pool.min and session.manager.pool.max

The session manager's connection is also used to query the data security policy (ACL) at the mid-tier. This connection is managed as a pool. The <code>session.manager.pool.min</code> parameter determines the minimum size of the pool. This parameter is optional. The default value is 1.



The session.manager.pool.max parameter determines the maximum size of the pool. This parameter is optional. The default value is 3.

If the privilege check is not needed, the pool size should be set to 1 for both session.manager.pool.min and session.manager.pool.max values.

Example 8-2 shows an application session filter sample configuration that includes the servlet filter, its parameters, and the listener. Any parameters, which have default values, are omitted from this example.

#### **Example 8-2** Application Session Filter Sample Configuration

# 8.6 Domain Configuration: Setting Up an Application Session Service to Work with OPSS and Oracle Fusion Middleware

This section describes the prerequisites and configuration required for an application to use an application session service.

This section includes the following topics:

- Prerequisites
- Manual Configuration
- About Automatic Configuration

## 8.6.1 Prerequisites

To use Real Application Security, both the application session service and OPSS must be deployed and configured in a Oracle Fusion Middleware's Java EE container.

For WebLogic server, the prerequisites include:

- A JRF based WLS domain (OPSS is built-in) certified with the Oracle database 12c JDBC driver. The required JDBC jars could be many, not just one driver jar depending on the features you need (UCP, I18N, SQLXML and so forth).
- Oracle Database 12c Release 1 (12.1) and later



For WebLogic server 10.3.6 and 12.1.2 JRF release (part of Oracle Fusion Middleware), the JDBC driver shipped is not Oracle Database 12c compatible. You must obtain the Oracle Database 12c JDBC jars (ojdbc6.jar or ojdbc7.jar and other matched jars depending on the features you need), and add these jars to the front of your WebLogic Server's classpath. For detailed instruction, see *Administering JDBC Data Sources for Oracle WebLogic Server*, Section B.

If there is version mismatch between the JDBC driver and the database, the Real Application Security filter initialization fails with an error message. For example,

- If the Oracle Database 11g JDBC driver is being used with Oracle Database 12c, the following error message appears in the server log: Fail to initialize RAS session manager due to method missing.
- If the Oracle Database 12c JDBC driver is being used with Oracle Database 11g, the following error message appears in the server log: ORA-00439: feature not enabled: Fusion Security.

## 8.6.2 Manual Configuration

Follow these manual configuration steps for an application to use an application session service. These steps should work for both WebLogic 10.3.6 and 12.1.2, JRF release.

1. Install the Real Application Security jars.

Copy the <code>xsee.jar</code> and <code>xs.jar</code> (<code>ORACLE\_HOME/jlib/</code>) to a common directory that applications can consume. For WebLogic, a good location is <code>DOMAIN\_HOME/lib</code>. This allows Real Application Security jars to be shared by many applications deployed in the same domain.

2. Create a Real Application Security session manager credential.

As discussed in "About Application Configuration of the Application Session Filter", a session manager's credential must be created in OPSS's credential store. This can be done using an OPSS script. For details about how to use OPSS script, see the section about the OPSS script in *Oracle Application Server Containers for J2EE Security Guide*.

```
createCred(map='oracle.rdbms.ras', key='default', user='myUsr',
password='myPassword')
```

The session manager's credential is stored in the default credential store, which is configured for the domain. The map name must be oracle.rdbms.ras, which is predefined for the Real Application Security application session service. This is fixed and cannot be changed.

3. Grant code permission to the Real Application Security jar files.

As discussed in "About Deployment", the CSF permission must be granted to the xsee.jar file. This is also done using OPSS script.

```
grantPermission(codeBaseURL='file:${domain.home}/lib/xsee.jar',
permClass='oracle.security.jps.service.credstore.CredentialAccessPermission',
permTarget='context=SYSTEM, mapName=oracle.rdbms.ras, keyName=*', permActions='read')
```

Note that the above keyName~(\*) is for all keys. No further grants are needed for a non-default key, if it is created for a specific application.

4. Configure web.xml, invoke Real Application Security APIs (attach/detach), and build/deploy the application. See Example 8-2 to see how web.xml is configured.



These are standard Java EE development procedures.

If the attachSessionPrivileged API is invoked in the application code, SessionCodePermission must be granted to the application code as discussed in "About the Privilege Elevation API". That is similar to step 3. Here is an example:

```
grantPermission(codeBaseURL='file:${domain.home}/servers/DefaultServer/tmp/
_WL_user/MyWar/pi47ig/war/WEB-INF/lib/trusted.jar', permClass='
oracle.security.xs.ee.session.SessionCodePermission', permTarget='
MY_NS_DROLE, permActions='attach')
grantPermission(codeBaseURL='file:${domain.home}/lib/xsee.jar', permClass='
oracle.security.xs.ee.session.SessionCodePermission', permTarget='
MY_NS_DROLE, permActions='attach')
```

OPSS scripts require that the WLS administrative server is running. This manual approach only supports online configuration. Step 4 is always the responsibility of the application administrator, while Steps 1 through 3 can be automated as discussed in "About Automatic Configuration".

## 8.6.3 About Automatic Configuration

With Oracle Fusion Middleware, you can use a configuration utility to configure common settings for a group of applications. For WebLogic, this is the domain configuration wizard. In a future WLS release (release 12.1.3), the Steps 1 through 3 (in "Manual Configuration") could be automated by this configuration wizard. This automatic approach also has the advantage of supporting offline configuration (when the administrative server is not running).

When the configuration wizard is started (<ORACLE\_HOME>/oracle\_common/common/bin/config.sh), the following user interfaces (UIs) will be shown to prompt for Real Application Security configuration information.

- In the first UI, the application session service is shown as one of the Oracle Fusion Middleware features for selection. Once selected, its dependency (OPSS, part of JRF) is automatically selected.
- In the second UI, you are prompted to enter the default session manager's credential.

There is no UI for granting code permission. This is automatically done by merging a predefined xml file to the domain's system-jazn-data.xml file. The predefined xml file contains all the Real Application Security code permission grants that are needed.

If the administrator decides to use a different session manager for an application, then the administrator must complete manual Step 2 or add a special key name from the UI. The same key name must be passed to the application's web.xml. In this case, the map name (store name) is still fixed as oracle.rdbms.ras, and you do not need to grant code permission because all keys have already been granted internally.

## 8.7 About Application Session APIs

All application session APIs are exposed through class <code>ApplicationSessionService</code> as static methods. The APIs operate on the current application session, which is set up based on the current Subject. Inside each API, an identity assertion is performed internally, to make sure the current application session matches the subject. If a mismatch is found, an <code>ApplicationSesseionException</code> exception is thrown. The caller



code of the application session API should always be executed inside Subject.doAs, to be invoked as the subject. See the JDK's Subject.doAs for more information.

This section describes the following topics:

- About Application Session APIs
- About the Privilege Elevation API
- About Namespace APIs
- About the Check Privilege API

## 8.7.1 About Application Session APIs

This section describes the following topics:

- About Attaching to an Application Session
- Detaching from an Application Session
- Destroying an Application Session

## 8.7.1.1 About Attaching to an Application Session

Attach the current user's application session to the given database connection.

For application code to attach to the current user's application session, no code based permission is needed. The application session works as is, no extra privilege is elevated through the attach.

#### **Syntax**

#### **Parameter**

Parameter	Description
conn	The JDBC connection for database server roundtrip

#### Example

See Example 8-3 and Example 8-6.

## 8.7.1.2 Detaching from an Application Session

Detach the current user's application session from the given database connection.

It is always a good practice to detach the application session at the application code's final block. Not doing so may give an attached connection to some code that is not running under the correct user. It is caller's responsibility to properly detach the application session once used.

If detach is not called, but attach is called again on the same connection, the server forces the detach from the previous attached application session, and attaches to the current application session.



#### **Syntax**

#### **Parameter**

Parameter	Description		
conn	The JDBC connection for database server roundtrip		

#### **Example**

Example 8-3 shows sample code that uses the attach and detach API with a database query. The caller must decide the boundary of the attach and detach calls, based on the needs of the query.

#### Example 8-3 Application Session APIs: AttachSession and DetachSession

```
* Typical application code calling attach/detach for database query
public void queryHR(Connection conn) {
      String query = " select emp.employee id, emp.salary from hr.employees emp";
     Statement stmt = null;
     ResultSet rs = null;
     String id, salary;
      try {
        // attach connection to the current application session
       ApplicationSessionService.attachSession(conn);
       stmt = conn.createStatement();
       rs = stmt.executeQuery(query);
       while (rs.next()) {
          id = rs.getString("employee id");
          salary = rs.getString("salary");
      } catch (ApplicationSessionException e) {
      } catch (SQLException e) {
      } finally {
         // detach the current application session from the connection
         try { ApplicationSessionService.detachSession(conn); } catch (Exception e) {}
         if (stmt != null) try {stmt.close();} catch (SQLException e) {};
         if (rs != null) try { rs.close();} catch (SQLException e{});
```

## 8.7.1.3 Destroying an Application Session

Destroys the current application session at the database, and removes it from current thread's execution context. This should be invoked by the application at logout. It destroys the current application session originally set up by the filter.

#### **Syntax**



#### **Parameter**

Parameter	Description
conn	The JDBC connection for database server roundtrip

#### **Example**

Example 8-4 shows sample code that destroys the application session service.

#### Example 8-4 Application Session APIs: DestroySession

```
void doLogout(HttpServletRequest request) {
    DataSource dataSource = null;
    Connection conn = null;
    try {
        InitialContext ic;
        try {
            ic = new InitialContext();
            dataSource = (DataSource)ic.lookup("jdbc/myDBDS");
            if (dataSource != null)
                try {
                    conn = dataSource.getConnection();
                } catch (SQLException e) {
                    e.printStackTrace();
        } catch (NamingException e) {
            e.printStackTrace();
        // invalidate Http session
        request.getSession().invalidate();
        // destroy XS session at DB
        ApplicationSessionService.destroySession(conn);
    } catch (ApplicationSessionException e) {
        e.printStackTrace();
    } finally {
        if (conn != null)
            try {
               conn.close();
            } catch (SQLException e) {
```

## 8.7.2 About the Privilege Elevation API

This section describes the following topic: Enabling a Dynamic Role in the Application Session.

## 8.7.2.1 Enabling a Dynamic Role in the Application Session

Attaches the current application session to a given database connection, and enables the Real Application Security dynamic role in the attached application session. This allows trusted application code to have higher privileges temporarily in order to perform some database operations, such as setting up application namespace.

This is for certain trusted application code to elevate the application session privilege. A Real Application Security dynamic role is enabled during attach. The trusted code is identified by java code permission.

#### **Syntax**

#### **Parameter**

Parameter	Description				
conn	The JDBC connection for database server roundtrip				
role	The given dynamic role; must be request scope				

#### **Example**

Each given dynamic role is associated with a code base permission as shown in Example 8-5 (permission grant in jazn-data.xml)

See Example 8-6.

#### **Usage Notes**

The permission is always checked internally in the API, whether the java security manager is on or off. If the caller has the permission (that implies that the given role also matches the role defined in the policy file), the given dynamic role is enabled during attach; otherwise, the API fails with an AccessControlException.

The caller code (caller.jar file) and application session service code (xsee.jar) should both have the SessioncodePermission permission. This is sufficient when the caller.jar is invoked directly by the container. When caller.jar is invoked by another application code, it is up to the caller to decide whether the application code needs to have this permission. If the caller does not need the application to have this permission, the caller can invoke attachSessionPrivileged under AccessController.doPrivileged with a null AccessControllerContext. See the Java API for details. By doing this, the caller.jar fully trusts the application code.

Note that the dynamic role is only enabled on the attached application session, not the current application session. It is enabled within the window of attach and detach. The dynamic role must be defined as request scope at the database; otherwise, the following exception ORA-46055: invalid role specified is thrown.

#### Example 8-5 Privilege Elevation API

<grant>

<grantee>



```
<codesource>
                    <url>file:${domain.home}/servers/DefaultServer/tmp/ WL user/MyWar/pi47ig/war/WEB-
INF/lib/trusted.jar' </url>
                </codesource>
            </grantee>
            <permissions>
                <permission>
                    <class>oracle.security.xs.ee.session.SessionCodePermission</class>
                    <name> MY NS DROLE</name>
                    <actions>attach </actions>
                </permission>
            </permissions>
        </grant>
        <grant>
            <grantee>
                <codesource>
                    <url>file:${domain.home}/lib/xsee.jar</url>
                </codesource>
            </grantee>
            <permissions>
                <permission>
                    <class>oracle.security.xs.ee.session.SessionCodePermission</class>
                    <name>MY NS DROLE</name>
                    <actions>attac </actions>
                </permission>
            </permissions>
        </grant>
```

## 8.7.3 About Namespace APIs

This section describes the following topics:

- About Creating a Namespace
- About Deleting a Namespace
- About Setting the Namespace Attribute
- About Deleting a Namespace Attribute
- Getting a Namespace Attribute

## 8.7.3.1 About Creating a Namespace

Creates a namespace in the current application session. The namespace given must be predefined at the database, and the namespace ACL must allow the attached application session to perform a <code>MODIFY\_NAMESPACE</code> operation, unless the <code>ADMIN\_ANY\_NAMESPACE</code> privilege is enabled in the application session.

#### **Syntax**

#### **Parameter**

Parameter	Description
conn	The JDBC connection for database server roundtrip



Parameter	Description		
name	The given namespace name		

#### **Example**

See Example 8-6.

## 8.7.3.2 About Deleting a Namespace

Deletes a namespace from the current application session. The namespace given must be predefined at the database, and the namespace ACL must allow the attached application session to perform a MODIFY\_NAMESPACE operation, unless the ADMIN ANY NAMESPACE privilege is enabled in the application session.

#### **Syntax**

#### **Parameter**

Parameter	Description			
conn	The JDBC connection for database server roundtrip			
name	The given namespace name.			

#### **Example**

See Example 8-6.

## 8.7.3.3 About Setting the Namespace Attribute

Sets the attribute value to the namespace in the current application session. The namespace given must be predefined at the database, and the namespace ACL must allow the attached application session to perform a <code>MODIFY\_ATTRIBUTE</code> operation, unless the <code>ADMIN\_ANY\_NAMESPACE</code> privilege is enabled in the application session.

If the attribute does not exist on the namespace, the API creates the attribute with the given value; otherwise, it simply sets the existing value to the given value.

#### **Syntax**



#### **Parameter**

Parameter	Description			
conn	The JDBC connection for database server roundtrip			
name	The given namespace name			
attribute	The given namespace attribute name			
value	The given namespace attribute value			

#### **Example**

See Example 8-6.

## 8.7.3.4 About Deleting a Namespace Attribute

Deletes the attribute from the namespace in the current application session. The namespace given must be predefined at the database, and the namespace ACL must allow the attached application session to perform a MODIFY\_ATTRIBUTE operation, unless the ADMIN ANY NAMESPACE privilege is enabled in the application session.

#### **Syntax**

#### **Parameter**

Parameter	Description			
conn	The JDBC connection for database server roundtrip			
name	The given namespace name			
attribute	The given namespace attribute name			

#### **Example**

See Example 8-6.

## 8.7.3.5 Getting a Namespace Attribute

Gets the attribute from the namespace in the current application session. The given namespace must be created. No database connection is needed and no privilege is checked for this operation.

The APIs that change namespace (other than <code>getNamespaceAttribute</code>) have a database connection as an input parameter. Those APIs update the namespace in the current application session in the JVM, as well as serialize the change to the database table. The connection must be attached. It uses the attached application session to determine whether the server can authorize the namespace change.



To allow *only* certain trusted application code to set up namespace. The connection can be attached with a dynamic role, which has elevated privileges (MODIFY\_NAMESPACE, MODIFY\_ATTRIBUTE) on the namespace. This is achieved using the attachSessionPrivileged API, and only granting the namespace privileges to the dynamic role.

#### **Syntax**

#### **Parameter**

Parameter	Description			
name	The given namespace name			
attribute	The given namespace attribute name			

#### **Example**

Example 8-6 shows a sample servlet filter that sets up namespace using namespace APIs and uses the application session privilege elevation API.

#### **Important Points to Know About Using Application Namespace**

The following usage information summarizes important points about using application namespace.

- The Real Application Security filter caches all the application namespace to the current application session.
  - For first time access, a new application session must be created in the database. No application namespace has been set up yet at this time.
  - For the user's subsequent access, the filter always brings all the namespaces created for the application session, and caches them in the current application session in JVM.
- Application code always accesses namespace from the current application session. Each update operation is a round trip to the server to change the values in the table and current application session (JVM). That is why each update API has a database connection parameter. However, the read attribute is a local operation to read from the current application session in JVM without accessing the database.
- Whenever a namespace change is successfully done, the change is propagated to
  the already attached application sessions, as well as newly attached application
  sessions because all these attached application sessions refer to the single source
   the current application session.
- The namespace in the current application session is consistent within an http request scope for the web application. Even the namespace can be changed at any time by other applications. The change is only picked up once at the beginning of the current http request by the Real Application Security filter. All attaches that happen within the same http request refer to the same namespace in the current application session.



Application code has complete control for changing the namespace value. It can read the
current application session's namespace at any time and decide whether to update the
namespace by calling the namespace APIs.

#### Example 8-6 Namespace APIs

```
* Trusted application code (servlet filter) sets up namespace
  * Using privilege elevation and namespace APIs
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws
IOException, ServletException {
  Connection conn = null;
    conn = myDatasource.getConnection();
    // Attach an application session with a dynamic role.
    ApplicationSessionService.attachSessionPrivileged(conn, "myNSRole");
     try {
      // Get the current value.
      String currentValue = ApplicationSessionService.getNamespaceAttribute("mySecuredNS",
"myAttribute");
       // If the current value is not desired, set it.
       if ("myValue".compareToIgnoreCase(currentValue) != 0)
         ApplicationSessionService.setNamespaceAttribute(conn, "mySecuredNS", "myAttribute",
"myValue");
     } catch (NamespaceNotFoundException e) {
      // Namespace is not found, create it.
      ApplicationSessionService.createNamespace(conn, "mySecuredNS");
       // Set the attribute.
      ApplicationSessionService.setNamespaceAttribute(conn, "mySecuredNS", "myAttribute",
"myValue");
    }
   } catch (SQLException e) {
   } catch (ApplicationSessionException e) {
   } finally {
    // Detach an application session.
    try { ApplicationSessionService.detachSession(conn); } catch (Exception e) {}
    if (conn != null) try { conn.close();} catch (Exception e) {}
    // Execution of application code.
    chain.doFilter(request, response);
```

## 8.7.4 About the Check Privilege API

This section describes the following topic: Checking a Privilege on the ACLs.

## 8.7.4.1 Checking a Privilege on the ACLs

Checks the privilege on the ACLs using the attached application session of the given connection and includes these usage notes:

- An attached connection must be given. The privilege check is based on the attached application session. Note that an attached application session can have extra privileges compared to the current application session through the attachSessionPrivileged call.
- The API takes the input parameter of ACL IDs, which can be queried from the table using the ORA\_GET\_ACLID operator. The operator returns a set of ACL IDs associated with the current row.

• This API takes the input parameter of privilege name. This input parameter can be DML privileges, such as SELECT or UPDATE, or it can be any user defined privilege.

#### **Syntax**

#### **Parameter**

Parameter	Description				
conn	The JDBC connection for database server roundtrip				
acls	The given ACL IDs in row format				
privilege	The given privilege name				

#### **Example**

Example 8-7 shows getting the ACL associated with the row and checking the UPDATE privilege on the ACL.

#### Example 8-7 CheckPrivilege API

```
public Collection<Employee> queryHR(Connection conn ) {
  Statement stmt = null;
  ResultSet rs = null;
  Collection<Employee> result = new ArrayList<Employee>();
  try {
    // attach session
   ApplicationSessionService.attachSession(conn);
   stmt = conn.createStatement();
   rs = stmt.executeQuery(query);
   while (rs.next()) {
     Employee emp = new Employee();
     emp.setId(rs.getString("EMPLOYEE ID"));
     AuthorizationIndicator ai =
                ((OracleResultSet)rs).getAuthorizationIndicator("salary");
     if (ai == AuthorizationIndicator.NONE) {
        emp.setSalary(rs.getString("salary"));
      } else {
        emp.setSalary("*****") ;
     // get ACL associated with the row
     emp.setAcl(rs.getBytes("acl id"));
     // check "update" privilege
```



```
boolean canUpdate = ApplicationSessionService.checkPrivilege(conn, emp.getAcl(), "UPDATE");
    emp.setUpdate(canUpdate);
    result.add(emp);
   emp.setFname(rs.getString("first name"));
   emp.setLname(rs.getString("last name"));
   emp.setEmail(rs.getString("email"));
   emp.setPhone(rs.getString("phone number"));
   emp.setManagerId(rs.getString("manager id"));
    emp.setDepId(rs.getString("department id"));
} catch (ApplicationSessionException e) {
   e.printStackTrace();
    // process me
} catch (SQLException e) {
   // process me
   e.printStackTrace();
} finally {
  if (stmt != null) try {stmt.close();} catch (SQLException e) {};
  if (rs != null) try { rs.close();} catch (SQLException e) {};
   try {ApplicationSessionService.detachSession(conn);} catch (ApplicationSessionException e) {};
return result;
```

## 8.8 Human Resources Demo Use Case: Implementation in Java

This section describes how an application session service supports user and roles managed externally by Oracle Fusion Middleware. This Java example is based on the Security Human Resources (HR) scenario. It uses the EMPLOYEES table in the sample HR schema.

#### See Also:

For information about user and group to application roles mapping, see About the HR Demo Use Case - User Roles.

This example includes the following files and employee records that the three types of users can access:

- Setting Up the HR Demo Application for External Principals (setup.sql)
- About the Application Session Filter Configuration File (web.xml)
- About the Sample Servlet Application (MyHR.java)
- About the Filter to Set Up the Application Namespace (MyFilter.java)
- About the HR Demo (1) Logged in as Employee LPOPP
- · About the HR Demo (2) Logged in as HRMGR
- About the HR Demo (3) Logged in as a Team Manager



## 8.8.1 Setting Up the HR Demo Application for External Principals (setup.sql)

Example 8-8 shows a set up script (setup.sql) for setting up the HR Demo application for external principals.

This setup script performs the following operations:

- Creates a dynamic role, HROBJ, for object privileges for the external user
- Creates a security class, HRPRIVS, with privilege view\_sensitive\_info, and aggregate privilege update\_info that implies data privileges, update, delete, insert, which come from pre-defined security class DML.
- Creates an EMP ACL, EMP\_ACL, to grant EMP, HRMGR and HRREP privileges to access employee record in the restricted departments. Note that each external principal, (application role: HRREP, HRMGR, and EMP) must match the OPSS policy store GUID values.
- Creates an self ACL, SELF\_ACL, to grant EMP privileges for an employee to see and update their own record.
- Creates a Manager ACL, MGR\_ACL, to allow a manager to see their employee's salary information.
- Creates a data security policy, EMPLOYEE\_DS, for the EMPLOYEES table. The policy
  defines an instance set to control access to the employees in department 60 and
  100 to EMP\_ACL. It also defines an attribute constraint to control access to the
  sensitive SALARY column.
- Defines two additional instance sets to <code>SELF\_ACL</code> and <code>MGR\_ACL</code> that are appended to the data security policy, <code>EMPLOYEE DS</code>.
- Grants to the dispatcher some additional privileges.

#### Example 8-8 Set Up the HR Demo Application for External Principals

```
Rem Copyright (c) 2009, 2014, Oracle and/or its affiliates.
Rem All rights reserved.
SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100
-- A PL/SQL function to determine manager-report relationship
conn hr/hr;
create or replace package hrutil as
function ismyreport(id IN PLS INTEGER)
return PLS INTEGER ;
end hrutil;
create or replace package body hrutil as
function ismyreport(id IN PLS INTEGER)
```



```
return PLS INTEGER is
   mycount PLS INTEGER;
         PLS INTEGER ;
   select employee id into myid from hr.employees
   where UPPER(email) = XS SYS CONTEXT('PROFILE NS', 'EMAIL');
  select count (employee id) into mycount from hr.employees
  where employee id = id start with manager id = myid
  connect by prior employee id = manager id ;
  return mycount ;
end ismyreport;
end hrutil ;
-- Create a dynamic role for object privileges for external users.
connect sys/password as sysdba
show con name;
-- Create a dynamic role for HR object privileges.
exec xs principal.delete principal('HROBJ', XS ADMIN UTIL.CASCADE OPTION);
exec xs_principal.create_dynamic_role('HROBJ');
-- Create a db role to have HR object privileges.
drop role hr db obj;
create role hr db obj;
-- Grant object privilege to the db role.
grant select, insert, update, delete on hr.employees to hr db obj;
-- Grant db role to dynamic role.
grant hr db obj to HROBJ;
-- Create a security class with privilege view sensitive info, and
-- aggregate privilege update info that implies data privileges,
-- update, delete, insert, which come from pre-defined security class
-- DML.
DECLARE
 priv list XS$PRIVILEGE LIST;
BEGIN
 priv list :=XS$PRIVILEGE LIST(
    XS$PRIVILEGE(name=>'VIEW SENSITIVE INFO'),
    XS$PRIVILEGE(name=>'UPDATE INFO',
                  implied priv list=>XS$NAME LIST
                    ('"UPDATE"', '"DELETE"', '"INSERT"')));
 xs security class.create security class(
    name=>'HRPRIVS',
    parent list=>XS$NAME LIST('DML'),
    priv list=>priv list);
END;
-- External Principal (app role) Used for data security:
-- Such a principal must match the OPSS policy store.
-- roleName="HRREP" guid="37ED0D108C2F11E2BF802D569259982"
-- roleName="HRMGR" guid="4077A2B08C2F11E2BF802D569259982"
-- roleName="EMP"
                   guid="F917C3608CF011E2BF802D569259982"
-- Create an EMP Acl to grant EMP, HRMGR and HRREP privileges to access an employee record in the
restricted departments.
```

```
DECLARE
 ace list XS$ACE LIST;
BEGIN
  ace list := XS$ACE LIST(
      XS$ACE TYPE(privilege list=>XS$NAME LIST('"SELECT"','VIEW SENSITIVE INFO'),
                  granted=>true,
                  principal name=>'"37ED0D108C2F11E2BF802D569259982"',
principal type=>XS ACL.PTYPE EXTERNAL),
      XS$ACE TYPE(privilege list=>XS$NAME LIST('UPDATE INFO'),
                  granted=>true,
                  principal name=>'"4077A2B08C2F11E2BF802D569259982"',
principal type=>XS ACL.PTYPE EXTERNAL),
      XS$ACE TYPE (privilege list=>XS$NAME LIST('"SELECT"'),
                  granted=>true,
                  principal name=>'"F917C3608CF011E2BF802D569259982"',
principal type=>XS ACL.PTYPE EXTERNAL));
 xs acl.create acl(name=> 'EMP ACL',
                   ace list=> ace list,
                   sec class=>'HRPRIVS',
                   description=> 'Employee access to his/her data');
END:
-- Create a self Acl to grant EMP privileges to for an employee to see and update their own
-- Grant UPDATE, VIEW SENSITIVE INFO privileges to the EMP role.
DECLARE
ace_list XS$ACE_LIST;
BEGIN
ace list := XS$ACE LIST(
  XS$ACE TYPE(privilege list=> XS$NAME LIST('"UPDATE"', 'VIEW SENSITIVE INFO'),
               principal name=>'"F917C3608CF011E2BF802D569259982"',
principal type=>XS ACL.PTYPE EXTERNAL));
xs acl.create acl(name=> 'SELF ACL',
                   ace list=> ace list,
                   sec class=>'HRPRIVS',
                   description=> 'Employee access to his/her data');
END;
   Create Manager ACL, to allow a manager to see their employee's salary.
   Grant VIEW SENSITIVE INFO privileges to EMP role on the Manager's employees.
DECLARE
ace list XS$ACE LIST;
ace list := XS$ACE LIST(
  XS$ACE TYPE(privilege list=> XS$NAME LIST('VIEW SENSITIVE INFO'),
               principal name=>'"F917C3608CF011E2BF802D569259982"',
principal_type=>XS_ACL.PTYPE_EXTERNAL));
xs_acl.create_acl(name=> 'MGR ACL',
                   ace list=> ace list,
                   sec class=>'HRPRIVS',
                   description=> 'Manager can see his reports salaray');
END;
-- Create data security policy for the EMPLOYEE table. The policy defines
```

```
-- an instant set to control the access to the employees in department
-- 60 and 100. It also defines an attribute constraint to control
-- the access to sensitive column SALARY.
 inst sets XS$REALM CONSTRAINT LIST;
 attr secs XS$COLUMN CONSTRAINT LIST;
  inst sets :=
    XS$REALM CONSTRAINT LIST(
     XS$REALM CONSTRAINT TYPE (realm=> 'DEPARTMENT ID in (60, 100)',
                           acl list=> XS$NAME LIST('EMP ACL')));
  attr secs :=
    XS$COLUMN CONSTRAINT LIST(
      XS$COLUMN CONSTRAINT TYPE(column_list=> XS$LIST('SALARY'),
                            privilege=> 'VIEW SENSITIVE INFO'));
 xs data security.create policy(
         name=>'EMPLOYEES DS',
          realm constraint list=>inst sets,
          column_constraint_list=>attr secs);
END;
-- Add more instance sets to the above data security.
 inst1 xs$REALM CONSTRAINT TYPE;
 inst2 xs$REALM CONSTRAINT TYPE;
begin
  inst1 := xs$REALM CONSTRAINT TYPE(realm=> 'UPPER(email) = XS SYS CONTEXT(''PROFILE NS'',''EMAIL'')',
                                acl list=> XS$NAME LIST('SELF ACL'));
  xs data security.append realm constraints('EMPLOYEES DS', inst1);
  inst2 := xs$REALM CONSTRAINT TYPE(realm=> 'hr.hrutil.ismyreport(employee id) = 1',
                                acl list=> XS$NAME LIST('MGR ACL'));
 xs data security.append realm constraints('EMPLOYEES DS', inst2);
end;
/
-- Apply the data security policy on the table.
begin
 XS DATA SECURITY.apply object policy(schema=>'HR', object=>'EMPLOYEES',
                                       policy=>'EMPLOYEES DS');
end;
-- Grant more privileges for the dispatcher.
exec XS ADMIN UTIL.GRANT SYSTEM PRIVILEGE('ADMIN ANY NAMESPACE', 'ts', XS ADMIN UTIL.PTYPE XS);
grant select on sys.dba_xs_session_roles to ts_role;
EXIT;
```



## 8.8.2 About the Application Session Filter Configuration File (web.xml)

Example 8-9 shows a complete application session filter sample configuration file (web.xml) that includes the filter, its parameters, and the listener. It references a filter for setting up the namespace (MyFilter.java) shown in Example 8-11, and the sample servlet applications named MyHR.java shown in Example 8-10, in addition to: MySession.java, MyUpdate.java, and LogoutServlet.java, which are not shown.

MySession queries the V\$XS\_SESSION\_ROLES view to show the roles in the application session, queries the users in XS\$SESSION namespace to show the user in the application session, and queries the V\$XS\_SESSION\_NS\_ATTRIBUTES view to show the namespace in the application session, and then attaches to an application session.

 ${\tt MyUpdate}$  performs an update on the <code>HR.EMPLOYEES</code> table to update the phone number for an employee.

LogoutServlet performs a logout operation, and then destroys the application session at the database.

In the ApplicationSessionFilter filter configuration, the filter section references the class ApplicationSessionFilter, describes a parameter application.datasource with a parameter value jdbc/myDBDS, and describes a parameter dynamic roles with a value of HROBJ that was created in the set up script in Example 8-8.

#### **Example 8-9** A Complete Application Session Filter Sample Configuration

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"</pre>
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/
web-app_2_5.xsd"
        version="2.5" xmlns="http://java.sun.com/xml/ns/javaee">
    <filter>
        <filter-name>JpsFilter</filter-name>
        <filter-class>oracle.security.jps.ee.http.JpsFilter</filter-class>
        <init-param>
           <param-name>enable.anonymous</param-name>
           <param-value>true</param-value>
        </init-param>
        <init-param>
           <param-name>remove.anonymous.role</param-name>
            <param-value>false</param-value>
        </init-param>
        <init-param>
            <param-name>application.name
            <param-value>MyHRApp</param-value>
        </init-param>
        <!-- Following needed for Menu Security -->
        <!--init-param>
           <param-name>oracle.security.jps.jaas.mode</param-name>
            <param-value>subjectOnly</param-value>
        </init-param-->
   </filter>
   <filter>
        <filter-name>ApplicationSessionFilter</filter-name>
        <filter-class>oracle.security.xs.ee.session.ApplicationSessionFilter</filter-class>
            <param-name>application.datasource
```



```
<param-value>jdbc/myDBDS</param-value>
    </init-param>
    <init-param>
        <param-name>dynamic.roles</param-name>
        <param-value>HROBJ</param-value>
    </init-param>
    <!--
     <init-param>
        <param-name>dispatcher.pool.max</param-name>
        <param-value>90</param-value>
    </init-param>
    <!-- init-param>
        <param-name>application.id</param-name>
        <param-value>MyHRApp</param-value>
    </init-param>
    <init-param>
        <param-name>session.provider</param-name>
        <param-value>XS</param-value>
    </init-param>
    <init-param>
        <param-name>db.url</param-name>
        <param-value>jdbc:oracle:thin:@myhost:1521:orcl</param-value>
    </init-param>
    <init-param>
        <param-name>dispatcher.id</param-name>
        <param-value>ts</param-value>
    </init-param>
    <init-param>
        <param-name>dispatcher.pwd.map</param-name>
        <param-value>XS MAP</param-value>
    </init-param>
    <init-param>
        <param-name>dispatcher.pwd.key</param-name>
        <param-value>XS KEY</param-value>
    </init-param>
    <init-param>
        <param-name>dispatcher.pool.min</param-name>
        <param-value>3</param-value>
    </init-param>
    <init-param>
        <param-name>dispatcher.pool.max</param-name>
        <param-value>10</param-value>
    </init-param -->
    <!--init-param>
        <param-name>namespaces</param-name>
        <param-value>sec ns</param-value>
    </init-param-->
</filter>
<filter>
    <filter-name>MyFilter</filter-name>
    <filter-class>trusted.MyFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>JpsFilter</filter-name>
    <url-pattern>/*</url-pattern>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>REQUEST</dispatcher>
```



```
<dispatcher>INCLUDE</dispatcher>
</filter-mapping>
<filter-mapping>
    <filter-name>ApplicationSessionFilter</filter-name>
    <url-pattern>/myhr</url-pattern>
    <url-pattern>/mysession</url-pattern>
    <url-pattern>/myupdate</url-pattern>
    <url-pattern>/logout</url-pattern>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>INCLUDE</dispatcher>
</filter-mapping>
<filter-mapping>
    <filter-name>MyFilter</filter-name>
    <url-pattern>/myhr</url-pattern>
    <url-pattern>/mysession</url-pattern>
    <url-pattern>/myupdate</url-pattern>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>INCLUDE</dispatcher>
</filter-mapping>
stener>
    stener-class>oracle.security.xs.ee.session.ApplicationSessionListener
</listener>
<servlet>
    <servlet-name>MySession
    <servlet-class>app.MySession</servlet-class>
</servlet>
<servlet>
    <servlet-name>LogoutServlet</servlet-name>
    <servlet-class>app.MyLogout</servlet-class>
</servlet>
<servlet>
    <servlet-name>MyHR</servlet-name>
    <servlet-class>app.MyHR</servlet-class>
</servlet>
<servlet>
    <servlet-name>MyUpdate
    <servlet-class>app.MyUpdate</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>MySession</servlet-name>
    <url-pattern>/mysession</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>LogoutServlet</servlet-name>
    <url-pattern>/logout</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>MyHR</servlet-name>
    <url-pattern>/myhr</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>MyUpdate/servlet-name>
    <url-pattern>/myupdate</url-pattern>
</servlet-mapping>
<security-constraint>
    <web-resource-collection>
       <web-resource-name>my servlet</web-resource-name>
        <url-pattern>/myhr</url-pattern>
        <url-pattern>/mysession</url-pattern>
```

```
<url-pattern>/myupdate</url-pattern>
        </web-resource-collection>
        <auth-constraint>
            <role-name>valid-users</role-name>
        </auth-constraint>
    </security-constraint>
    <login-config>
        <auth-method>CLIENT-CERT, FORM</auth-method>
        <form-login-config>
            <form-login-page>/login.jsp</form-login-page>
            <form-error-page>/error.jsp</form-error-page>
        </form-login-config>
    </login-config>
    <security-role>
        <role-name>valid-users</role-name>
    </security-role>
</web-app>
```

## 8.8.3 About the Sample Servlet Application (MyHR.java)

Example 8-10 shows the sample servlet application named MyHR.java, which is referenced in the application session filter sample configuration (web.xml file) shown in Example 8-9.

The MyHR application performs a query on the EMPLOYEES table and returns the results. If you have authorization, depending on your login credentials, you can perform certain tasks as described in:

About the HR Demo (1) - Logged in as Employee LPOPP

As an employee, you can see your own salary information, but no one elses, and you can update only your own contact information.

About the HR Demo (2) - Logged in as HRMGR

If you are logged in as a HR Manager, you can see the salary records of all employees and you can update their contact information.

About the HR Demo (3) - Logged in as a Team Manager

If you are logged in as a Team Manager, you can see only your teams's employees salary information, but you cannot update their contact information, only your own contact information.

From a check of the privilege on the ACLs (checkPrivilege), if you have UPDATE privilege, then you are authorized to perform an update of that employee's record and the EMPLOYEE\_ID will show a link that allows you access to that employee's record.

#### Example 8-10 Sample Servlet Application MyHR.java

```
/* Copyright (c) 2009, 2014, Oracle and/or its affiliates.
All rights reserved.*/
package app;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
```



```
import java.util.ArrayList;
import java.util.Collection;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.servlet.ServletConfig;
import javax.servlet.http.HttpServlet;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;
import oracle.jdbc.OracleResultSet;
import oracle.jdbc.OracleResultSet.AuthorizationIndicator;
import oracle.security.xs.ee.session.ApplicationSessionException;
import oracle.security.xs.ee.session.ApplicationSessionService;
public class MyHR extends HttpServlet {
   private static final String CONTENT TYPE = "text/html; charset=UTF-8";
    String query = " select emp.EMPLOYEE ID, emp.first name, emp.last name, " +
                            emp.email, emp.phone_number, salary, emp.manager id, " +
                            emp.department id,ora get aclids(emp) as acl id" +
                   " from hr.employees emp";
   public void init(ServletConfig config) throws ServletException {
        super.init(config);
    }
    public void queryHR(PrintWriter out) throws ApplicationSessionException {
        DataSource dataSource = null;
        Connection conn = null;
        try {
            InitialContext ic;
            try {
                ic = new InitialContext();
                dataSource = (DataSource)ic.lookup("jdbc/myDBDS");
                if (dataSource != null)
                        conn = dataSource.getConnection();
                    } catch (SQLException e) {
                        e.printStackTrace();
            } catch (NamingException e) {
                e.printStackTrace();
            try {
                queryHR(conn, out);
            } catch (Exception e) {
                e.printStackTrace();
        } finally {
```



```
if (conn != null)
            try {
                conn.close();
            } catch (SQLException e) {
}
public void doGet(HttpServletRequest request,
                  HttpServletResponse response) throws ServletException,
                                                       IOException {
    response.setContentType(CONTENT TYPE);
    PrintWriter pw = response.getWriter();
    pw.println(HEADER);
    pw.println("<h1><font size=\"+2\">RAS Session Service Demo</font></h1>");
    pw.println("<font size=\"+1\">");
    pw.println("You are logged in as <b>" + request.getRemoteUser() + "</b>");
    try {
        queryHR(pw);
    } catch (ApplicationSessionException e) {
        e.printStackTrace();
    pw.println(FOOTER);
    pw.close();
}
public Collection<Employee> queryHR(Connection conn ) {
  Statement stmt = null;
  ResultSet rs = null;
  Collection<Employee> result = new ArrayList<Employee>();
  try {
    // attach session
    ApplicationSessionService.attachSession(conn);
    stmt = conn.createStatement();
    rs = stmt.executeQuery(query);
    while (rs.next()) {
      Employee emp = new Employee();
      emp.setId(rs.getString("EMPLOYEE ID"));
      AuthorizationIndicator ai =
                ((OracleResultSet)rs).getAuthorizationIndicator("salary");
      if (ai == AuthorizationIndicator.NONE) {
        emp.setSalary(rs.getString("salary"));
      } else {
        emp.setSalary("*****") ;
      // get ACL associated with the row
```



```
emp.setAcl(rs.getBytes("acl id"));
         // check "update" privilege
         boolean canUpdate = ApplicationSessionService.checkPrivilege(conn, emp.getAcl(),
"UPDATE");
         emp.setUpdate(canUpdate);
         result.add(emp);
         emp.setFname(rs.getString("first name"));
         emp.setLname(rs.getString("last name"));
         emp.setEmail(rs.getString("email"));
         emp.setPhone(rs.getString("phone number"));
         emp.setManagerId(rs.getString("manager id"));
         emp.setDepId(rs.getString("department id"));
     } catch (ApplicationSessionException e) {
         e.printStackTrace();
         // process me
     } catch (SQLException e) {
         // process me
         e.printStackTrace();
     } finally {
        if (stmt != null) try {stmt.close();} catch (SQLException e) {};
        if (rs != null) try { rs.close();} catch (SQLException e) {};
        try {ApplicationSessionService.detachSession(conn);} catch (ApplicationSessionException
e) {};
     }
     return result;
   public void queryHR(Connection conn, PrintWriter out ) {
       Collection < Employee > list = queryHR (conn);
       PrintWriter pw = out;
       pw.println("<br>br>Displaying employee record(s) that you can access.<br>");
       pw.println("</font>");
       pw.println("<i>NOTE: Salary is only shown if you are authorized to view,
 and ID is shown as a link if you are authorized to perform an update.</i>
       out.println("");
       String tmp;
       if (list.size() > 0) {
           out.println("");
           out.println("ID");
           out.println(">First Name");
           out.println("Last Name");
           out.println("Email");
           out.println("Phone");
           out.println(">Salary");
           out.println("Department ID");
           out.println("Manager ID");
           out.println("");
       }
```

```
for (Employee e: list) {
       if (e.canUpdate()) {
          tmp = "<a href=\"update.jsp?id=" + e.getId() + "\">" + e.getId() + "</a>";
       } else {
           tmp = e.getId();
       out.println("" + tmp + "");
       out.println("" + e.getFname() + "");
       out.println("" + e.getLname() + "");
       out.println("" + e.getEmail() + "");
       out.println("" + e.getPhone() + "");
       out.println("" + e.getSalary() + "");
       out.println("" + e.getDepId() + "");
       out.println("" + e.getManagerId() + "");
   out.println("</TABLE>");
} ;
class Employee {
   String id;
   String salary;
   boolean update;
   String fname;
   String lname;
   String email;
   String phone;
   String managerId;
   String depId;
   byte[] acl;
   public void setId(String id) {
       this.id = id;
   public String getId() {
      return id;
   public void setSalary(String salary) {
       this.salary = salary;
   public String getSalary() {
       return salary;
   public void setUpdate(boolean canUpdate) {
       this.update = canUpdate;
   public boolean canUpdate() {
       return update;
```



```
public void setFname(String fname) {
        this.fname = fname;
    public String getFname() {
        return fname;
    public void setLname(String lname) {
        this.lname = lname;
    public String getLname() {
        return lname;
    public void setEmail(String email) {
        this.email = email;
    public String getEmail() {
       return email;
    public void setPhone(String phone) {
        this.phone = phone;
    public String getPhone() {
        return phone;
    public void setManagerId(String managerId) {
        this.managerId = managerId;
    public String getManagerId() {
       return managerId;
    public void setDepId(String depId) {
       this.depId = depId;
    public String getDepId() {
       return depId;
    public void setAcl(byte[] acl) {
       this.acl = acl;
    public byte[] getAcl() {
       return acl;
private static String HEADER = "<html xmlns=\"http://www.w3.org/1999/xhtml\"><head>"
       + "<meta content=\"text/html; charset=UTF-8\" http-equiv=\"content-type\"/>"
        + "<title>Oracle</title>"
        + "<link href=\"css/general.css\" type=\"text/css\" rel=\"stylesheet\"/>"
```

```
+ "<link href=\"css/window.css\" type=\"text/css\" rel=\"stylesheet\"/>"
    + "<link href=\"css/login.css\" type=\"text/css\" rel=\"stylesheet\"/>"
    + "<script type=\"text/javascript\">"
    + " if (top != self) top.location.href = location.href;"
    + "</script>"
    + "<style type=\"text/css\">"
    + "html { background-color: #001C34;}"
    + "</style>"
    + "</head>"
    + "<body onload=\"document.loginData.j_username.focus();\">"
    + " <div id=\"top\">"
        <div id=\"login-header\">"
          <div id=\"login-logo\">"
    + " <img src=\"images/logo.png\"/>"
    + "</div>"
    + " </div>"
    + " <div id=\"content\">"
    + "<div id=\"app data\"><div id=\"title\"></div>";
private static String FOOTER = "<a href=\"/myapp/logout\">Logout</a>"
        + "</div></div></div></div></div></div></body></html>";
```

## 8.8.4 About the Filter to Set Up the Application Namespace (MyFilter.java)

Example 8-11 shows a filter to set up the application namespace. This filter is named MyFilter.java, which is referenced in the application session filter sample configuration (web.xml file) shown in Example 8-9.

This filter should be deployed as a separate jar, and SessionCodePermission should be granted to the jar file.

This filter first queries the V\$XS\_SESSION\_ROLES view to show the roles in the Real Security Application session. Next, this filter demonstrates how trusted application code (a filter) firsts checks to see if a namespace exists (getNamespaceAttribute); then if not, it can set up a security critical namespace using session privilege elevation (attachSessionPrivileged) and namespace APIs (createNamespace, and setNamespaceAttribute) to create the namespace and set some namespace attributes.

#### Example 8-11 Filter to Set Up Application Namespace

```
/* Copyright (c) 2009, 2014, Oracle and/or its affiliates. All rights reserved.*/

package trusted;

import java.io.IOException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import javax.servlet.Filter;
import javax.servlet.Filterchain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
```



```
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.sql.DataSource;
import oracle.security.xs.ee.session.ApplicationSessionException;
import oracle.security.xs.ee.session.ApplicationSessionService;
import oracle.security.xs.ee.session.NamespaceNotFoundException;
/**
 ^{\star} Demonstrate how trusted application code (a filter) can set up
 * security critical namespace using session privilege elevation and
 * namespace APIs.
 * The filter should be deployed as a separate jar, and SessionCodePermission
 * should be granted to the jar.
public class MyFilter implements Filter {
   private FilterConfig filterConfig = null;
   DataSource myDatasource = null;
   public void init(FilterConfig filterConfig) throws ServletException {
       filterConfig = filterConfig;
   public void destroy() {
       filterConfig = null;
   public void querySessionRoles(Connection conn) throws SQLException {
       String query =
           "select role name from v$xs session roles order by role name";
       String roles = null;
       try {
           Statement stmt = conn.createStatement();
           ResultSet rs = stmt.executeQuery(query);
           System.out.println("<TABLE>");
           while (rs.next()) {
               roles = rs.getString(1);
               System.out.println("" + roles + "");
           System.out.println("</TABLE>");
       } finally {
       return;
```

private boolean namespaceExists(String ns, String attribute, String value) throws

```
ApplicationSessionException {
        try {
            return value.equalsIgnoreCase(ApplicationSessionService.getNamespaceAttribute(ns,
attribute));
        } catch (NamespaceNotFoundException e) {
            return false;
   private Connection getConnection() {
        DataSource dataSource = null;
        InitialContext ic;
        try {
           ic = new InitialContext(); //TODO cache context
            dataSource = (DataSource)ic.lookup("jdbc/myDBDS");
            if (dataSource != null)
                   return dataSource.getConnection();
                } catch (SQLException e) {
                   e.printStackTrace();
        } catch (NamingException e) {
            e.printStackTrace();
       return null;
    public void doFilter(ServletRequest request, ServletResponse response,
                         FilterChain chain) {
        Connection conn = null;
       try {
            String email = ((HttpServletRequest)request).getRemoteUser();
            if ( email != null && !namespaceExists("PROFILE NS", "EMAIL", email )) {
                conn = getConnection();
                //AccessController.doPrivileged(new AttachAction(conn), null);
                ApplicationSessionService.attachSessionPrivileged(conn, "SESSION_NS_DROLE");
                ApplicationSessionService.createNamespace(conn, "PROFILE NS");
                ApplicationSessionService.setNamespaceAttribute(conn, "PROFILE_NS", "EMAIL", email);
                ApplicationSessionService.detachSession(conn);
        } catch (ApplicationSessionException e) {
            e.printStackTrace();
        } catch (Exception e) {
```



```
e.printStackTrace();
} finally {

    if (conn != null)
        try {
            conn.close();
        } catch (SQLException e) {
        }

    try {
        chain.doFilter(request, response);
} catch (IOException e) {
        e.printStackTrace();
} catch (ServletException e) {
        e.printStackTrace();
}
```

### 8.8.5 About the HR Demo Use Case - User Roles

In the HR Demo Use Case, the Identity Management store contains the user name, user name's password, and group name, while the OPSS security store contains the application roles and the user and group to application roles mapping. Example 8-12 shows a code snippet for the user and group to application roles mapping for one user LPOPP.

#### **Example 8-12 User and Group to Application Roles Mapping**

## 8.8.6 About the HR Demo (1) - Logged in as Employee LPOPP

Table 8-1 displays employee records that you can access logged in as an employee, LPOPP. You can see everyone's record except their salary information, you can see your own salary information, and you can update your own contact information.

This access is set by:

- Realm and grant (1): DEPARTMENT ID in (60, 100) and SELECT to EMP
- Realm and grant (2): UPPER (email) = XS\_SYS\_CONTEXT ("PROFILE\_NS", "EMAIL")
  and UPDATE, VIEW SENSITIVE INFO to EMP

Column Constraints: SALARY requires VIEW SENSITIVE INFO privilege

Salary is only shown if you are authorized to view, and ID is shown as a link (Italic format in the table) if you are authorized to update.

Table 8-1 Session Service HR Demo(1) Logged in as Employee LPOPP

ID	First Name	Last Name	Email	Phone	Salary	Department ID	Manager ID
103	Alexander	Hunold	AHUNOLD	510.222.3388	*****	60	102
104	Bruce	Ernst	BERNST	590.423.4568	*****	60	103
105	David	Austin	DAUSTIN	590.423.4569	*****	60	103
106	Valli	Pataballa	VPATABAL	590.423.4560	*****	60	103
107	Diana	Lorentz	DLORENTZ	590.423.4567	*****	60	103
108	Nancy	Greenberg	NGREENBE	515.124.4569	*****	100	101
109	Daniel	Faviet	DFAVIET	515.124.4169	*****	100	108
110	John	Chen	JCHEN	515.124.4269	*****	100	108
111	Ismael	Sciarra	ISCIARRA	515.124.4369	*****	100	108
112	Jose Manuel	Urman	JMURMAN	515.124.4469	*****	100	108
113	Luis	Popp	LPOOP	133.444.5555	6900	100	108

## 8.8.7 About the HR Demo (2) - Logged in as HRMGR

Table 8-2 displays employee records that you can access logged in as an HR Manager, HRMGR. You can see every employee's salary information, and you can update every employee's contact information.

This access is set by the realm and grant: DEPARTMENT\_ID in (60, 100), SELECT, UPDATE, and VIEW SENSITIVE INFO to HRMGR.

Salary is only shown if you are authorized to view, and ID is shown as a link (Italic format in the table) if you are authorized to update.

Table 8-2 Session Service HR Demo(2) Logged in as HR Manager HRMGR

ID	First Name	Last Name	Email	Phone	Salary	Department ID	Manager ID
103	Alexander	Hunold	AHUNOLD	510.222.3388	9000	60	102
104	Bruce	Ernst	BERNST	590.423.4568	6000	60	103
105	David	Austin	DAUSTIN	590.423.4569	4800	60	103
106	Valli	Pataballa	VPATABAL	590.423.4560	4800	60	103
107	Diana	Lorentz	DLORENTZ	590.423.4567	4200	60	103
108	Nancy	Greenberg	NGREENBE	515.124.4569	12008	100	101
109	Daniel	Faviet	DFAVIET	515.124.4169	9000	100	108
110	John	Chen	JCHEN	515.124.4269	8200	100	108
111	Ismael	Sciarra	ISCIARRA	515.124.4369	7700	100	108



Table 8-2 (Cont.) Session Service HR Demo(2) Logged in as HR Manager HRMGR

ID	First Name	Last Name	Email	Phone	Salary	Department ID	Manager ID
112	Jose Manuel	Urman	JMURMAN	515.124.4469	7800	100	108
113	Luis	Рорр	LPOOP	133.444.5555	6900	100	108

### 8.8.8 About the HR Demo (3) - Logged in as a Team Manager

Table 8-3 displays employee records that you can access logged in as a Team Manager, AHUNOLD. You can see your team member's salary information; however, you cannot update their contact information, only your own contact information.

This access is set by the realm and grant: is my member(employee\_id) =1 and VIEW SENSITIVE INFO to EMP.

Salary is only shown if you are authorized to view, and ID is shown as a link (Italic format in the table) if you are authorized to update.

Table 8-3 Session Service HR Demo(3) Logged in as Team Manager AHUNOLD

ID	First Name	Last Name	Email	Phone	Salary	Department ID	Manager ID
103	Alexander	Hunold	AHUNOLD	510.222.3388	9000	60	102
104	Bruce	Ernst	BERNST	590.423.4568	6000	60	103
105	David	Austin	DAUSTIN	590.423.4569	4800	60	103
106	Valli	Pataballa	VPATABAL	590.423.4560	4800	60	103
107	Diana	Lorentz	DLORENTZ	590.423.4567	4200	60	103
108	Nancy	Greenberg	NGREENBE	515.124.4569	*****	100	101
109	Daniel	Faviet	DFAVIET	515.124.4169	*****	100	108
110	John	Chen	JCHEN	515.124.4269	*****	100	108
111	Ismael	Sciarra	ISCIARRA	515.124.4369	*****	100	108
112	Jose Manuel	Urman	JMURMAN	515.124.4469	*****	100	108
113	Luis	Рорр	LPOOP	133.444.5555	*****	100	108



9

# Oracle Database Real Application Security Data Dictionary Views

This chapter describes the data dictionary views provided with Oracle Database Real Application Security.

Table 9-1 summarizes these views. For additional data dictionary views related to Oracle Real Application Security, see *Oracle Database Reference*.

Table 9-1 Oracle Database Real Application Security Data Dictionary Views

Data Dictionary View	Summary Description
DBA_XS_OBJECTS	Displays all Real Application Security objects
DBA_XS_PRINCIPALS	Displays all application users and application roles
DBA_XS_EXTERNAL_PRINCIPALS	Displays all external application users and application roles
DBA_XS_USERS	Displays all application users
USER_XS_USERS	Displays the application users own account information
USER_XS_PASSWORD_LIMITS	Displays password limits for the currently logged on application user
DBA_XS_ROLES	Displays all application roles
DBA_XS_DYNAMIC_ROLES	Displays all dynamic application roles
DBA_XS_PROXY_ROLES	Displays all proxy application roles
DBA_XS_ROLE_GRANTS	Displays all Real Application Security application role grants
DBA_XS_PRIVILEGES	Lists all Real Application Security application privileges defined in the database.
USER_XS_PRIVILEGES	Lists application privileges contained in security classes owned by the current user
DBA_XS_IMPLIED_PRIVILEGES	Lists all the Real Application Security implied application privileges defined in the database
USER_XS_IMPLIED_PRIVILEGES	Lists all the implied application privileges contained in security classes owned by the current user
DBA_XS_SECURITY_CLASSES	Lists all security classes defined in the database
USER_XS_SECURITY_CLASSES	Lists all security classes owned by the current application user

Table 9-1 (Cont.) Oracle Database Real Application Security Data Dictionary Views

Data Dictionary View	<b>Summary Description</b>
DBA_XS_SECURITY_CLASS_DEP	Lists the dependencies between security classes.
USER_XS_SECURITY_CLASS_DEP	Lists the parent security classes for the dependent security classes owned by the current user.
DBA_XS_ACLS	Lists all existing ACLs
USER_XS_ACLS	lists all ACLs owned by the current user
DBA_XS_ACES	Lists all the Access Control Entries (ACEs)
USER_XS_ACES	Lists all the ACEs from the ACLs owned by the current user
DBA_XS_POLICIES	Lists all the data security policies
USER_XS_POLICIES	Lists all the data security policies owned by the current application user
DBA_XS_REALM_CONSTRAINTS	Lists all Real Application Security realms
USER_XS_REALM_CONSTRAINTS	Lists all Real Application Security realms owned by the current user
DBA_XS_INHERITED_REALMS	Lists all Real Application Security inherited realms
USER_XS_INHERITED_REALMS	Lists all Real Application Security inherited realms owned by the current user
DBA_XS_ACL_PARAMETERS	Lists all Real Application Security ACL parameters
USER_XS_ACL_PARAMETERS	Lists all Real Application Security ACL parameters defined in data security policies owned by the current user
DBA_XS_COLUMN_CONSTRAINTS	Lists all Real Application Security column constraints
USER_XS_COLUMN_CONSTRAINTS	Lists all Real Application Security column constraints owned by the current user
DBA_XS_APPLIED_POLICIES	Displays all database objects on which Real Application Security data security policies are enabled
DBA_XS_MODIFIED_POLICIES	Displays all database objects on which Real Application Security data security policies are modified
DBA_XS_SESSIONS	Lists all application sessions in the database
DBA_XS_ACTIVE_SESSIONS	Lists all attached application sessions in the database
DBA_XS_SESSION_ROLES	Lists application roles enabled in application sessions



Table 9-1 (Cont.) Oracle Database Real Application Security Data Dictionary Views

Data Dictionary View	Summary Description
DBA_XS_SESSION_NS_ATTRIBUTES	Displays namespace attributes across application sessions as of last saved state
DBA_XS_NS_TEMPLATES	Describes all Real Application Security namespace templates
DBA_XS_NS_TEMPLATE_ATTRIBUTES	Describes all namespace templates together with their attribute details
ALL_XDS_ACL_REFRESH	Displays all static ACL refresh settings for tables that are accessible to the application user.
ALL_XDS_ACL_REFSTAT	Displays all static ACL refresh job status history that has been done for tables accessible to the application user.
ALL_XDS_LATEST_ACL_REFSTAT	Displays the ACL refreshjob status for the most recent refreshment job for each table accessible to the application user.
DBA_XDS_ACL_REFRESH	Displays all static ACL refresh settings in the database.
DBA_XDS_ACL_REFSTAT	Displays all static ACL refresh job status history that has been done in the database
DBA_XDS_LATEST_ACL_REFSTAT	Displays the ACL refresh job status for the most recent refreshment job for each table in the database
USER_XDS_ACL_REFRESH	Displays all static ACL refresh settings for tables that are owned by the user.
USER_XDS_ACL_REFSTAT	Displays all static ACL refresh job status history that has been done for tables owned by the user.
USER_XDS_LATEST_ACL_REFSTAT	Displays the ACL refresh job status for the most recent refreshment job for each table owned by the user.
V\$XS_SESSION_NS_ATTRIBUTES	Displays information about the namespaces and attributes in the current application session.
V\$XS_SESSION_ROLES	Displays all enabled application roles in the current application session.
DBA_XS_AUDIT_POLICY_OPTIONS	Describes the auditing options that were defined for Real Application Security unified audit policies. See <i>Oracle Database Reference</i> for more information. For information about unified auditing in an Oracle Database Real Application Security environment, see <i>Oracle Database Security Guide</i> .



Table 9-1 (Cont.) Oracle Database Real Application Security Data Dictionary Views

Data Dictionary View	Summary Description
DBA_XS_AUDIT_TRAIL	Provides detailed information about Real Application Security that were audited. See <i>Oracle Database Reference</i> for more information. For information about unified auditing in an Oracle Database Real Application Security environment, see <i>Oracle Database Security Guide</i> .
DBA_XS_ENB_AUDIT_POLICIES	Lists users for whom Real Application Security unified audit polices are enabled. See <i>Oracle Database Reference</i> for more information. For information about unified auditing in an Oracle Database Real Application Security environment, see <i>Oracle Database Security Guide</i> .

This section describes the following Oracle Database Real Application Security data dictionary views:

# 9.1 DBA\_XS\_OBJECTS

The  $\mbox{DBA}_{\mbox{XS}}\mbox{OBJECTS}$  data dictionary view lists all the existing Real Application Security objects in the database.

- DBA\_XS\_PRINCIPALS
- DBA\_XS\_SECURITY\_CLASSES
- DBA\_XS\_ACLS
- DBA\_XS\_POLICIES
- DBA\_XS\_NS\_TEMPLATES

Column	Datatype	NULL	Description	
NAME	VARCHAR2 (128)		Name of the object	
OWNER	VARCHAR2 (128)		Owner of the object	
ID	NUMBER	NOT NULL	Identifier number for the object	
TYPE	VARCHAR2(18)		Type of the object. Possible values are:  PRINCIPAL (Application User/Application Role)  SECURITY CLASS  ACL	
			<ul><li>PRIVILEGE</li><li>DATA SECURITY (Policy)</li><li>NAMESPACE TEMPLATE</li></ul>	



Column	Datatype	NULL	Description
STATUS	VARCHAR2(8)		Status of the object. Possible values are:
			• INVALID
			• VALID
			• EXTERNAL

# 9.2 DBA\_XS\_PRINCIPALS

The DBA\_XS\_PRINCIPALS data dictionary view describes all the existing application users and application roles in the database.

#### **Related Views**

- DBA\_XS\_USERS
- DBA\_XS\_ROLES
- DBA\_XS\_DYNAMIC\_ROLES
- DBA\_XS\_PROXY\_ROLES
- DBA\_XS\_EXTERNAL\_PRINCIPALS

Column	Datatype	NULL	Description
NAME	VARCHAR2 (128)		Name of the principal (application user or application role)
GUID	RAW(16)		Globally unique identifier for the principal
TYPE	VARCHAR2 (12)		Type of the principal. Possible values are:  USER ROLE DYNAMIC ROLE
EXTERNAL_SOURCE	VARCHAR2 (128)		External source of the principal
DESCRIPTION	VARCHAR2 (4000)		Description of the principal

# 9.3 DBA\_XS\_EXTERNAL\_PRINCIPALS

The  $\mbox{DBA}_{XS}_{EXTERNAL}_{PRINCIPALS}$  data dictionary view lists all the external application users and application roles.

- DBA\_XS\_PRINCIPALS
- DBA XS USERS
- DBA\_XS\_ROLES
- DBA\_XS\_DYNAMIC\_ROLES
- DBA\_XS\_PROXY\_ROLES



Column	Datatype	NULL	Description
NAME	VARCHAR2(128)		Name of the external principal

# 9.4 DBA\_XS\_USERS

The  $\mbox{DBA}_{\mbox{\scriptsize LXS}}$   $\mbox{\tiny USERS}$  data dictionary view describes all existing application users defined in the database.

- DBA\_XS\_PRINCIPALS
- USER\_XS\_USERS
- DBA\_XS\_ROLES
- DBA\_XS\_DYNAMIC\_ROLES
- DBA\_XS\_PROXY\_ROLES

Column	Datatype	NULL	Description
NAME	VARCHAR2 (128)		Name of the application user
GUID	RAW (16)		Globally unique identifier for the application user
EXTERNAL_SOURCE	VARCHAR2 (128)		External Source of application users, such as LDAP
ROLES_DEFAULT_ENABLED	VARCHAR2(3)		Indicates whether all the application roles granted to the application user are enabled by default. Valid values are YES and NO.
STATUS	VARCHAR2(8)		Status of the application user. Valid values are ACTIVE and INACTIVE.
ACCOUNT_STATUS	VARCHAR2(32)	NOT NULL	Direct login password policy account status of the user. Indicates whether the account is locked, expired, or unlocked.
LOCK_DATE	DATE		The date the account became locked for the direct login user
EXPIRY_DATE	DATE		The date the passward became expired for the direct login user
PROFILE	VARCHAR2(128)		The name of the database profile associated with the application user
SCHEMA	VARCHAR2 (128)		Application user schema
START_DATE	TIMESTAMP(6) WITH TIME ZONE		Effective start date for the user



Column	Datatype	NULL	Description
END_DATE	TIMESTAMP(6) WITH TIME ZONE		Effective end date for the user
DIRECT_LOGON_USER	VARCHAR2(3)		Indicates whether this user has direct logon capability
VERIFIER_TYPE	VARCHAR2(11)		Type of the verifier assigned to the direct logon user. Only XS_SHA512 and XS_SALTED_SHA1 are allowed.)
ACL	VARCHAR2(128)		The Real Application Security session privilege.
DESCRIPTION	VARCHAR2(4000)		Description of the application user

# 9.5 USER\_XS\_USERS

The  ${\tt USER\_XS\_USERS}$  data dictionary view describes the current application users own account information.

- DBA\_XS\_USERS
- DBA\_XS\_PRINCIPALS
- DBA\_XS\_ROLES
- DBA\_XS\_DYNAMIC\_ROLES
- DBA\_XS\_PROXY\_ROLES

Column	Datatype	NULL	Description
NAME	VARCHAR2 (128)		Name of the current application user
STATUS	VARCHAR2(8)		Status of the current application user. Valid values are ACTIVE and INACTIVE only.
ACCOUNT_STATUS	VARCHAR2(32)	NOT NULL	Direct login password policy account status of the current user. Valid values are UNLOCK, LOCKED, and EXPIRED. UNLOCK means the current user's account is open.
LOCK_DATE	DATE		The date the account became locked for the direct login session for the current user
EXPIRY_DATE	DATE		The date the passward became expired or the direct login session for the current user
DIRECT_LOGON_USER	VARCHAR2(3)		Indicates whether this user has direct logon capability



Column	Datatype	NULL	Description
DESCRIPTION	VARCHAR2 (4000)		Description of the application user

# 9.6 USER\_XS\_PASSWORD\_LIMITS

The <code>USER\_XS\_PASSWORD\_LIMITS</code> data dictionary view describes the password limits for the currently logged on application user. The DBA can query this view to check the limits for any direct login user.

#### **Related Views**

Column	Datatype	NULL	Description
RESOURCE_NAME	VARCHAR2(32)	NOT NULL	Name of the password resource
LIMIT	VARCHAR2(128)		The limit placed on this resource

# 9.7 DBA\_XS\_ROLES

The  $\mbox{DBA}_{\mbox{XS}}\mbox{ROLES}$  data dictionary view describes all existing application roles in the database.

- DBA\_XS\_PRINCIPALS
- DBA\_XS\_USERS
- DBA\_XS\_DYNAMIC\_ROLES
- DBA\_XS\_PROXY\_ROLES
- DBA\_XS\_ROLE\_GRANTS

Column	Datatype	NULL	Description
NAME	VARCHAR2 (128)		Name of the application role
GUID	RAW(16)		Globally unique identifier for the application role
EXTERNAL_SOURCE	VARCHAR2(128)		External Source of the application role, such as LDAP
DEFAULT_ENABLED	VARCHAR(3)		Whether or not the application role is enabled by default. Values can be YES or NO.
START_DATE	TIMESTAMP(6) WITH TIME ZONE		Start date from which the application role is valid
END_DATE	TIMESTAMP(6) WITH TIME ZONE		End date until which the application role is valid
DESCRIPTION	VARCHAR2 (4000)		Description of the application role



# 9.8 DBA\_XS\_DYNAMIC\_ROLES

The  $\DBA_XS_DYNAMIC_ROLES$  data dictionary view describes all existing dynamic application roles in the database.

#### **Related Views**

- DBA\_XS\_PRINCIPALS
- DBA\_XS\_USERS
- DBA\_XS\_ROLES
- DBA\_XS\_ROLE\_GRANTS

Column	Datatype	NULL	Description
NAME	VARCHAR2 (128)		Name of the dynamic application role
GUID	RAW(16)		Globally unique identifier for the dynamic application role
DURATION	NUMBER		Duration (in minutes) for which the role has been active
SYSTEM_DEFINED	VARCHAR2(3)		Indicates whether the application role is a system-defined role. Possible values are YES and NO.
SCOPE	VARCHAR2(7)		Scope of the application role. Possible values are SESSION and REQUEST.
ACL	VARCHAR2 (128)		The Real Application Security session privilege.
DESCRIPTION	VARCHAR2 (4000)		Description of the dynamic application role.

# 9.9 DBA\_XS\_PROXY\_ROLES

The DBA\_XS\_PROXY\_ROLES data dictionary view describes all Real Application Security proxy application role grants.

- DBA\_XS\_PRINCIPALS
- DBA\_XS\_USERS
- DBA\_XS\_ROLES
- DBA\_XS\_DYNAMIC\_ROLES
- DBA\_XS\_ROLE\_GRANTS

Column	Datatype	NULL	Description
PROXY_USER	VARCHAR2 (128)		Name of the proxy application user
TARGET_USER	VARCHAR2 (128)		Name of the target application user
TARGET_ROLE	VARCHAR2 (128)		Name of the target application role



# 9.10 DBA\_XS\_ROLE\_GRANTS

The DBA\_XS\_ROLE\_GRANTS data dictionary view describes all Real Application Security application role grants.

#### **Related Views**

- DBA\_XS\_PRINCIPALS
- DBA\_XS\_USERS
- DBA\_XS\_ROLES
- DBA\_XS\_DYNAMIC\_ROLES
- DBA\_XS\_PROXY\_ROLES

Column	Datatype	NULL	Description
GRANTEE	VARCHAR2 (128)		Name of the principal to which the application role is granted
GRANTED_ROLE	VARCHAR2 (128)		Name of the granted application role
GRANTED_ROLE_TYPE	VARCHAR2(11)		Name of the granted role
START_DATE	TIMESTAMP(6) WITH TIME ZONE		Start date from which the application role grant is valid
END_DATE	TIMESTAMP(6) WITH TIME ZONE		End date until which the application role grant is valid

# 9.11 DBA\_XS\_PRIVILEGES

The  $\mbox{DBA}_{\mbox{XS}_{\mbox{PRIVILEGES}}}$  data dictionary view lists all the Real Application Security application privileges defined in the database.

- USER\_XS\_PRIVILEGES
- DBA\_XS\_IMPLIED\_PRIVILEGES
- USER\_XS\_IMPLIED\_PRIVILEGES
- DBA\_XS\_SECURITY\_CLASSES
- ALL\_XS\_PRIVILEGES

Column	Datatype	NULL	Description
NAME	VARCHAR2 (128)		Name of the application privilege
SECURITY_CLASS	VARCHAR2 (128)		Name of the security class that contains the application privilege
SECURITY_CLASS_OWNER	VARCHAR2 (128)		Owner of the security class that contains the application privilege
DESCRIPTION	VARCHAR2(4000)		Description of the application privilege.



# 9.12 USER\_XS\_PRIVILEGES

The USER\_XS\_PRIVILEGES data dictionary view lists the application privileges contained in security classes owned by the current user.

#### **Related Views**

- DBA\_XS\_PRIVILEGES
- DBA\_XS\_IMPLIED\_PRIVILEGES
- USER\_XS\_IMPLIED\_PRIVILEGES
- USER\_XS\_SECURITY\_CLASSES
- ALL\_XS\_PRIVILEGES
- ALL\_XS\_IMPLIED\_PRIVILEGES

Column	Datatype	NULL	Description
NAME	VARCHAR2(128)		Name of the application privilege
SECURITY_CLASS	VARCHAR2 (128)		Name of the security class that contains the application privilege
DESCRIPTION	VARCHAR2 (4000)		Description of the application privilege.

# 9.13 ALL\_XS\_PRIVILEGES

The ALL\_XS\_PRIVILEGES data dictionary view lists all the Real Application Security application privileges scoped by the security classes accessible to the current user.

- USER\_XS\_PRIVILEGES
- DBA\_XS\_IMPLIED\_PRIVILEGES
- USER\_XS\_IMPLIED\_PRIVILEGES
- DBA XS SECURITY CLASSES

Column	Datatype	NULL	Description
NAME	VARCHAR2(128)		Name of the application privilege
SECURITY_CLASS	VARCHAR2 (128)		Name of the security class that contains the application privilege
SECURITY_CLASS_OWNER	VARCHAR2(128)		Owner of the security class that contains the application privilege
DESCRIPTION	VARCHAR2 (4000)		Description of the application privilege.



### 9.14 DBA\_XS\_IMPLIED\_PRIVILEGES

The DBA\_XS\_IMPLIED\_PRIVILEGES data dictionary view lists all the Real Application Security implied application privileges defined in the database.

#### **Related Views**

- DBA\_XS\_PRIVILEGES
- USER\_XS\_PRIVILEGES
- USER\_XS\_IMPLIED\_PRIVILEGES
- DBA\_XS\_SECURITY\_CLASSES
- ALL\_XS\_PRIVILEGES
- ALL\_XS\_IMPLIED\_PRIVILEGES
- ALL\_XS\_SECURITY\_CLASSES

Column	Datatype	NULL	Description
PRIVILEGE	VARCHAR2(128)		Name of the application privilege containing the implied application privilege
IMPLIED_PRIVILEGE	VARCHAR2 (128)		Name of the implied application privilege
SECURITY_CLASS	VARCHAR2 (128)		Name of the security class that contains the application privilege
SECURITY_CLASS_OWNER	VARCHAR2 (128)		Owner of the security class that contains the application privilege

### 9.15 USER\_XS\_IMPLIED\_PRIVILEGES

The  ${\tt USER\_XS\_IMPLIED\_PRIVILEGES}$  data dictionary view lists the implied application privileges contained in security classes owned by the current user.

- DBA\_XS\_PRIVILEGES
- USER\_XS\_PRIVILEGES
- DBA\_XS\_IMPLIED\_PRIVILEGES
- USER\_XS\_SECURITY\_CLASSES
- ALL\_XS\_PRIVILEGES
- ALL\_XS\_IMPLIED\_PRIVILEGES
- ALL\_XS\_SECURITY\_CLASSES



Column	Datatype	NULL	Description
PRIVILEGE	VARCHAR2(128)		Name of the application privilege containing the implied application privilege
IMPLIED_PRIVILEGE	VARCHAR2 (128)		Name of the implied application privilege
SECURITY_CLASS	VARCHAR2 (128)		Name of the security class that contains the application privilege

# 9.16 ALL\_XS\_IMPLIED\_PRIVILEGES

The ALL\_XS\_IMPLIED\_PRIVILEGES data dictionary view lists all the Real Application Security implied application privileges scoped by the security classes accessible to the current user.

#### **Related Views**

- DBA\_XS\_PRIVILEGES
- USER\_XS\_PRIVILEGES
- USER\_XS\_IMPLIED\_PRIVILEGES
- DBA\_XS\_SECURITY\_CLASSES

Column	Datatype	NULL	Description
PRIVILEGE	VARCHAR2 (128)		Name of the application privilege containing the implied application privilege
IMPLIED_PRIVILEGE	VARCHAR2 (128)		Name of the implied application privilege
SECURITY_CLASS	VARCHAR2 (128)		Name of the security class that contains the application privilege
SECURITY_CLASS_OWNER	VARCHAR2 (128)		Owner of the security class that contains the application privilege

### 9.17 DBA\_XS\_PRIVILEGE\_GRANTS

The DBA\_XS\_PRIVILEGE\_GRANTS data dictionary view lists all the Real Application Security system or schema level privilege grants defined in the database.

- USER\_XS\_PRIVILEGES
- DBA\_XS\_PRIVILEGES

Column	Datatype	NULL	Description
PRIVILEGE	VARCHAR2 (128)		Name of the application privilege
GRANTEE	VARCHAR2 (128)		Name of the user to whom access was granted



Column	Datatype	NULL	Description
GRANTEE_TYPE	VARCHAR2(5)		Type of the grantee: Database or Real Application Security user or role
SCHEMA	VARCHAR2(128)		Schema of the privilege

# 9.18 DBA\_XS\_SECURITY\_CLASSES

The DBA\_XS\_SECURITY\_CLASSES data dictionary view lists all Real Application Security security classes defined in the database.

#### **Related Views**

- USER\_XS\_SECURITY\_CLASSES
- DBA\_XS\_SECURITY\_CLASS\_DEP
- USER\_XS\_SECURITY\_CLASS\_DEP
- ALL\_XS\_SECURITY\_CLASSES
- ALL\_XS\_SECURITY\_CLASS\_DEP

Column	Datatype	NULL	Description
NAME	VARCHAR2 (128)		Name of the security class.
OWNER	VARCHAR2 (128)		Owner of the security class.
DESCRIPTION	VARCHAR2 (4000)		Description of the security class.

### 9.19 USER\_XS\_SECURITY\_CLASSES

The <code>USER\_XS\_SECURITY\_CLASSES</code> data dictionary view lists all Real Application Security security classes owned by the current user.

- DBA\_XS\_SECURITY\_CLASSES
- DBA\_XS\_SECURITY\_CLASS\_DEP
- USER\_XS\_SECURITY\_CLASS\_DEP
- ALL\_XS\_SECURITY\_CLASS\_DEP
- ALL\_XS\_SECURITY\_CLASSES

Column	Datatype	NULL	Description
NAME	VARCHAR2(128)		Name of the security class.
DESCRIPTION	VARCHAR2(4000)		Description of the security class.



### 9.20 ALL\_XS\_SECURITY\_CLASSES

The ALL\_XS\_SECURITY\_CLASSES data dictionary view lists all Real Application Security security classes accessible to the current user.

#### **Related Views**

- USER\_XS\_SECURITY\_CLASSES
- DBA\_XS\_SECURITY\_CLASS\_DEP
- USER\_XS\_SECURITY\_CLASS\_DEP
- ALL\_XS\_SECURITY\_CLASS\_DEP

Column	Datatype	NULL	Description
NAME	VARCHAR2 (128)		Name of the security class.
OWNER	VARCHAR2(128)		Owner of the security class.
DESCRIPTION	VARCHAR2 (4000)		Description of the security class.

### 9.21 DBA\_XS\_SECURITY\_CLASS\_DEP

The DBA\_XS\_SECURITY\_CLASS\_DEP data dictionary view lists the dependencies between all security classes defined in the database.

#### **Related Views**

- USER\_XS\_SECURITY\_CLASS\_DEP
- DBA\_XS\_SECURITY\_CLASSES
- USER\_XS\_SECURITY\_CLASSES
- ALL\_XS\_SECURITY\_CLASS\_DEP
- ALL\_XS\_SECURITY\_CLASSES

Column	Datatype	NULL	Description
SECURITY_CLASS	VARCHAR2 (128)		Name of the security class
OWNER	VARCHAR2(128)		Owner of the security class
PARENT	VARCHAR2(128)		Name of the parent security class
PARENT_OWNER	VARCHAR2 (128)		Owner of the parent security class

# 9.22 USER\_XS\_SECURITY\_CLASS\_DEP

The <code>USER\_XS\_SECURITY\_CLASS\_DEP</code> data dictionary view lists the parent security classes for the dependent security classes owned by the current user.

#### **Related Views**

DBA XS SECURITY CLASS DEP



- DBA\_XS\_SECURITY\_CLASSES
- USER\_XS\_SECURITY\_CLASSES
- ALL\_XS\_SECURITY\_CLASS\_DEP
- ALL\_XS\_SECURITY\_CLASSES

Column	Datatype	NULL	Description
SECURITY_CLASS	VARCHAR2 (128)		Name of the security class
PARENT	VARCHAR2(128)		Name of the parent security class
PARENT_OWNER	VARCHAR2(128)		Owner of the parent security class

# 9.23 ALL\_XS\_SECURITY\_CLASS\_DEP

The ALL\_XS\_SECURITY\_CLASS\_DEP data dictionary view lists all the RAS security classes that the security classes accessible to the current user are dependent on.

#### **Related Views**

- USER\_XS\_SECURITY\_CLASS\_DEP
- DBA\_XS\_SECURITY\_CLASSES
- USER\_XS\_SECURITY\_CLASSES
- ALL\_XS\_SECURITY\_CLASSES

Column	Datatype	NULL	Description
SECURITY_CLASS	VARCHAR2 (128 <b>)</b>		Name of the security class
OWNER	VARCHAR2 (128)		Owner of the security class
PARENT	VARCHAR2 (128)		Name of the parent security class
PARENT_OWNER	VARCHAR2 (128)		Owner of the parent security class

### 9.24 DBA\_XS\_ACLS

The DBA\_XS\_ACLS data dictionary view lists all the existing Real Application Security ACLs defined in the database.

- USER\_XS\_ACLS
- DBA\_XS\_ACES
- ALL\_XS\_ACLS

Column	Datatype	NULL	Description
NAME	VARCHAR2 (128)		Name of the ACL.
OWNER	VARCHAR2 (128)		Owner of the ACL.
SECURITY_CLASS	VARCHAR2 (128)		Name of the security class associated with the ACL



Column	Datatype	NULL	Description
SECURITY_CLASS_OWNER	VARCHAR2 (128)		Owner of the security class associated with the ACL.
PARENT_ACL	VARCHAR2 (128)		Name of the parent ACL.
PARENT_ACL_OWNER	VARCHAR2 (128)		Owner of the parent ACL
INHERITANCE_TYPE	VARCHAR2(11)		Inheritance type of the ACL (EXTENDED or CONSTRAINED)
DESCRIPTION	VARCHAR2(4000)		Description of the ACL

# 9.25 USER\_XS\_ACLS

The USER XS ACLS data dictionary view lists all the ACLs owned by the current user.

#### **Related Views**

- DBA\_XS\_ACLS
- USER\_XS\_ACES
- ALL\_XS\_ACLS
- ALL\_XS\_ACES

Column	Datatype	NULL	Description
NAME	VARCHAR2 (128)		Name of the ACL.
SECURITY_CLASS	VARCHAR2 (128)		Name of the security class associated with the ACL
SECURITY_CLASS_OWNER	VARCHAR2 (128)		Owner of the security class associated with the ACL.
PARENT_ACL	VARCHAR2(128)		Name of the parent ACL.
PARENT_ACL_OWNER	VARCHAR2(128)		Owner of the parent ACL
INHERITANCE_TYPE	VARCHAR2(11)		Inheritance type of the ACL (EXTENDED or CONSTRAINED)
DESCRIPTION	VARCHAR2 (4000)		Description of the ACL

# 9.26 ALL\_XS\_ACLS

The  $ALL\_XS\_ACLS$  data dictionary view lists all the existing Real Application Security ACLs accessible to the current user.

- USER\_XS\_ACLS
- DBA\_XS\_ACES

Column	Datatype	NULL	Description
NAME	VARCHAR2 (128)		Name of the ACL.



Column	Datatype	NULL	Description
OWNER	VARCHAR2 (128)		Owner of the ACL.
SECURITY_CLASS	VARCHAR2 (128)		Name of the security class associated with the ACL
SECURITY_CLASS_OWNER	VARCHAR2 (128)		Owner of the security class associated with the ACL.
PARENT_ACL	VARCHAR2 (128)		Name of the parent ACL.
PARENT_ACL_OWNER	VARCHAR2 (128)		Owner of the parent ACL
INHERITANCE_TYPE	VARCHAR2 (11)		Inheritance type of the ACL (EXTENDED or CONSTRAINED)
DESCRIPTION	VARCHAR2 (4000)		Description of the ACL

# 9.27 DBA\_XS\_ACES

The  $DBA\_XS\_ACES$  data dictionary view lists all the Access Control Entries (ACEs) defined in the database.

- USER\_XS\_ACES
- DBA\_XS\_ACLS
- ALL\_XS\_ACES
- ALL\_XS\_ACLS

Column	Datatype	NULL	Description
ACL	VARCHAR2 (128)		Name of the ACL
OWNER	VARCHAR2 (128)		Owner of the ACL
ACE_ORDER	NUMBER	NOT NULL	Order number of the ACE in the ACL
START_DATE	TIMESTAMP(6)		Effective start date of the ACE
END_DATE	TIMESTAMP(6)		Effective end date of the ACE
GRANT_TYPE	VARCHAR2(5)		Specifies whether the ACE is a GRANT or DENY
INVERTED_PRINCIPAL	VARCHAR2(3)		${\tt YES}$ if the principal is inverted, else ${\tt NO}$
PRINCIPAL	VARCHAR2 (128)		Name of the principal to whom the ACE applies
PRINCIPAL_TYPE	VARCHAR2(16)		Type of the principal, such as application user or application role
PRIVILEGE	VARCHAR2 (128)		Name of the application privilege
SECURITY_CLASS	VARCHAR2(128)		Name of the security class that scopes the ACL



Column	Datatype	NULL	Description
SECURITY_CLASS_OWNER	VARCHAR2 (128)		Owner of the security class that scopes the ACL

# 9.28 USER\_XS\_ACES

The  $\tt USER\_XS\_ACES$  data dictionary view lists all the Access Control Entries (ACEs) from the ACLs owned by the current user.

#### **Related Views**

- DBA\_XS\_ACES
- USER\_XS\_ACLS
- ALL\_XS\_ACES
- ALL\_XS\_ACLS

Column	Datatype	NULL	Description
ACL	VARCHAR2(128)		Name of the ACL
ACE_ORDER	NUMBER	NOT NULL	Order number of the ACE in the ACL
START_DATE	TIMESTAMP(6)		Effective start date of the ACE
END_DATE	TIMESTAMP(6)		Effective end date of the ACE
GRANT_TYPE	VARCHAR2(5)		Specifies whether the ACE is a GRANT or DENY
INVERTED_PRINCIPAL	VARCHAR2(3)		${\tt YES}$ if the principal is inverted, else ${\tt NO}$
PRINCIPAL	VARCHAR2 (128)		Name of the principal to whom the ACE applies
PRINCIPAL_TYPE	VARCHAR2 (16)		Type of the principal, such as application user or application role
PRIVILEGE	VARCHAR2(128)		Name of the application privilege
SECURITY_CLASS	VARCHAR2 (128)		Name of the security class that scopes the ACL
SECURITY_CLASS_OWNER	VARCHAR2 (128)		Owner of the security class that scopes the ACL

# 9.29 ALL\_XS\_ACES

The  ${\tt ALL\_XS\_ACES}$  data dictionary view lists all the Access Control Entries (ACEs) accessible to the current user.

- USER\_XS\_ACES
- DBA\_XS\_ACLS



Column	Datatype	NULL	Description
ACL	VARCHAR2 (128)		Name of the ACL
OWNER	VARCHAR2 (128)		Owner of the ACL
ACE_ORDER	NUMBER	NOT NULL	Order number of the ACE in the ACL
START_DATE	TIMESTAMP(6)		Effective start date of the ACE
END_DATE	TIMESTAMP(6)		Effective end date of the ACE
GRANT_TYPE	VARCHAR2(5)		Specifies whether the ACE is a GRANT or DENY
INVERTED_PRINCIPAL	VARCHAR2(3)		${\tt YES}$ if the principal is inverted, else ${\tt NO}$
PRINCIPAL	VARCHAR2(128)		Name of the principal to whom the ACE applies
PRINCIPAL_TYPE	VARCHAR2 (16)		Type of the principal, such as application user or application role
PRIVILEGE	VARCHAR2 (128)		Name of the application privilege
SECURITY_CLASS	VARCHAR2(128)		Name of the security class that scopes the ACL
SECURITY_CLASS_OWNER	VARCHAR2(128)		Owner of the security class that scopes the ACL

# 9.30 DBA\_XS\_POLICIES

The DBA\_XS\_POLICIES data dictionary view lists all the existing Real Application Security data security policies defined in the database.

- USER\_XS\_POLICIES
- DBA\_XS\_REALM\_CONSTRAINTS
- DBA\_XS\_COLUMN\_CONSTRAINTS
- ALL\_XS\_POLICIES
- ALL\_XS\_COLUMN\_CONSTRAINTS
- ALL\_XS\_REALM\_CONSTRAINTS

Column	Datatype	NULL	Description
NAME	VARCHAR2 (128)		Name of the data security policy
OWNER	VARCHAR2 (128)		Owner of the data security policy
CREATE_TIME	TIMESTAMP(6)		When was the policy created
MODIFY_TIME	TIMESTAMP(6)		When was the policy last modified
DESCRIPTION	VARCHAR2 (4000)		Description of the data security policy



### 9.31 USER\_XS\_POLICIES

The USER\_XS\_POLICIES data dictionary view lists all the existing Real Application Security data security policies owned by the current user.

#### **Related Views**

- DBA\_XS\_POLICIES
- USER\_XS\_REALM\_CONSTRAINTS
- USER\_XS\_COLUMN\_CONSTRAINTS
- ALL\_XS\_POLICIES
- ALL\_XS\_COLUMN\_CONSTRAINTS
- ALL\_XS\_REALM\_CONSTRAINTS

Column	Datatype	NULL	Description
NAME	VARCHAR2 (128)		Name of the data security policy
CREATE_TIME	TIMESTAMP(6)		When was the policy created
MODIFY_TIME	TIMESTAMP(6)		When was the policy last modified
DESCRIPTION	VARCHAR2 (4000)		Description of the data security policy

### 9.32 ALL\_XS\_POLICIES

The  $ALL\_XS\_POLICIES$  data dictionary view lists all the existing Real Application Security data security policies accessible to the current user.

#### **Related Views**

- USER XS POLICIES
- DBA\_XS\_REALM\_CONSTRAINTS
- DBA XS COLUMN CONSTRAINTS

Column	Datatype	NULL	Description
NAME	VARCHAR2 (128)		Name of the data security policy
OWNER	VARCHAR2 (128)		Owner of the data security policy
CREATE_TIME	TIMESTAMP(6)		When was the policy created
MODIFY_TIME	TIMESTAMP(6)		When was the policy last modified
DESCRIPTION	VARCHAR2(4000)		Description of the data security policy

# 9.33 DBA\_XS\_REALM\_CONSTRAINTS

The DBA\_XS\_REALM\_CONSTRAINTS data dictionary view displays all existing Real Application Security realms in the database.



#### **Related Views**

- USER\_XS\_REALM\_CONSTRAINTS
- DBA\_XS\_COLUMN\_CONSTRAINTS
- ALL\_XS\_REALM\_CONSTRAINTS
- ALL\_XS\_COLUMN\_CONSTRAINTS

Column	Datatype	NULL	Description
POLICY	VARCHAR2 (128)		Name of the data security policy
POLICY_OWNER	VARCHAR2 (128)		Owner of the data security policy
REALM_ORDER	NUMBER	NOT NULL	The order of the realm within the data security policy
REALM_TYPE	VARCHAR2(13)		The type of the realm. Valid values are REGULAR, PARAMETERIZED, and INHERITED.
STATIC	VARCHAR2(7)		Indicates whether the realm is STATIC or DYNAMIC
REALM	VARCHAR2 (4000)		The data realm.
ACL	VARCHAR2 (128)		ACL associated with the realm if the realm type is REGULAR
ACL_OWNER	VARCHAR2(128)		Owner of the ACL associated with the REGULAR realm
PARENT_OBJECT	VARCHAR2 (128)		Name of the parent object if the realm type is INHERITED
PARENT_SCHEMA	VARCHAR2 (128)		Schema of the parent object if the realm type is INHERITED

# 9.34 USER\_XS\_REALM\_CONSTRAINTS

The  ${\tt USER\_XS\_REALM\_CONSTRAINTS}$  data dictionary view displays all existing Real Application Security realms owned by the current user.

- DBA\_XS\_REALM\_CONSTRAINTS
- USER\_XS\_COLUMN\_CONSTRAINTS
- ALL\_XS\_REALM\_CONSTRAINTS
- ALL\_XS\_COLUMN\_CONSTRAINTS

Column	Datatype	NULL	Description
POLICY	VARCHAR2 (128)		Name of the data security policy
REALM_ORDER	NUMBER	NOT NULL	The order of the realm within the data security policy



Column	Datatype	NULL	Description
REALM_TYPE	VARCHAR2 (13)		The type of the realm. Valid values are REGULAR, PARAMETERIZED, and INHERITED.
STATIC	VARCHAR2(7)		Indicates whether the realm is STATIC or DYNAMIC
REALM	VARCHAR2 (4000)		The data realm.
ACL	VARCHAR2 (128)		ACL associated with the realm if the realm type is REGULAR
ACL_OWNER	VARCHAR2 (128)		Owner of the ACL associated with the REGULAR realm
PARENT_OBJECT	VARCHAR2 (128)		Name of the parent object if the realm type is INHERITED
PARENT_SCHEMA	VARCHAR2 (128)		Schema of the parent object if the realm type is INHERITED

# 9.35 ALL\_XS\_REALM\_CONSTRAINTS

The  ${\tt ALL\_XS\_REALM\_CONSTRAINTS}$  data dictionary view displays all existing Real Application Security realms accessible to the current user.

- USER\_XS\_REALM\_CONSTRAINTS
- DBA\_XS\_COLUMN\_CONSTRAINTS

Column	Datatype	NULL	Description
POLICY	VARCHAR2 (128)		Name of the data security policy
POLICY_OWNER	VARCHAR2 (128)		Owner of the data security policy
REALM_ORDER	NUMBER	NOT NULL	The order of the realm within the data security policy
REALM_TYPE	VARCHAR2(13)		The type of the realm. Valid values are REGULAR, PARAMETERIZED, and INHERITED.
STATIC	VARCHAR2(7)		Indicates whether the realm is STATIC or DYNAMIC
REALM	VARCHAR2 (4000)		The data realm.
ACL	VARCHAR2(128)		ACL associated with the realm if the realm type is REGULAR
ACL_OWNER	VARCHAR2(128)		Owner of the ACL associated with the REGULAR realm
PARENT_OBJECT	VARCHAR2(128)		Name of the parent object if the realm type is INHERITED
PARENT_SCHEMA	VARCHAR2 (128)		Schema of the parent object if the realm type is INHERITED



# 9.36 DBA\_XS\_INHERITED\_REALMS

The DBA\_XS\_INHERITED\_REALMS data dictionary view displays all the inherited Real Application Security realms in the database.

#### **Related Views**

- USER\_XS\_INHERITED\_REALMS
- DBA\_XS\_REALM\_CONSTRAINTS
- ALL\_XS\_INHERITED\_REALMS
- ALL\_XS\_REALM\_CONSTRAINTS

Column	Datatype	NULL	Description
POLICY	VARCHAR2 (128)	·	Name of the data security policy
POLICY_OWNER	VARCHAR2 (128)		Owner of the data security policy
REALM_ORDER	NUMBER	NOT NULL	The order of the realm within the data security policy
PARENT_OBJECT	VARCHAR2 (128)		Name of the parent object
PARENT_SCHEMA	VARCHAR2 (128)		Schema of the parent object
PRIMARY_KEY	VARCHAR2 (128)		The column name in the master table
FOREIGN_KEY	VARCHAR2(4000)		The column name or value in the detail table
FOREIGN_KEY_TYPE	VARCHAR2(5)		Type of the foreign key. Possible values are ${\tt NAME}$ and ${\tt VALUE}$ .

### 9.37 USER XS INHERITED REALMS

The  ${\tt USER\_XS\_INHERITED\_REALMS}$  data dictionary view displays all the inherited Real Application Security realms owned by the current user.

- DBA XS INHERITED REALMS
- USER\_XS\_REALM\_CONSTRAINTS
- ALL\_XS\_INHERITED\_REALMS
- ALL\_XS\_REALM\_CONSTRAINTS

Column	Datatype	NULL	Description
POLICY	VARCHAR2 (128)		Name of the data security policy
REALM_ORDER	NUMBER	NOT NULL	The order of the realm within the data security policy
PARENT_OBJECT	VARCHAR2(128)		Name of the parent object
PARENT_SCHEMA	VARCHAR2 (128)		Schema of the parent object



Column	Datatype	NULL	Description
PRIMARY_KEY	VARCHAR2 (128)		The column name in the master table
FOREIGN_KEY	VARCHAR2 (4000)		The column name or value in the detail table
FOREIGN_KEY_TYPE	VARCHAR2(5)		Type of the foreign key. Possible values are ${\tt NAME}$ and ${\tt VALUE}$ .

# 9.38 ALL\_XS\_INHERITED\_REALMS

The  $ALL\_XS\_INHERITED\_REALMS$  data dictionary view displays all the inherited Real Application Security realms accessible to the current user.

#### **Related Views**

- USER\_XS\_INHERITED\_REALMS
- DBA\_XS\_REALM\_CONSTRAINTS

Column	Datatype	NULL	Description
POLICY	VARCHAR2 (128)		Name of the data security policy
POLICY_OWNER	VARCHAR2(128)		Owner of the data security policy
REALM_ORDER	NUMBER	NOT NULL	The order of the realm within the data security policy
PARENT_OBJECT	VARCHAR2(128)		Name of the parent object
PARENT_SCHEMA	VARCHAR2(128)		Schema of the parent object
PRIMARY_KEY	VARCHAR2 (128)		The column name in the master table
FOREIGN_KEY	VARCHAR2(4000)		The column name or value in the detail table
FOREIGN_KEY_TYPE	VARCHAR2(5)		Type of the foreign key. Possible values are NAME and VALUE.

# 9.39 DBA\_XS\_ACL\_PARAMETERS

The DBA\_XS\_ACL\_PARAMETERS data dictionary view displays all existing Real Application Security ACL parameters.

- USER\_XS\_ACL\_PARAMETERS
- DBA\_XS\_REALM\_CONSTRAINTS
- ALL\_XS\_ACL\_PARAMETERS
- ALL\_XS\_REALM\_CONSTRAINTS



Column	Datatype	NULL	Description
POLICY	VARCHAR2 (128)		Name of the data security policy where the ACL parameter is defined
POLICY_OWNER	VARCHAR2 (128)		Owner of the data security policy where the ACL parameter is defined
ACL	VARCHAR2 (128)		Name of the ACL
ACL_OWNER	VARCHAR2 (128)		Owner of the ACL
PARAMETER	VARCHAR2 (128)		Name of the ACL parameter
DATATYPE	VARCHAR2(9)		Data type of the ACL parameter
VALUE	VARCHAR2 (4000)		Value of the ACL parameter
REALM_ORDER	NUMBER		The order of the realm within the data security policy
REALM	VARCHAR2(4000)		The realm that contains the ACL parameter

# 9.40 USER\_XS\_ACL\_PARAMETERS

The <code>USER\_XS\_ACL\_PARAMETERS</code> data dictionary view displays all ACL parameters defined in the data security policies owned by the current user.

- DBA\_XS\_ACL\_PARAMETERS
- USER\_XS\_REALM\_CONSTRAINTS
- ALL\_XS\_ACL\_PARAMETERS
- ALL\_XS\_REALM\_CONSTRAINTS

Column	Datatype	NULL	Description
POLICY	VARCHAR2 (128 <b>)</b>		Name of the data security policy where the ACL parameter is defined
ACL	VARCHAR2(128)		Name of the ACL
ACL_OWNER	VARCHAR2(128)		Owner of the ACL
PARAMETER	VARCHAR2(128)		Name of the ACL parameter
DATATYPE	VARCHAR2(9)		Data type of the ACL parameter
VALUE	VARCHAR2(4000)		Value of the ACL parameter
REALM_ORDER	NUMBER		The order of the realm within the data security policy
REALM	VARCHAR2 (4000)		The realm that contains the ACL parameter



# 9.41 ALL\_XS\_ACL\_PARAMETERS

The ALL\_XS\_ACL\_PARAMETERS data dictionary view displays all existing Real Application Security ACL parameters defined in the data security policies accessible to the current user.

#### **Related Views**

- USER\_XS\_ACL\_PARAMETERS
- DBA\_XS\_REALM\_CONSTRAINTS

Column	Datatype	NULL	Description
POLICY	VARCHAR2 (128)		Name of the data security policy where the ACL parameter is defined
POLICY_OWNER	VARCHAR2(128)		Owner of the data security policy where the ACL parameter is defined
ACL	VARCHAR2 (128)		Name of the ACL
ACL_OWNER	VARCHAR2 (128)		Owner of the ACL
PARAMETER	VARCHAR2 (128)		Name of the ACL parameter
DATATYPE	VARCHAR2(9)		Data type of the ACL parameter
VALUE	VARCHAR2 (4000)		Value of the ACL parameter
REALM_ORDER	NUMBER		The order of the realm within the data security policy
REALM	VARCHAR2 (4000)		The realm that contains the ACL parameter

# 9.42 DBA\_XS\_COLUMN\_CONSTRAINTS

The  $\mbox{DBA}_{\mbox{XS}}\mbox{COLUMN}_{\mbox{CONSTRAINTS}}$  data dictionary view lists all Real Application Security column constraints defined in the database.

- USER\_XS\_COLUMN\_CONSTRAINTS
- DBA\_XS\_POLICIES
- ALL\_XS\_COLUMN\_CONSTRAINTS
- ALL\_XS\_POLICIES

Column	Datatype	NULL	Description
POLICY	VARCHAR2(128)	NA	Name of the data security policy containing the column constraint
OWNER	VARCHAR2(128)	NA	Owner of the data security policy containing the column constraint
COLUMN_NAME	VARCHAR2(128)	NA	Name of the column that has the column constraint applied to it
PRIVILEGE	VARCHAR2 (128)	NA	Name of the application privilege required to access the column



### 9.43 USER\_XS\_COLUMN\_CONSTRAINTS

The <code>USER\_XS\_COLUMN\_CONSTRAINTS</code> data dictionary view lists all Real Application Security column constraints owned by the current user.

#### **Related Views**

- DBA\_XS\_COLUMN\_CONSTRAINTS
- USER\_XS\_POLICIES
- ALL\_XS\_COLUMN\_CONSTRAINTS
- ALL\_XS\_POLICIES

Column	Datatype	NULL	Description
POLICY	VARCHAR2 (128)		Name of the data security policy containing the column constraint
OWNER	VARCHAR2(128)		Owner of the data security policy containing the column constraint
COLUMN_NAME	VARCHAR2(128)		Name of the column that has the column constraint applied to it
PRIVILEGE	VARCHAR2 (128)		Name of the application privilege required to access the column

# 9.44 ALL\_XS\_COLUMN\_CONSTRAINTS

The ALL\_XS\_COLUMN\_CONSTRAINTS data dictionary view lists all Real Application Security column constraints accessible to the current user.

#### **Related Views**

- USER\_XS\_COLUMN\_CONSTRAINTS
- DBA\_XS\_POLICIES

Column	Datatype	NULL	Description
POLICY	VARCHAR2 (128)	NA	Name of the data security policy containing the column constraint
OWNER	VARCHAR2 (128)	NA	Owner of the data security policy containing the column constraint
COLUMN_NAME	VARCHAR2 (128)	NA	Name of the column that has the column constraint applied to it
PRIVILEGE	VARCHAR2 (128)	NA	Name of the application privilege required to access the column

### 9.45 DBA\_XS\_APPLIED\_POLICIES

The  $\mbox{DBA}_{XS}_{APPLIED}_{POLICIES}$  data dictionary view displays all database objects on which Real Application Security data security policies are enabled.



#### **Related Views**

- DBA\_XS\_POLICIES
- ALL\_XS\_POLICIES

Column	Datatype	NULL	Description
SCHEMA	VARCHAR2 (128)	NOT NULL	Schema containing the object
OBJECT	VARCHAR2 (128)	NOT NULL	Name of the data security enabled object in the database
POLICY	VARCHAR2 (128)		Name of the data security policy associated with the object
POLICY_OWNER	VARCHAR2 (128)	NOT NULL	Owner of the data security policy associated with the object
SEL	VARCHAR2(3)		Policy enabled for SELECT statements
INS	VARCHAR2(3)		Policy enabled for INSERT statements
UPD	VARCHAR2(3)		Policy enabled for UPDATE statements
DEL	VARCHAR2(3)		Policy enabled for DELETE statements
IDX	VARCHAR2(3)		Policy enabled for INDEX statements
STATUS	VARCHAR2(8)		ENABLED if the data security policy is enabled for the object, else DISABLED

# 9.46 ALL\_XS\_APPLIED\_POLICIES

The ALL\_XS\_APPLIED\_POLICIES data dictionary view displays all database objects on which Real Application Security data security policies are accessible to the current user are enabled.

#### **Related Views**

DBA\_XS\_POLICIES

Column	Datatype	NULL	Description
SCHEMA	VARCHAR2 (128)	NOT NULL	Schema containing the object
OBJECT	VARCHAR2 (128)	NOT NULL	Name of the data security enabled object in the database
POLICY	VARCHAR2 (128)		Name of the data security policy associated with the object
POLICY_OWNER	VARCHAR2 (128)	NOT NULL	Owner of the data security policy associated with the object
SEL	VARCHAR2 (3)		Policy enabled for SELECT statements
INS	VARCHAR2(3)		Policy enabled for INSERT statements
UPD	VARCHAR2(3)		Policy enabled for UPDATE statements
DEL	VARCHAR2(3)		Policy enabled for DELETE statements
IDX	VARCHAR2(3)		Policy enabled for INDEX statements



Column	Datatype	NULL	Description
STATUS	VARCHAR2(8)		ENABLED if the data security policy is enabled for the object, else DISABLED

# 9.47 DBA\_XS\_MODIFIED\_POLICIES

The DBA\_XS\_MODIFIED\_POLICIES data dictionary view displays all database objects on which Real Application Security data security policies are modified.

#### **Related Views**

- DBA\_XS\_POLICIES
- DBA\_XS\_APPLIED\_POLICIES

Column	Datatype	NULL	Description
POLICY	VARCHAR2(128)	NOT NULL	Name of the data security policy associated with the object
OBJECT	VARCHAR2 (128)	NOT NULL	Name of the data security modified object in the database

# 9.48 DBA\_XS\_SESSIONS

The DBA\_XS\_SESSIONS dynamic data dictionary view displays all the application sessions in the database. Only database administrators can select from this view.

- DBA\_XS\_ACTIVE\_SESSIONS
- DBA\_XS\_SESSION\_ROLES
- DBA\_XS\_SESSION\_NS\_ATTRIBUTES

Column	Datatype	NULL	Description
USER_NAME	VARCHAR2 (128 <b>)</b>	NOT NULL	Application user name of the application session
SESSIONID	RAW(16)	NOT NULL	Application Session identifier
PROXY_USER	VARCHAR2 (128)		Name of the proxy application user
COOKIE	VARCHAR2 (1024)		The server-unique cookie value associated with the session
CREATE_TIME	TIMESTAMP(6)	NOT NULL	Creation time for the application session
AUTH_TIME	TIMESTAMP(6)	NOT NULL	Last time the application user was authenticated.
ACCESS_TIME	TIMESTAMP(6)	NOT NULL	Last time that the application session was accessed
INACTIVE_TIMEOUT	NUMBER(6)		The amount of time (in minutes) before the application session is considered timed out



# 9.49 DBA\_XS\_ACTIVE\_SESSIONS

The DBA\_XS\_ACTIVE\_SESSIONS dynamic data dictionary view displays all attached application sessions in the database. Only database administrators can select from this view.

#### **Related Views**

- DBA\_XS\_SESSIONS
- DBA\_XS\_SESSION\_ROLES
- DBA\_XS\_SESSION\_NS\_ATTRIBUTES

Column	Datatype	NULL	Description
USER_NAME	VARCHAR2 (128)	NOT NULL	Application user name of the application session
SESSIONID	RAW(16)	NOT NULL	Application Session identifier
DATABASE_SESSIONID	NUMBER		The database session ID to which the application session is associated.
PROXY_USER	VARCHAR2(128)		Name of the proxy application user
COOKIE	VARCHAR2(1024)		The server-unique cookie value associated with the session
CREATE_TIME	TIMESTAMP(6)	NOT NULL	Creation time for the application session
AUTH_TIME	TIMESTAMP(6)	NOT NULL	Last time the application user was authenticated.
ACCESS_TIME	TIMESTAMP(6)	NOT NULL	Last time that the application session was accessed
INACTIVE_TIMEOUT	NUMBER(6)		The amount of time (in minutes) before the application session is considered timed out

# 9.50 DBA\_XS\_SESSION\_ROLES

The  $\DBA_XS_SESSION_ROLES$  dynamic data dictionary view lists application roles enabled in application sessions.

- DBA\_XS\_SESSIONS
- DBA\_XS\_ACTIVE\_SESSIONS
- DBA\_XS\_SESSION\_NS\_ATTRIBUTES

Column	Datatype	NULL	Description
SESSIONID	RAW (16)	NOT NULL	Application session ID
ROLE	VARCHAR2 (128)	NOT NULL	Name of the application role



# 9.51 DBA\_XS\_SESSION\_NS\_ATTRIBUTES

The DBA\_XS\_SESSION\_NS\_ATTRIBUTES data dictionary view displays namespace attributes across application sessions as of last saved state.

#### **Related Views**

- DBA\_XS\_SESSIONS
- DBA\_XS\_ACTIVE\_SESSIONS
- DBA\_XS\_SESSION\_ROLES

Column	Datatype	NULL	Description
SESSIONID	RAW(16)	NOT NULL	Session ID of the application session
ATTRIBUTE	VARCHAR2 (4000)		Name of the attribute
NAMESPACE	VARCHAR2 (128)	NOT NULL	Name of the namespace
VALUE	VARCHAR2 (4000)		Value of the attribute
DEFAULT_VALUE	VARCHAR2 (4000)		Default value of the attribute
FIRSTREAD_EVENT	VARCHAR2(2)		Indicates whether the handler function is invoked when the attribute is first read. Possible values are YES and NO.
MODIFY_EVENT	VARCHAR2(2)		Indicates whether the handler function is invoked when the attribute is modified. Possible values are YES and NO.

# 9.52 DBA\_XS\_NS\_TEMPLATES

The DBA\_XS\_NS\_TEMPLATES data dictionary view describes all Real Application Security namespace templates.

#### **Related Views**

• DBA XS NS TEMPLATE ATTRIBUTES

Column	Datatype	NULL	Description
NAME	VARCHAR2 (128)		Name of the namespace template
HANDLER_SCHEMA	VARCHAR2 (128)		Schema of the namespace handler function



Column	Datatype	NULL	Description
HANDLER_PACKAGE	VARCHAR2 (128)		Package containing the namespace handler function
HANDLER_FUNCTION	VARCHAR2 (128)		The namespace handler function
HANDLER_STATUS	VARCHAR2 (7)		Indicates whether the namespace handler function is VALID or INVALID.
ACL	VARCHAR2 (128)		Name of ACL for the namespace template.
DESCRIPTION	VARCHAR2 (4000)		Description of the namespace template.

# 9.53 DBA\_XS\_NS\_TEMPLATE\_ATTRIBUTES

The  $\mbox{dba}_{\mbox{XS}_{\mbox{NS}_{\mbox{TEMPLATE}}}$  attributes data dictionary view describes all namespace attributes defined in namespace template documents.

#### **Related Views**

### • DBA\_XS\_NS\_TEMPLATES

Column	Datatype	NULL	Description
ATTRIBUTE	VARCHAR2 (4000)		Name of the attribute defined in the namespace template
NAMESPACE	VARCHAR2 (128)		Name of the namespace instantiated by the namespace template
DEFAULT_VALUE	VARCHAR2 (4000)		Default value of the attribute defined in the namespace template
FIRSTREAD_EVENT	VARCHAR2(3)		Indicates whether the namespace handler function is invoked when the attribute is first read. Valid values are YES and NO.
MODIFY_EVENT	VARCHAR2(3)		Indicates whether the namespace handler function is invoked when the attribute value is modified. Valid values are YES and NO.



# 9.54 ALL\_XDS\_ACL\_REFRESH

The ALL\_XDS\_ACL\_REFRESH data dictionary view displays all static ACL refresh settings for tables that are accessible to the application user.

#### **Related Views**

- DBA\_XDS\_ACL\_REFRESH
- USER\_XDS\_ACL\_REFRESH

Column	Datatype	NULL	Description
SCHEMA_NAME	VARCHAR2 (128)	NOT NULL	Name of schema
TABLE_NAME	VARCHAR2(128)	NOT NULL	Name of table
ACL_MVIEW_NAME	VARCHAR2(128)	NOT NULL	Name of ACL MV for this table
REFRESH_MODE	VARCHAR2(9)		ON COMMIT, SCHEDULED, or ON DEMAND
REFRESH_ABILITY	VARCHAR2(11)		COMPLETE or INCREMENTAL
ACL_STATUS	VARCHAR2(5)		STALE or FRESH
USER_SUPPLIED_MV	VARCHAR2(1)		Y or N
START_DATE	TIMESTAMP(6) WITH TIME ZONE		The refreshment job scheduled to run after the timestamp, if scheduled
REPEAT_INTERVAL	VARCHAR2(4000)		The repeat_interval to run the refreshment job, if scheduled
REFRESH_COUNT	NUMBER		Number of times this ACL MV has been refreshed so far
COMMENTS	VARCHAR2(240)		Comments for the refreshment

# 9.55 ALL\_XDS\_ACL\_REFSTAT

The ALL\_XDS\_ACL\_REFSTAT data dictionary view displays all static ACL refresh job status history that has been done for tables that are accessible to the application user.

- DBA\_XDS\_ACL\_REFSTAT
- USER\_XDS\_ACL\_REFSTAT

Column	Datatype	NULL	Description
SCHEMA_NAME	VARCHAR2 (128 <b>)</b>	NOT NULL	Name of schema
TABLE_NAME	VARCHAR2 (128)	NOT NULL	Name of table



Column	Datatype	NULL	Description
REFRESH_MODE	VARCHAR2(9)		ON COMMIT, SCHEDULED, or ON DEMAND
REFRESH_ABILITY	VARCHAR2(11)		COMPLETE or INCREMENTAL
JOB_START_TIME	TIMESTAMP(6) WITH TIME ZONE		The refreshment job starting time
JOB_END_TIME	TIMESTAMP(6) WITH TIME ZONE		The refreshment job ending time
ROW_UPDATE_COUNT	NUMBER		Number of rows have been updated for static ACL sync
STATUS	NUMBER		Refreshment job status:
			0 means success, otherwise an error number is displayed.
ERROR_MESSAGE	VARCHAR2 (4000)		The error message for the error, if there is any.

# 9.56 ALL\_XDS\_LATEST\_ACL\_REFSTAT

The ALL\_XDS\_LATEST\_ACL\_REFSTAT data dictionary view displays all latest static ACL refresh job status history that has been done for tables that are accessible to the application user. It has the same schema as ALL XDS ACL REFSTAT dictionary view, but a subset of its rows.

- DBA\_XDS\_LATEST\_ACL\_REFSTAT
- USER\_XDS\_LATEST\_ACL\_REFSTAT
- ALL\_XDS\_ACL\_REFSTAT

Column	Datatype	NULL	Description
SCHEMA_NAME	VARCHAR2 (128)	NOT NULL	Name of schema
TABLE_NAME	VARCHAR2 (128)	NOT NULL	Name of table
REFRESH_MODE	VARCHAR2(9)		ON COMMIT, SCHEDULED, or ON DEMAND
REFRESH_ABILITY	VARCHAR2(11)		COMPLETE or INCREMENTAL
JOB_START_TIME	TIMESTAMP(6) WITH TIME ZONE		The refreshment job starting time
JOB_END_TIME	TIMESTAMP(6) WITH TIME ZONE		The refreshment job ending time
ROW_UPDATE_COUNT	NUMBER		Number of rows have been updated for static ACL sync
STATUS	NUMBER		Refreshment job status:
			0 means success, otherwise an error number is displayed.



Column	Datatype	NULL	Description
ERROR_MESSAGE	VARCHAR2 (4000)		The error message for the error, if there is any.

# 9.57 DBA\_XDS\_ACL\_REFRESH

The  $\mbox{DBA\_XDS\_ACL\_REFRESH}$  data dictionary view displays all static ACL refresh settings in the database.

### **Related Views**

- ALL\_XDS\_ACL\_REFRESH
- USER\_XDS\_ACL\_REFRESH

Column	Datatype	NULL	Description
SCHEMA_NAME	VARCHAR2 (128)	NOT NULL	Name of schema
TABLE_NAME	VARCHAR2 (128)	NOT NULL	Name of table
ACL_MVIEW_NAME	VARCHAR2 (128)	NOT NULL	Name of ACL MV for this table
REFRESH_MODE	VARCHAR2(9)		ON COMMIT, SCHEDULED, or ON DEMAND
REFRESH_ABILITY	VARCHAR2(11)		COMPLETE or INCREMENTAL
ACL_STATUS	VARCHAR2(5)		STALE or FRESH
USER_SUPPLIED_MV	VARCHAR2(1)		Y or N
START_DATE	TIMESTAMP(6) WITH TIME ZONE		The refreshment job scheduled to run after the timestamp, if scheduled
REPEAT_INTERVAL	VARCHAR2(4000)		The repeat_interval to run the refreshment job, if scheduled.
REFRESH_COUNT	NUMBER		Number of refreshment has been done so far
COMMENTS	VARCHAR2(240)		Comments for the refreshment

# 9.58 DBA\_XDS\_ACL\_REFSTAT

The  $\DBA\_XDS\_ACL\_REFSTAT$  data dictionary view displays all static ACL refresh job status history that has been done in the database.

- ALL\_XDS\_ACL\_REFSTAT
- USER\_XDS\_ACL\_REFSTAT



Column	Datatype	NULL	Description
SCHEMA_NAME	VARCHAR2 (128)	NOT NULL	Name of schema
TABLE_NAME	VARCHAR2 (128)	NOT NULL	Name of table
REFRESH_MODE	VARCHAR2(9)		ON COMMIT, SCHEDULED, or ON DEMAND
REFRESH_ABILITY	VARCHAR2(11)		COMPLETE or INCREMENTAL
JOB_START_TIME	TIMESTAMP(6) WITH TIME ZONE		The refreshment job starting time
JOB_END_TIME	TIMESTAMP(6) WITH		The refreshment job ending time
ROW_UPDATE_COUNT	NUMBER		Number of rows have been updated for static ACL sync
STATUS	NUMBER		Refreshment job status:
			0 means success, otherwise an error number is displayed.
ERROR_MESSAGE	VARCHAR2(4000)		The error message for the error, if there is any.

# 9.59 DBA\_XDS\_LATEST\_ACL\_REFSTAT

The <code>DBA\_XDS\_LATEST\_ACL\_REFSTAT</code> data dictionary view displays all latest static ACL refresh job status history that has been done in the database. It has the same schema as  $DBA_XDS_ACL_REFSTAT$  dictionary view, but a subset of its rows.

- ALL\_XDS\_LATEST\_ACL\_REFSTAT
- USER\_XDS\_LATEST\_ACL\_REFSTAT
- DBA\_XDS\_ACL\_REFSTAT

Column	Datatype	NULL	Description
SCHEMA_NAME	VARCHAR2 (128)	NOT NULL	Name of schema
TABLE_NAME	VARCHAR2 (128)	NOT NULL	Name of table
REFRESH_MODE	VARCHAR2(9)		ON COMMIT, SCHEDULED, or ON DEMAND
REFRESH_ABILITY	VARCHAR2(11)		COMPLETE or INCREMENTAL
JOB_START_TIME	TIMESTAMP(6) WITH TIME ZONE		The refreshment job starting time
JOB_END_TIME	TIMESTAMP(6) WITH TIME ZONE		The refreshment job ending time
ROW_UPDATE_COUNT	NUMBER		Number of rows have been updated for static ACL sync



Column	Datatype	NULL	Description
STATUS	NUMBER		Refreshment job status:
			0 means success, otherwise an error number is displayed.
ERROR_MESSAGE	VARCHAR2 (4000)		The error message for the error, if there is any.

# 9.60 USER\_XDS\_ACL\_REFRESH

The  $\tt USER\_XDS\_ACL\_REFRESH$  data dictionary view displays all static ACL refresh settings for tables that are owned by the user.

### **Related Views**

- ALL\_XDS\_ACL\_REFRESH
- DBA\_XDS\_ACL\_REFRESH

Column	Datatype	NULL	Description
SCHEMA_NAME	VARCHAR2 (128)	NOT NULL	Name of schema
TABLE_NAME	VARCHAR2 (128)	NOT NULL	Name of table
ACL_MVIEW_NAME	VARCHAR2 (128)	NOT NULL	Name of ACL MV for this table
REFRESH_MODE	VARCHAR2(9)		ON COMMIT, SCHEDULED, or ON DEMAND
REFRESH_ABILITY	VARCHAR2(11)		COMPLETE or INCREMENTAL
ACL_STATUS	VARCHAR2(5)		STALE or FRESH
USER_SUPPLIED_MV	VARCHAR2(1)		Y or N
START_DATE	TIMESTAMP(6) WITH TIME ZONE		The refreshment job scheduled to run after the timestamp, if scheduled
REPEAT_INTERVAL	VARCHAR2 (4000)		The repeat_interval to run the refreshment job, if scheduled.
REFRESH_COUNT	NUMBER		Number of refreshment has been done so far
COMMENTS	VARCHAR2(240)		Comments for the refreshment

# 9.61 USER\_XDS\_ACL\_REFSTAT

The  ${\tt USER\_XDS\_ACL\_REFSTAT}$  data dictionary view displays all static ACL refresh job status history that has been done for tables that are owned by the user.

- ALL\_XDS\_ACL\_REFSTAT
- DBA\_XDS\_ACL\_REFSTAT



Column	Datatype	NULL	Description
SCHEMA_NAME	VARCHAR2 (128)	NOT NULL	Name of schema
TABLE_NAME	VARCHAR2 (128)	NOT NULL	Name of table
REFRESH_MODE	VARCHAR2(9)		ON COMMIT, SCHEDULED, or ON DEMAND
REFRESH_ABILITY	VARCHAR2(11)		COMPLETE or INCREMENTAL
JOB_START_TIME	TIMESTAMP(6) WITH		The refreshment job starting time
JOB_END_TIME	TIMESTAMP(6) WITH		The refreshment job ending time
ROW_UPDATE_COUNT	NUMBER		Number of rows have been updated for static ACL sync
STATUS	NUMBER		Refreshment job status:
			0 means success, otherwise an error number is displayed.
ERROR_MESSAGE	VARCHAR2(4000)		The error message for the error, if there is any.

# 9.62 USER\_XDS\_LATEST\_ACL\_REFSTAT

The  ${\tt USER\_XDS\_LATEST\_ACL\_REFSTAT}$  data dictionary view displays all latest static ACL refresh job status history that has been done for tables that are owned by the user. It has the same schema as  ${\tt USER\_XDS\_ACL\_REFSTAT}$  dictionary view, but a subset of its rows.

- ALL\_XDS\_LATEST\_ACL\_REFSTAT
- DBA\_XDS\_LATEST\_ACL\_REFSTAT
- USER\_XDS\_ACL\_REFSTAT

Column	Datatype	NULL	Description
SCHEMA_NAME	VARCHAR2 (128)	NOT NULL	Name of schema
TABLE_NAME	VARCHAR2 (128)	NOT NULL	Name of table
REFRESH_MODE	VARCHAR2(9)		ON COMMIT, SCHEDULED, or ON DEMAND
REFRESH_ABILITY	VARCHAR2(11)		COMPLETE or INCREMENTAL
JOB_START_TIME	TIMESTAMP(6) WITH TIME ZONE		The refreshment job starting time
JOB_END_TIME	TIMESTAMP(6) WITH TIME ZONE		The refreshment job ending time
ROW_UPDATE_COUNT	NUMBER		Number of rows have been updated for static ACL sync



Column	Datatype	NULL	Description
STATUS	NUMBER		Refreshment job status:
			0 means success, otherwise an error number is displayed.
ERROR_MESSAGE	VARCHAR2 (4000)		The error message for the error, if there is any.

# 9.63 V\$XS\_SESSION\_NS\_ATTRIBUTES

The V\$XS\_SESSION\_NS\_ATTRIBUTES dynamic data dictionary view displays information about the namespaces and attributes in all application sessions in the database as of the end of the last request. The state of any active request is not reflected in this view. Only database administrators can select from this view.

### **Related Views**

- DBA\_XS\_SESSIONS
- DBA\_XS\_SESSION\_NS\_ATTRIBUTES
- V\$XS\_SESSION\_ROLES

Column	Datatype	NULL	Description
NAMESPACE_NAME	VARCHAR2 (4000)		Name of the namespace
WORKSPACE_NAME	VARCHAR2 (129)		Name of the workspace space for the namespace
ATTRIBUTE_NAME	VARCHAR2(4000)		Name of the attribute
ATTRIBUTE_VALUE	VARCHAR2(4000)		Value of the attribute
ATTRIBUTE_EVENTS	VARCHAR2(4000)		Events associated with this attribute
ATTRIBUTE_DEFAULT_VAL UE	VARCHAR2 (4000)		Default value for the attribute
ATTRIBUTE_TYPE	VARCHAR2(4000)		Type of attribute, either TEMPLATE or CUSTOM
CON_ID	NUMBER		Container ID

# 9.64 V\$XS\_SESSION\_ROLES

The V\$XS\_SESSION\_ROLES static data dictionary view displays all enabled application roles in application session in the current request.

- DBA\_XS\_SESSIONS
- DBA\_XS\_SESSION\_ROLES
- V\$XS\_SESSION\_NS\_ATTRIBUTES



Column	Datatype	NULL	Description
ROLE_WSPACE	VARCHAR2 (129)		The workspace of the application role.
ROLE_NAME	VARCHAR2 (4000)		Name of enabled application role
FLAGS	NUMBER		Status flag
CON_ID	NUMBER		Container ID



10

# Oracle Database Real Application Security SQL Functions

This chapter describes the SQL functions and procedures that are available with Oracle Database Real Application Security.

Table 10-1 summarizes these functions and procedures. Detailed information on each function and procedure follows this table.

Table 10-1 Oracle Database Real Application Security SQL Functions and Procedures

SQL Function or Procedure	Brief Description
COLUMN_AUTH_INDICATOR Function	Checks whether the specified table column is authorized on a particular table row.
XS_SYS_CONTEXT Function	Retrieves the session attributes and the XS\$GLOBAL_VAR namespace attribute for the current application session.
ORA_CHECK_ACL Function	Checks whether an application user has the queried application privileges according to a list of ACLs.
ORA_GET_ACLIDS Function	Returns a list of ACL identifiers associated with an object instance of the XDS-enabled tables for the current application user.
ORA_CHECK_PRIVILEGE Function	Checks whether the specified system privileges have been granted to an application user
TO_ACLID Function	Returns the ACL IDs of the supplied ACL names

# 10.1 COLUMN AUTH INDICATOR Function

The <code>COLUMN\_AUTH\_INDICATOR</code> function checks whether the specified table column is authorized on a particular table row. If the current application user is authorized by data security policies to access the column value of the current row, or if the column is not protected by any data security policies, then it returns 1. If the application user is not authorized, it returns 0.

### **Syntax**

COLUMN\_AUTH\_INDICATOR(col)
RETURN BOOLEAN;

Parameter	Description	
col	A column in a table or view.	
	This parameter does not accept object type columns or expressions.	



### **Example**

```
SELECT po_number, project_id, region,
DECODE(COLUMN_AUTH_INDICATOR(price), 0, 'xxxxxx', 1, price) price
FROM purchaseorder
WHERE po_number
BETWEEN 10000 and 10003;
```

### See Also:

- "Applying Additional Application Privileges to a Column" for more detailed example of using the COLUMN AUTH INDICATOR function
- Oracle Database Real Application Security Data Dictionary Views for information about the ALL\_ATTRIBUTE\_SECS, DBA\_ATTRIBUTE\_SECS, and USER\_ATTRIBUTE\_SECS data dictionary views, which list existing tables that use column level security

# 10.2 XS\_SYS\_CONTEXT Function

The <code>xs\_sys\_context</code> function provides quick access to session attributes in the current application session without incurring the overhead that results from using the PL/SQL APIs. The <code>sys\_xs\_context</code> function definition mirrors that of the <code>sys\_context</code> function and can be described as application session counterpart to <code>sys\_context</code>. <code>xs\_sys\_context</code> returns the requested namespace and attribute. If they do not exist, then it returns <code>NULL</code>.

Table 10-2 lists the attributes in predefined namespace XS\$SESSION.

**Table 10-2** Predefined Parameters

Parameter	Return Value
CREATED_BY	The owner who created the current application session.
CREATE_TIME	The time in which the current application session was created.
COOKIE	The secure session cookie, passed as the parameter, that can be used to identify the newly created Real Application Security application session in future calls, until the cookie value is changed or the session is destroyed.
CURRENT_XS_USER	The name of the Real Application Security session application user whose privileges are currently active.
CURRENT_XS_USER_GUID	The identifier of the Real Application Security session application user whose privileges are currently active.
INACTIVITY_TIMEOUT	The specified inactivity timeout value in minutes for the current application session.
LAST_ACCESS_TIME	The last time the session was accessed by a session application user.



Table 10-2 (Cont.) Predefined Parameters

Parameter	Return Value
LAST_AUTHENTICATION_TIME	The last time the session application user was authenticated.
LAST_UPDATED_BY	The last time the application session was updated.
PROXY_GUID	Identifier of the Real Application Security session application user who opened the current session on behalf of SESSION_XS_USER.
SESSION_ID	The session identifier for the application session.
SESSION_XS_USER	The name of the Real Application Security session application user at logon.
SESSION_XS_USER_GUID	The identifier of the Real Application Security session application user at logon.
USERNAME	The session application user name.
USER_ID	The identifier of the session application user.

To retrieve the name of the currently attached Real Application Security session application user, you can use the following form of the XS SYS CONTEXT function:

```
XS_SYS_CONTEXT('XS$SESSION', 'SESSION_XS_USER')
```

The function returns <code>NULL</code> if no Real Application Security session is currently attached to the database session. The function returns the currently attached Real Application Security session application user even if it is called from within the body of a definer's rights unit, like a definer's rights view.

To retrieve the identifier (ID) for the currently attached Real Application Security session application user, you can use the following form of the XS SYS CONTEXT function:

```
XS_SYS_CONTEXT('XS$SESSION', 'SESSION_XS_USER_GUID')
```

The function returns <code>NULL</code> if no Real Application Security session is currently attached to the database session. The function returns the currently attached Real Application Security session application user ID even if it is called from within the body of a definer's rights unit, like a definer's rights view.

#### **Syntax**

```
XS_SYS_CONTEXT (
namespace IN VARCHAR2
attribute IN VARCHAR2)
RETURN VARCHAR2;
```

Parameter	Description
namespace	The name of the application context. You can specify either a string or an expression.
	To find information about the namespaces and attributes for the current application session, query the V\$XS_SESSION_NS_ATTRIBUTES data dictionary view.



Parameter	Description
attribute	A parameter within the namespace application context.

### **Example**

SELECT XS SYS CONTEXT ('XS\$SESSION', 'SESSION ID') FROM DUAL;

# 10.3 ORA CHECK ACL Function

The ORA\_CHECK\_ACL function checks whether an application user has the queried application privileges according to a list of ACLs. Oracle Database uses this function automatically when the application user runs a query on a table that has data security policy enabled. If the specified application privileges have been granted to the application user, ORA\_CHECK\_ACL returns 1. If they are not granted to the application user, then it returns 0.

### **Syntax**

#### **Parameters**

Parameter	Description
acls	RAW list of ACL ids of 8 byte. The maximum number of acls allowed is 250.
privileges	The application privilege names being checked. The maximum number of application privileges allowed is 100.

### **Examples**

The following example uses <code>ORA\_CHECK\_ACL</code> to check whether the application user has been granted the <code>P1</code> and <code>P2</code> application privileges in the <code>ACL1</code> ACL.

SELECT ORA CHECK ACL(TO ACLID('ACL1'), 'P1', 'P2') INTO ACLRESULT FROM DUAL;

# 10.4 ORA GET ACLIDS Function

The ORA\_GET\_ACLIDS function returns a list of ACL IDS associated with an object instance of data security policy enabled tables for the current application user. Oracle Database evaluates every dynamic data realm constraint rule, because ORA\_GET\_ACLIDS captures all ACL identifiers that are associated with the matching data realm constraints, if access to the current row has been granted. If the data realm constraints are from detail tables in a master-detail relationship, ORA\_GET\_ACLIDS retrieves the ACL identifiers from the master table as well as the detail table. If multiple data security policies have been applied to a table, ORA\_GET\_ACLIDS returns the ACLs associated with each policy.



### **Syntax**

```
ORA_GET_ACLIDS (
  table_alias IN VARCHAR2,
  (privileges IN VARCHAR(128))+)
RETURN RAW;
```

#### **Parameters**

Parameter	Description
table_alias	Table or view object alias in the query from a clause.
	Ensure that the table is XDS-enabled. To do so, query the DBA_XS_APPLIED_POLICIES data dictionary view.
	If you specify a view that is resolved to XDS-enabled tables, and if there are more than one XDS-enabled tables in the view, then Oracle Database only returns one of the tables.
privileges	The application privilege names that are associated with the returned ACL identifiers. The maximum number of application privileges allowed is 100.

### Example

```
SELECT ORA GET ACLIDS(t, 'SELECT', 'VIEW LOC') from SCOTT.DEPT t;
```

# 10.5 ORA\_CHECK\_PRIVILEGE Function

The <code>ORA\_CHECK\_PRIVILEGE</code> function checks whether the specified privileges have been granted to an application user. If the specified privileges have been granted to the application user, <code>ORA\_CHECK\_PRIVILEGE</code> returns 1. This function only works for system privileges, such as <code>CREATE\_SESSION</code>. If the system privileges are not granted to the application user, then it returns 0.

### **Syntax**

```
ORA_CHECK_PRIVILEGE(
  (privs IN VARCHAR(128))+)
return NUMBER;
```

#### **Parameters**

Parameter	Description
privs	The privilege names being checked. The maximum number of privileges allowed is 100.

### **Examples**

The following example uses <code>ORA\_CHECK\_PRIVILEGE</code> to check whether the application user has been granted the <code>CREATE SESSION</code> system privilege.

```
SELECT ORA_CHECK_PRIVILEGE('CREATE_SESSION') FROM DUAL;
```



# 10.6 TO\_ACLID Function

The  ${\tt TO\_ACLID}$  function returns the ACL IDs of the ACL names supplied to it.

### **Syntax**

TO\_ACLID(
 (acls IN VARCHAR(128))+)
return NUMBER;

### **Parameters**

Parameter	Description
acls	The ACL names whose ACL IDs are returned.

### **Examples**

The following example uses the  ${\tt TO\_ACLID}$  function to return the ACL ID for ACL1.

SELECT ORA\_CHECK\_ACL(TO\_ACLID('ACL1'), 'P1', 'P2') INTO ACLRESULT FROM DUAL;



11

# Oracle Database Real Application Security PL/SQL Packages

This chapter describes the PL/SQL packages that are available with Oracle Database Real Application Security.

Table 11-1 lists these packages. Detailed information on each package follows this table.

Table 11-1 Oracle Database Real Application Security PL/SQL Packages

PL/SQL Package	Description
DBMS_XS_SESSIONS Package	Includes subprograms to manage an application session.
XS_ACL Package	Includes subprograms to create, manage, and delete Access Control Lists (ACLs) and to add and remove parameter values.
XS_ADMIN_UTIL Package	Includes helper subprograms.
XS_DATA_SECURITY Package	Includes subprograms to create, manage, and delete data security policies, associated data realm constraints, column constraints, and ACL parameters.
XS_DATA_SECURITY_UTIL Package	Includes subprograms to schedule automatic refreshment for static ACL to a user table and change the ACL refreshment mode to on-commit or on-demand refresh.
XS_DIAG Package	Includes subprograms to diagnose potential problems in Real Application Security objects and report identified inconsistencies.
XS_NAMESPACE Package	Includes subprograms to create, manage, and delete namespace templates and attributes.
XS_PRINCIPAL Package	Includes subprograms to create, manage, and delete application users and roles.
XS_SECURITY_CLASS Package	Includes subprograms to create, manage, and delete security classes and their privileges. Also includes subprograms for managing security class inheritance.

This section describes the following Oracle Database Real Application Security PL/SQL packages:

# 11.1 DBMS\_XS\_SESSIONS Package

The DBMS XS SESSIONS package manages an application session.

This section includes the following topics:

- Security Model
- Constants
- Object Types, Constructor Functions, Synonyms, and Grants



Summary of DBMS\_XS\_SESSIONS Subprograms

### 11.1.1 Security Model

The <code>DBMS\_XS\_SESSIONS</code> package is created in the <code>SYS</code> schema. The privilege to execute the package is granted to <code>PUBLIC</code>. The executing user must have the appropriate privilege for the particular operation.

### 11.1.2 Constants

The following constants define operation codes passed into namespace event handling functions:

```
attribute_first_read_operation CONSTANT PLS_INTEGER := 1;
modify attribute operation CONSTANT PLS INTEGER := 2;
```

The following constants represent bit values that identify events of interest for a particular attribute in a namespace that has an event handling function:

The following constants define return codes that can be returned by a namespace event handling function:

The following constants are used as input into the <code>ADD\_GLOBAL\_CALLBACK</code>, <code>DELETE GLOBAL CALLBACK</code>, and <code>ENABLE GLOBAL CALLBACK</code> procedures:

```
create_session_event
attach_session_event
constant PLS_INTEGER := 1;
guest_to_user_event
proxy_to_user_event
constant PLS_INTEGER := 3;
proxy_to_user_event
constant PLS_INTEGER := 4;
revert_to_user_event
constant PLS_INTEGER := 5;
enable_role_event
disable_role_event
constant PLS_INTEGER := 6;
disable_dynamic_role_event
constant PLS_INTEGER := 7;
enable_dynamic_role_event
disable_dynamic_role_event
constant PLS_INTEGER := 8;
detach_session_event
constant PLS_INTEGER := 10;
terminate_session_event
direct_login_event
constant PLS_INTEGER := 11;
direct_logoff_event
constant PLS_INTEGER := 12;
direct_logoff_event
constant PLS_INTEGER := 13;
```

### 11.1.3 Object Types, Constructor Functions, Synonyms, and Grants

The following object types, constructor functions, synonyms, and GRANT statements are defined for this package.

```
CREATE OR REPLACE TYPE DBMS_XS_NSATTR AS OBJECT (
--- Member variables
namespace varchar2(130),
attribute varchar2(4000),
attribute_value varchar2(4000),
--- Constructor for DBMS_XS_NSATTR type
--- Only namespace name is mandatory
```



CONSTRUCTOR FUNCTION DBMS XS NSATTR(

namespace IN VARCHAR2, attribute IN VARCHAR2 DEFAULT NULL, attribute value IN VARCHAR2 DEFAULT NULL) RETURN SELF AS RESULT);

CREATE OR REPLACE PUBLIC SYNONYM DBMS XS NSATTR FOR SYS.DBMS XS NSATTR; CREATE OR REPLACE TYPE DBMS XS NSATTRLIST AS VARRAY(1000) OF DBMS XS NSATTR; CREATE OR REPLACE PUBLIC SYNONYM DBMS XS NSATTRLIST FOR SYS.DBMS XS NSATTRLIST; GRANT EXECUTE ON DBMS XS NSATTR TO PUBLIC; GRANT EXECUTE ON DBMS XS NSATTRLIST TO PUBLIC; CREATE OR REPLACE PUBLIC SYNONYM DBMS XS SESSIONS FOR SYS.DBMS XS SESSIONS; GRANT EXECUTE ON DBMS XS SESSIONS TO PUBLIC;

## 11.1.4 Summary of DBMS\_XS\_SESSIONS Subprograms

Table 11-2 Summary of DBMS\_XS\_SESSIONS Subprograms

Subprogram	Description
CREATE_SESSION Procedure	Creates a new application session for the specified application user name.
ATTACH_SESSION Procedure	Attaches the current traditional database session to the application session identified by the session ID.
ASSIGN_USER Procedure	Assigns a named user to the currently attached anonymous Real Application Security session.
SWITCH_USER Procedure	Switches the application user in the currently attached session.
CREATE_NAMESPACE Procedure	Creates a new application namespace in the currently attached application session.
CREATE_ATTRIBUTE Procedure	Creates a new custom attribute for the specified application namespace in the currently attached application session.
SET_ATTRIBUTE Procedure	Sets a new value for the specified attribute in the namespace in the currently attached application session.
GET_ATTRIBUTE Procedure	Gets the value of an attribute in the namespace in the currently attached application session.
RESET_ATTRIBUTE Procedure	Resets an application namespace attribute to its original value in the specified namespace in the currently attached application session.
DELETE_ATTRIBUTE Procedure	Deletes the specified attribute from the specified namespace in the currently attached application session.
DELETE_NAMESPACE Procedure	Deletes the specified namespace and its attributes from the currently attached application session.
ENABLE_ROLE Procedure	Enables a real application role in the currently attached application session.
DISABLE_ROLE Procedure	Disables a real application role from the currently attached application session.
SET_SESSION_COOKIE Procedure	Sets a new cookie value with the specified session ID.
REAUTH_SESSION Procedure	Updates the last authentication time for the session identified by specified session ID.
SET_INACTIVITY_TIMEOUT Procedure	Sets an inactivity timeout value, in minutes, for the specified session.



Table 11-2 (Cont.) Summary of DBMS\_XS\_SESSIONS Subprograms

Subprogram	Description
SAVE_SESSION Procedure	Saves or persists the changes performed in the currently attached session.
DETACH_SESSION Procedure	Detaches the current traditional database session from the application session to which it is attached.
DESTROY_SESSION Procedure	Destroys or terminates the session specified by the session ID.
ADD_GLOBAL_CALLBACK Procedure	Registers an existing event handler with the database.
ENABLE_GLOBAL_CALLBACK Procedure	Enables or disables the global callback for the session event specified by the event_type parameter.
DELETE_GLOBAL_CALLBACK Procedure	Deletes an existing global callback association.

This section describes the following DBMS XS SESSIONS subprograms:

### 11.1.4.1 CREATE\_SESSION Procedure

The <code>CREATE\_SESSION</code> procedure creates a new application session for the specified user name. It returns a session identifier that you can use to reference the session in future calls.

The session can be created with a regular application user or an external application user. The session can be created in trusted mode or secure mode. In trusted mode, data security checks are bypassed; in secure mode, they are enforced.

The combination of regular session in trusted mode is not supported. Other combinations, regular session in secure mode, external session in trusted mode, or external session in secure mode are supported.

The namespaces parameter is a list of triplet namespaces to be created, the attribute to be created, and the attribute value to be set. This is an optional parameter. The default value is <code>NULL</code>. The <code>XS\$GLOBAL\_VAR</code> and <code>XS\$SESSION</code> namespaces and their attributes are always available to the session.

This function does not attach the current traditional session to the newly created application session. Use the ATTACH\_SESSION Procedure to perform this task.

The user executing the procedure must have the <code>CREATE\_SESSION</code> application privilege for the application user specified by the <code>username</code> parameter. You can also specify a list of namespaces to be created when the session is created. If you specify namespaces during creation of the session, the caller is required to be granted application privileges <code>MODIFY\_NAMESPACE</code> or <code>MODIFY\_ATTRIBUTE</code> on the namespaces, or be granted the <code>ADMIN\_NAMESPACE</code> system privilege.

```
CREATE_SESSION (
username IN VARCHAR2,
sessionid OUT NOCOPY RAW,
is_external IN BOOLEAN DEFAULT FALSE,
is_trusted IN BOOLEAN DEFAULT FALSE,
namespaces IN DBMS_XS_NSATTRLIST DEFAULT NULL,
cookie IN VARCHAR2 DEFAULT NULL);
```



Parameter	Description
username	The name of a regular application user or an external application user for which to create the application session.
	To find a listing of the user names and application roles for the current session, query the DBA_XS_USERS data dictionary view. To find all application users and roles, query the DBA_XS_PRINCIPALS data dictionary view as follows:
	Users:
	SELECT NAME FROM DBA_XS_USERS;
	Roles:
	SELECT NAME FROM DBA_XS_ROLES;
	SELECT NAME FROM DBA_XS_DYNAMIC_ROLES;
sessionid	Session ID of the newly created application session. You can get the session ID by using one of the following methods:
	• SELECT XS_SYS_CONTEXT('XS\$SESSION', 'SESSION_ID') FROM DUAL;
	• Using the DBMS_XS_SESSIONS.GET_ATTRIBUTE procedure.
is_external	Specifies whether the session is to be created as an external principal session. This is an optional parameter. The default value is FALSE, indicating that a regular session is to be created. A NULL value is taken to mean FALSE.
is_trusted	Specifies if the session is to be created in trusted mode or secure mode. In trusted mode, data security checks are bypassed; in secure mode, they are enforced. This is an optional parameter. The default value is FALSE, indicating secure mode. A NULL value is taken to mean FALSE.
namespaces	The list of name, attribute, and attribute value triplet. If the namespace is not accessible to the session or no such namespace template exists, an error is thrown.
cookie	Specifies the server cookie to be set for the session. This is an optional parameter. The default value is <code>NULL</code> . The maximum allowed length of the cookie is 1024 bytes.

### **Examples**

```
DECLARE
   nsList DBMS_XS_NSATTRLIST;
   sessionid RAW(16);
BEGIN
   nsList := DBMS_XS_NSATTRLIST(DBMS_XS_NSATTR('ns1'), DBMS_XS_NSATTR('ns2'));
   SYS.DBMS_XS_SESSIONS.CREATE_SESSION('lwuser1', sessionid, FALSE, FALSE, nsList);
END;
```

### 11.1.4.2 ATTACH\_SESSION Procedure

The ATTACH\_SESSION procedure attaches the current traditional database session to the application session identified by the session ID (session\_id). The attached session enables the roles granted (directly or indirectly) to the application user with which the session was created and the session scope dynamic application roles that were enabled until the last

detach of this session. If you execute ATTACH\_SESSION with a list of dynamic application roles using the optional parameter <code>enable\_dynamic\_roles</code>, the provided dynamic application roles are enabled for the session. To disable a list of dynamic roles, specify the list using the optional parameter <code>disable\_dynamic\_roles</code>.

You can specify a list of triplet values (namespace, attribute, attribute value) during the attach operation. The namespaces and attributes are then created and attribute values set. This is in addition to any namespaces and attributes that were present in the session.

To execute this procedure, the traditional session user must have the ATTACH\_SESSION application privilege. If you specify namespaces, then the user is required to be granted application privileges MODIFY\_NAMESPACE or MODIFY\_ATTRIBUTE on the namespaces, or ADMIN NAMESPACE system privilege.

A self password change is allowed using the SQL\*Plus PASSWORD command if invoked from an explicitly attached session (a session attached using the ATTACH\_SESSION procedure or the attachSession() method in Java), provided that session has the ALTER USER privilege and the user name is provided with the PASSWORD command.

#### **Syntax**

```
ATTACH_SESSION (

sessionid IN RAW,

enable_dynamic_roles IN XS$NAME_LIST DEFAULT NULL,

disable_dynamic_roles IN XS$NAME_LIST DEFAULT NULL,

external_roles IN XS$NAME_LIST DEFAULT NULL,

authentication_time IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,

namespaces IN DBMS_XS_NSATTRLIST DEFAULT NULL);
```

Parameter	Description
sessionid	Session ID of the application session. You can get the session ID by using one of the following methods:  SELECT XS_SYS_CONTEXT('XS\$SESSION',
enable_dynamic_roles	A list of dynamic roles to be granted to be enabled in the application session. This is an optional parameter. If any of the dynamic roles specified does not exist, the attach session fails. If the session is an external principal session, a list of external roles can be specified for enabling. These roles will remain enabled until detach and will not be enabled in the next attach by default.
	To find a listing of the application roles for the current session, query the DBA_XS_SESSION_ROLES data dictionary view. To find a listing of all dynamic application roles, query the DBA_XS_PRINCIPALS data dictionary view as follows:
	SELECT NAME, TYPE FROM DBA_XS_PRINCIPALS;
disable_dynamic_roles	A list of dynamic roles to be disabled from the session. This is an optional parameter.



Parameter	Description
external_roles	A list of external roles if the session is an external principal session. This is an optional parameter. These external roles remain enabled until a detach operation and are not enabled again in the next attach by default.
authentication_time	The updated authentication time for the session. This is an optional parameter. The time must be specified in the following format:  YYYY-MM-DD HH:MI:SS.FF TZR
namespaces	The list of name, attribute, and attribute value triplet. If the namespace is not accessible to the session or no such namespace template exists, an error is thrown.

#### **Examples**

```
DECLARE
  nsList DBMS_XS_NSATTRLIST;
  sessionid RAW(16);
BEGIN
   nsList := DBMS_XS_NSATTRLIST(DBMS_XS_NSATTR('ns1'), DBMS_XS_NSATTR('ns2'));
   SYS.DBMS_XS_SESSIONS.CREATE_SESSION('lwuser1', sessionid);
   SYS.DBMS_XS_SESSIONS.ATTACH_SESSION(sessionid, NULL, NULL, NULL, nsList);
END:
```

### 11.1.4.3 ASSIGN USER Procedure

The ASSIGN\_USER procedure assigns a named application user to the currently attached anonymous application session.

Roles enabled in the current session are retained after this operation. The optional parameters <code>enable\_dynamic\_roles</code> and <code>disable\_dynamic\_roles</code> specify the additional lists of dynamic roles to be enabled or disabled. If the assigned user is external, you can specify a list of external roles to be enabled.

You can specify a list of triplet values (namespace, attribute, attribute value) during the assign operation. The namespaces and attributes are then created in the session and attribute values set. This is in addition to any namespaces and attributes that were already present in the session.

To execute this procedure, the dispatcher or connection user must have the <code>ASSIGN\_USER</code> application privilege. If you specify namespaces, then the user is required to be granted application privileges <code>MODIFY\_NAMESPACE</code> or <code>MODIFY\_ATTRIBUTE</code> on the namespaces, or <code>ADMIN\_NAMESPACE</code> system privilege.



Parameter	Description
username	The name of the real application user.
	To find a listing of existing application users, query the DBA_XS_PRINCIPALS data dictionary view as follows:
	SELECT NAME FROM DBA_XS_PRINCIPALS;
is_external	Specifies whether the named application user is an external user. This is an optional parameter. The default value is FALSE, indicating that a regular application user is assigned. A NULL value is taken to mean FALSE.
enable_dynamic_roles	A list of dynamic roles to be enabled in an application session. This is an optional parameter.
	To find a listing of the application roles for the current session, query the V\$XS_SESSION_ROLES data dictionary view. To find a listing of all dynamic application roles, query the DBA_XS_DYNAMIC_ROLES data dictionary view as follows:
	SELECT NAME FROM DBA_XS_DYNAMIC_ROLES;
disable_dynamic_roles	A list of dynamic roles to be disabled from the session. This is an optional parameter.
external_roles	A list of external roles if the application user is an external application user. This is an optional parameter.
authentication_time	The updated authentication time for the session. This is an optional parameter. The time must be specified in the following format:
	YYYY-MM-DD HH:MI:SS.FF TZR
namespaces	The list of name, attribute, and attribute value triplet. If the namespace is not accessible to the session or no such namespace template exists, an error is thrown.

### **Examples**

### 11.1.4.4 SWITCH USER Procedure

The  ${\tt SWITCH\_USER}$  procedure switches the application user in the currently attached session. The current application user must be a proxy user for the target application user before performing the switch operation by using the

XS\_PRINCIPAL.ADD\_PROXY\_USER PL/SQL API to acquire the proxy of another



application user privilege. The list of filtering application roles of the target user gets enabled in the session.

You can retain current application namespaces of the session or discard them. You can also specify a list of namespaces to be created and attribute values to be set after the switch. If you specify namespaces, then the user is required to be granted application privileges MODIFY\_NAMESPACE or MODIFY\_ATTRIBUTE on the namespaces, or ADMIN\_NAMESPACE system privilege.

### **Syntax**

```
SWITCH_USER (

username IN VARCHAR2,

keep_state IN BOOLEAN DEFAULT FALSE,

namespaces IN DBMS_XS_NSATTRLIST DEFAULT NULL);
```

#### **Parameters**

Parameter	Description
username	User name of the user whose security context you want to switch to.
	To find a listing of existing application users, query the DBA_XS_USERS data dictionary view as follows:
	SELECT NAME FROM DBA_XS_USERS;
keep_state	Controls whether application namespaces are retained.
	Possible values are:
	<ul> <li>TRUE: Sets all other session states to remain unchanged.</li> </ul>
	<ul> <li>FALSE: Clears the previous state in the session. The default value.</li> </ul>
namespaces	The list of name, attribute, and attribute value triplet. If the namespace is not accessible to the session or no such namespace template exists, an error is thrown.

### **Examples**

### 11.1.4.5 CREATE\_NAMESPACE Procedure

The CREATE\_NAMESPACE procedure creates a new namespace in the currently attached application session. The namespace template corresponding to the namespace must exist in the system, else this operation throws an error. After this operation, the namespace along with its attributes as they are created in the template are available to the session.

The calling user must have the MODIFY NAMESPACE application privilege.



### **Syntax**

### **Parameters**

Parameter	Description
namespace	The name of the namespace to create. There must be an existing namespace template document with this name. The maximum size of the case sensitive character string is 128 characters.
	To find a listing of existing namespaces for the current session, once attached, query the V\$XS_SESSION_NS_ATTRIBUTES data dictionary view. You can query the DBA_XS_SESSION_NS_ATTRIBUTES data dictionary view to find out all the namespaces in all application sessions.
	You can query the DBA_XS_NS_TEMPLATES and DBA_XS_NS_TEMPLATE_ATTRIBUTES data dictionary views for a list of namespace templates and attributes.

### **Examples**

```
BEGIN
   SYS.DBMS_XS_SESSIONS.CREATE_NAMESPACE('J_NS1');
END;
```

## 11.1.4.6 CREATE\_ATTRIBUTE Procedure

The CREATE\_ATTRIBUTE procedure creates a new custom attribute in the specified namespace in the currently attached application session. If the namespace is not already available in the session or no such namespace templates exist, an error is thrown.

The calling user is required to be granted the MODIFY ATTRIBUTE application privilege.

### **Syntax**

```
PROCEDURE create_attribute(
namespace IN VARCHAR2,
attribute IN VARCHAR2,
value IN VARCHAR2 DEFAULT NULL,
eventreg IN PLS INTEGER DEFAULT NULL);
```

Parameter	Description
namespace	The namespace in which the attribute gets created. If the namespace does not exist in the session, an error is thrown. The maximum size of the case sensitive character string is 128 characters.
attribute	The name of the attribute to be created. The maximum size of the case sensitive character string is 4000 characters.
value	The default value for the attribute. The maximum size of the case sensitive character string is 4000 characters.



### Description **Parameter** eventreg The event for which the handler is executed for the attribute. This is an optional parameter. This parameter can take the following values: DBMS XS SESSIONS.attribute\_first\_read\_event The handler function is called whenever an attribute get request is received and the value for the attribute has not been set. This event can be registered only if the default value is set to NULL. This value corresponds with the FIRSTREAD EVENT constant in the XS NAMESPACE package or Admin API. DBMS XS SESSIONS.modify attribute event: The handler function is called whenever an attribute set request is received. This value corresponds with the UPDATE EVENT constant in the in the XS\_NAMESPACE package or Admin API. If the attribute is registered for first read event, then the handler is executed if the attribute is uninitialized, before returning the value. If the update event is registered, the handler gets called whenever the attribute is modified. Events can be registered only if the namespace has an event handler, else an error is thrown.

### **Examples**

### 11.1.4.7 SET ATTRIBUTE Procedure

The SET\_ATTRIBUTE procedure sets a new value for the specified attribute in the namespace associated with the currently attached session. The handler function is called if the update event is set for the attribute. If the namespace does not exist or is deleted, an error is thrown. If there is no template corresponding to the namespace that exists, an error is thrown.

The calling user is required to be granted the MODIFY ATTRIBUTE application privilege.

```
SET_ATTRIBUTE (
namespace IN VARCHAR2,
attribute IN VARCHAR2,
value IN VARCHAR2);
```



Parameter	Description
namespace	Name of the namespace associated with the attribute. The maximum size of the case sensitive character string is 128 characters.
	To find a listing of existing namespaces for the current session, once attached, query the V\$XS_SESSION_NS_ATTRIBUTES data dictionary view. You can query the DBA_XS_SESSION_NS_ATTRIBUTES data dictionary view to find out all the namespaces in all application sessions.
	You can query the DBA_XS_NS_TEMPLATES and DBA_XS_NS_TEMPLATE data dictionary views for a list of namespace templates and attributes.
attribute	Name of an existing attribute in an existing namespace.
	To find a listing of existing namespace attributes, query the V\$XS_SESSION_NS_ATTRIBUTES data dictionary view.
value	New value for the attribute. The maximum size of the case sensitive character string is 4000 characters.
	To find an listing of existing values associated with the attribute, query the V\$XS_SESSION_NS_ATTRIBUTES data dictionary view.

### **Examples**

```
BEGIN
SYS.DBMS_XS_SESSIONS.SET_ATTRIBUTE('J_NS','JohnNSAttr1','John bio');
END:
```

### 11.1.4.8 GET ATTRIBUTE Procedure

The GET\_ATTRIBUTE procedure gets the value of the specified attribute in the namespace in the currently attached session. If no template corresponding to the namespace exists, an error is thrown. If the specified attribute does not exist, it returns empty string.

If the attribute value is <code>NULL</code>, the <code>firstRead</code> event is set, and it is the first time that the attribute value is being fetched, then the handler function for the attribute is called.

The calling user is not required to be granted any privileges.

```
GET_ATTRIBUTE (
namespace IN VARCHAR2,
attribute IN VARCHAR2,
value OUT NOCOPY VARCHAR2);
```



Parameter	Description
namespace	The namespace of the attribute to retrieve. The maximum size of the case sensitive character string is 128 characters.
	To find a listing of existing namespaces for the current session, once attached, query the V\$XS_SESSION_NS_ATTRIBUTES data dictionary view. You can query the DBA_XS_SESSION_NS_ATTRIBUTES data dictionary view to find out all the namespaces in all application sessions.
attribute	The name of the attribute to retrieve. The maximum size of the case sensitive character string is 4000 characters. To find a listing of available attributes, query the V\$XS_SESSION_NS_ATTRIBUTES data dictionary view.
value	The value of the attribute to retrieve.
	To find a listing of available attribute values, query the V\$XS_SESSION_NS_ATTRIBUTES data dictionary view.

### **Examples**

```
DECLARE
attrVal VARCHAR2(4000);

BEGIN
SYS.DBMS_XS_SESSIONS.GET_ATTRIBUTE('J_NS1','JohnNS1Attr1',attrVal);
END:
```

### 11.1.4.9 RESET\_ATTRIBUTE Procedure

The RESET\_ATTRIBUTE procedure resets the value of an attribute to its default value (if present) or to NULL in the namespace in the current attached session. If the attribute has a default value specified, then the value is reset to the default value. If the attribute was created without a default value and marked for the attribute\_first\_read\_event, then the value is set to NULL and the attribute is marked as uninitialized. If the attribute was created without a default value and not marked for the attribute\_first\_read\_event, then the value is set to NULL.

The calling user is required to be granted the MODIFY ATTRIBUTE application privilege.

### **Syntax**

```
PROCEDURE reset_attribute(
namespace IN VARCHAR2,
attribute IN VARCHAR2);
```

Parameter	Description
namespace	The name of the namespace containing the attribute. The maximum size of the case sensitive character string is 128 characters.
attribute	The name of the attribute to be reset. The maximum size of the case sensitive character string is 4000 characters.



### **Examples**

```
BEGIN
   SYS.DBMS_XS_SESSIONS.RESET_ATTRIBUTE('ns2','attr1');
END;
```

### 11.1.4.10 DELETE\_ATTRIBUTE Procedure

The DELETE\_ATTRIBUTE procedure deletes the specified attribute and its associated value from the specified namespace in the currently attached session. Only custom attributes can be deleted. Template attributes cannot be deleted. If the specified attribute does not exist, an error is thrown.

The calling application is required to be granted the <code>MODIFY\_ATTRIBUTE</code> application privilege.

### **Syntax**

```
DELETE_ATTRIBUTE (
namespace IN VARCHAR2,
attribute IN VARCHAR2);
```

#### **Parameters**

Parameter	Description
namespace	The namespace associated with the attribute to delete. The maximum size of the case sensitive character string is 128 characters.
	To find a listing of existing namespaces for the current session, once attached, query the V\$XS_SESSION_NS_ATTRIBUTES data dictionary view. You can query the DBA_XS_SESSION_NS_ATTRIBUTES data dictionary view to find out all the namespaces in all application sessions.
	You can query the DBA_XS_NS_TEMPLATES and DBA_XS_NS_TEMPLATE data dictionary views for a list of namespace templates and attributes.
attribute	The attribute to delete.  To find a listing of existing namespaces for the current session, once attached, query the V\$XS_SESSION_NS_ATTRIBUTES data dictionary view. You can query the DBA_XS_SESSION_NS_ATTRIBUTES data dictionary view to find out all the namespaces in all application sessions.

### **Examples**

```
BEGIN
   SYS.DBMS_XS_SESSIONS.DELETE_ATTRIBUTE('JohnNS1','JohnNS1Attr1');
END;
```

### 11.1.4.11 DELETE\_NAMESPACE Procedure

The DELETE\_NAMESPACE procedure deletes a namespace and its attributes from the currently attached application session.

The calling user must have the MODIFY NAMESPACE application privilege.



### **Syntax**

### **Parameters**

Parameter	Description
namespace	The name of the namespace to delete. The maximum size of the case sensitive character string is 128 characters.
	To find a listing of existing namespaces for the current session, once attached, query the V\$XS_SESSION_NS_ATTRIBUTES data dictionary view. You can query the DBA_XS_SESSION_NS_ATTRIBUTES data dictionary view to find out all the namespaces in all application sessions.
	You can query the DBA_XS_NS_TEMPLATES and DBA_XS_NS_TEMPLATE_ATTRIBUTES data dictionary views for a list of namespace templates and attributes.

### **Examples**

```
BEGIN
   SYS.DBMS_XS_SESSIONS.DELETE_NAMESPACE('JohnNS1');
END;
```

### 11.1.4.12 ENABLE\_ROLE Procedure

The <code>ENABLE\_ROLE</code> procedure enables a real application role in the currently attached application session. If the role is already enabled, then <code>ENABLE\_ROLE</code> procedure performs no action. This procedure can only enable a regular application role directly granted to the current application user. You cannot enable dynamic application roles.

This operation does not require the calling user to have any additional privilege.

### **Syntax**

```
ENABLE_ROLE (
  role     IN VARCHAR2);
```

Parameter	Description	
role	The name of the role to enable. The maximum size of the case sensitive character string is 128 characters.	
	To find a listing of the application roles for the current session, query the V\$XS_SESSION_ROLES data dictionary view. To find all application roles, query the DBA_XS_SESSION_ROLES data dictionary view as follows:	
	SELECT ROLE_NAME FROM V\$XS_SESSION_ROLES;	
	SELECT SESSIONID, ROLE FROM DBA_XS_SESSION_ROLES;	



### **Examples**

```
BEGIN
   SYS.DBMS_XS_SESSIONS.ENABLE_ROLE('auth2_role');
END:
```

### 11.1.4.13 DISABLE ROLE Procedure

The DISABLE\_ROLE procedure disables a real application role from the specified application session. If the role is already disabled or not enabled in the currently attached application session, then DISABLE\_ROLE performs no action. You cannot disable dynamic application roles. You can only disable a regular application role, which is directly granted to the application user with which the session is created.

This operation does not require the calling user to have any additional privilege.

### **Syntax**

```
DISABLE_ROLE (
  role    IN VARCHAR2);
```

#### **Parameters**

Parameter	Description
role	The name of the role to disable. The maximum size of the case sensitive character string is 128 characters.
	To find a listing of the application roles for the current session, query the V\$XS_SESSION_ROLES data dictionary view. To find all application roles, query the DBA_XS_SESSION_ROLES data dictionary view as follows:
	SELECT ROLE_NAME FROM V\$XS_SESSION_ROLES;
	SELECT SESSIONID, ROLE FROM DBA_XS_SESSION_ROLES;

### **Examples**

```
BEGIN
   SYS.DBMS_XS_SESSIONS.DISABLE_ROLE('auth1_role');
END;
```

### 11.1.4.14 SET SESSION COOKIE Procedure

The SET\_SESSION\_COOKIE procedure sets a new cookie value with the specified session ID. If the specified session does not exist or the cookie name is not unique among all the user application sessions, then an error is thrown.

To execute this procedure, the user is required to be granted the MODIFY\_SESSION application privilege.

```
SET_SESSION_COOKIE (
cookie IN VARCHAR2,
sessionid IN RAW DEFAULT NULL);
```



Parameter	Description
cookie	A name for the new cookie. The maximum allowed length for the cookie is 1024 characters. Cookie names must be unique.
	To find a listing of existing cookies for the current session, query XS_SYS_CONTEXT (XS\$SESSION', 'COOKIE').
sessionid	Session ID of the application session. The default value is <code>NULL</code> . You can get the session ID by using one of the following methods:
	<ul> <li>SELECT XS_SYS_CONTEXT('XS\$SESSION', 'SESSION_ID') FROM DUAL;</li> <li>Using the DBMS_XS_SESSIONS.GET_ATTRIBUTE procedure.</li> <li>If you do not specify a session ID or enter NULL, then SET_SESSION_COOKIE uses the session ID of the current application session.</li> </ul>

### **Examples**

```
DECLARE
    sessionid RAW(16);
BEGIN
    SYS.DBMS_XS_SESSIONS.CREATE_SESSION('lwuserl', sessionid);
    SYS.DBMS_XS_SESSIONS.SET_SESSION_COOKIE('cookiel', sessionid);
END:
```

### 11.1.4.15 REAUTH\_SESSION Procedure

The REAUTH\_SESSION procedure updates the last authentication time for the specified session ID as the current time. Applications must call this procedure when it has reauthenticated an application user.

Use the REAUTH\_SESSION procedure to enable a role that has timed out because of a lack of recent authentication in the application or middle-tier server. You can also call the reauthSession Java method.

To execute this function, the user is required to be granted the  ${\tt MODIFY\_SESSION}$  application privilege.

### **Syntax**

```
REAUTH_SESSION (
  sessionid IN RAW DEFAULT NULL);
```

Parameter	Description
sessionid Session ID of the application session. This parameter is optional. The dis NULL. You can get the session ID by using one of the following method	
	<ul> <li>SELECT XS_SYS_CONTEXT('XS\$SESSION', 'SESSION_ID') FROM DUAL;</li> <li>Using the DBMS_XS_SESSIONS.GET_ATTRIBUTE procedure.</li> </ul>
	If you do not specify a session ID or enter <code>NULL</code> , then <code>REAUTH_SESSION</code> uses the session ID of the current application session.



### **Examples**

```
DECLARE
   sessionid RAW(16);
BEGIN
   SYS.DBMS_XS_SESSIONS.REAUTH_SESSION(sessionid);
END:
```

### 11.1.4.16 SET\_INACTIVITY\_TIMEOUT Procedure

The SET\_INACTIVITY\_TIMEOUT procedure sets an inactivity time-out value, in minutes, for the current attached session. The inactivity time-out value represents the maximum period of inactivity allowed before Oracle Database terminates the application session and the resource is reclaimed. Trying to set a negative value for the time parameter throws an error. If an invalid session ID is specified or the session does not exist, an error is thrown.

Another way to set the time-out value is to use the setInactivityTimeout Java method. You can set a default global time-out value in the xmlconfig.xml configuration file. Oracle recommends 240 (4 hours).

An application session cannot time-out due to inactivity while a traditional session is attached. The last access time is updated each time a traditional session attaches to the application session.

To execute this procedure, the calling user is required to be granted the MODIFY SESSION application privilege.

#### **Syntax**

```
SET_INACTIVITY_TIMEOUT (
time IN NUMBER,
sessionid IN RAW DEFAULT NULL);
```

#### **Parameters**

Parameter	Description	
time	Inactivity time-out value in minutes. Oracle recommends setting the time parameter to 240 (4 hours). A zero (0) value means the value is infinite and that the session never expires due to inactivity.	
sessionid	Session ID of the application session. The default value is NULL. You can get the session ID by using one of the following methods:	
	• SELECT XS_SYS_CONTEXT('XS\$SESSION', 'SESSION_ID') FROM DUAL;	
	• Using the DBMS XS SESSIONS.GET ATTRIBUTE procedure.	
	If you do not specify a session ID or enter NULL, then	
	SET_INACTIVITY_TIMEOUT uses the session ID of the current application session.	

#### **Examples**

```
DECLARE
  sessionid RAW(16);
BEGIN
  SYS.DBMS_XS_SESSIONS.CREATE_SESSION('lwuser1', sessionid);
```



```
SYS.DBMS_XS_SESSIONS.SET_INACTIVITY_TIMEOUT (300, sessionid);
END;
/
```

### 11.1.4.17 SAVE\_SESSION Procedure

The SAVE\_SESSION procedure saves all changes performed in the currently attached session and remains attached to the session as it was before saving changes.

The calling user requires no privileges to perform this operation.

### **Syntax**

SAVE SESSION;

#### **Parameters**

None.

### **Examples**

```
BEGIN
SYS.DBMS_XS_SESSIONS.SAVE_SESSION;
END:
```

### 11.1.4.18 DETACH SESSION Procedure

The DETACH\_SESSION procedure detaches the current traditional database session from the application session to which it is attached. The database sessions goes back to the context it was in prior to attaching to the application session. Any user can execute this procedure as the operation does not require any privileges to execute.

### **Syntax**

```
DETACH SESSION (abort IN BOOLEAN DEFAULT FALSE);
```

### **Parameters**

Parameter	Description
abort	If specified as TRUE, it rolls back the changes performed in the current session. If specified as FALSE, the default value, all changes performed in the session are persisted. If a NULL value is specified for this parameter, it is treated as FALSE.

### **Examples**

```
BEGIN
   SYS.DBMS_XS_SESSIONS.DETACH_SESSION;
END;
```

### 11.1.4.19 DESTROY\_SESSION Procedure

The DESTROY\_SESSION procedure destroys the specified session. This procedure also implicitly detaches all traditional sessions from the application session. After the session is destroyed no further attaches can be made to the session. This operation cannot destroy sessions created through direct logon of the application user.

To execute this procedure, the user must have the <code>TERMINATE\_SESSION</code> application privilege.

#### **Syntax**

```
DESTROY_SESSION (
sessionid IN RAW,
force IN BOOLEAN DEFAULT FALSE);
```

#### **Parameters**

Parameter	Description
sessionid	Session ID of the application session. You can get the session ID by using one of the following methods:
	<ul> <li>SELECT XS_SYS_CONTEXT('XS\$SESSION', 'SESSION_ID') FROM DUAL;</li> </ul>
	• Using the DBMS_XS_SESSIONS.GET_ATTRIBUTE procedure.
	If you do not specify a session ID or enter <code>NULL</code> , then <code>DESTROY_SESSION</code> uses the session ID of the current application session.
force	If set to FALSE, this operation throws an error, in case the specified session is currently attached. If set to $\mathtt{TRUE}$ , the currently attached application session can be destroyed. This is an optional parameter.

#### **Examples**

```
DECLARE
sessionid RAW(16);
BEGIN
SYS.DBMS_XS_SESSIONS.CREATE_SESSION('lwtSession1', sessionid);
SYS.DBMS_XS_SESSIONS.DESTROY_SESSION (sessionid);
END:
```

### 11.1.4.20 ADD\_GLOBAL\_CALLBACK Procedure

The ADD\_GLOBAL\_CALLBACK procedure registers an existing PL/SQL procedure as the event handler with the session operation specified by the <code>event\_type</code> parameter. You can add more than one event handler for the same session operation for execution when the associated event occurs. Adding the global callback procedure automatically enables the callback procedure for execution. If more than one callback is added for the same session event, they are executed in according to their registration sequence, that is, the callback procedure that was registered first, is executed first. This procedure throws an error if an invalid event type is specified or the callback procedure does not exist.

Successful execution of this procedure requires the CALLBACK application privilege. This role can be obtained through PROVISIONER database role.

```
ADD_GLOBAL_CALLBACK(
event_type IN PLS_INTEGER,
callback_schema IN VARCHAR2,
callback_package IN VARCHAR2,
callback_procedure IN VARCHAR2);
```



Parameter	Description
event_type	Select from the following event types:
	• CREATE_SESSION_EVENT
	• ATTACH_SESSION_EVENT
	CREATE_NAMESPACE_EVENT
	• GUEST_TO_USER_EVENT
	• PROXY_TO_USER_EVENT
	• REVERT_TO_USER_EVENT
	ENABLE_ROLE_EVENT
	• DISABLE_ROLE_EVENT
	<ul> <li>ENABLE_DYNAMIC_ROLE_EVENT</li> </ul>
	• DISABLE_DYNAMIC_ROLE_EVENT
	• DETACH_SESSION_EVENT
	• TERMINATE_SESSION_EVENT
	• DIRECT_LOGIN_EVENT
callback_schema	Enter the name of the schema in which the callback procedure was created.
callback_package	Enter the name of the package in which the callback procedure was created. If callback procedure is standalone, NULL should be passed as callback_package parameter. This parameter is optional only if the callback procedure is in a package.
callback_procedure	Enter the name of the procedure that defines the global callback.

### **Examples**

```
BEGIN

SYS.DBMS_XS_SESSIONS.ADD_GLOBAL_CALLBACK (

DBMS_XS_SESSIONS.CREATE_SESSION_EVENT,

'APPS1_SCHEMA','APPS2_PKG','CREATE_SESSION_CB');

END;
```

### 11.1.4.21 ENABLE GLOBAL CALLBACK Procedure

The <code>ENABLE\_GLOBAL\_CALLBACk</code> procedure enables or disables the global callback procedure for execution. If a callback procedure associated with this event is not specified, all callback procedures associated with this global callback are enabled or disabled. If an invalid event type is specified or invalid callback procedure is specified, an error is thrown.

```
ENABLE_GLOBAL_CALLBACK(

event_type IN PLS_INTEGER,
enable IN BOOLEAN DEFAULT TRUE,
callback_schema IN VARCHAR2 DEFAULT NULL,
callback_package IN VARCHAR2 DEFAULT NULL,
callback_procedure IN VARCHAR2 DEFAULT NULL);
```



Parameter	Description
event_type	Select from the following event types:
	• CREATE_SESSION_EVENT
	• ATTACH_SESSION_EVENT
	• CREATE_NAMESPACE_EVENT
	• GUEST_TO_USER_EVENT
	• PROXY_TO_USER_EVENT
	• REVERT_TO_USER_EVENT
	• ENABLE_ROLE_EVENT
	• DISABLE_ROLE_EVENT
	<ul> <li>ENABLE_DYNAMIC_ROLE_EVENT</li> </ul>
	• DISABLE_DYNAMIC_ROLE_EVENT
	• DETACH_SESSION_EVENT
	• TERMINATE_SESSION_EVENT
	• DIRECT_LOGIN_EVENT
enable	Specifies whether the global callback is to be enabled or disabled. The default value is TRUE, meaning enable.
callback_schema	Enter the name of the schema in which the global callback was created.
callback_package	Enter the name of the package in which the global callback was created.
callback_procedure	Enter the name of the procedure that defines the global callback.

### **Examples**

```
BEGIN

SYS.DBMS_XS_SESSIONS.ENABLE_GLOBAL_CALLBACK (

DBMS_XS_SESSIONS.CREATE_SESSION_EVENT,

TRUE, 'APPS1_SCHEMA','APPS2_PKG','CREATE_SESSION_CB');
```

### 11.1.4.22 DELETE\_GLOBAL\_CALLBACK Procedure

The <code>DELETE\_GLOBAL\_CALLBACk</code> procedure removes the global callback from registration. (It does not delete the global callback itself.) If a callback procedure is not specified, all callback procedures associated with this global callback are deleted. If an invalid event type is specified, an error is thrown.

```
DELETE_GLOBAL_CALLBACK(

event_type IN PLS_INTEGER,

callback_schema IN VARCHAR2 DEFAULT NULL,

callback_package IN VARCHAR2 DEFAULT NULL,

callback_procedure IN VARCHAR2 DEFAULT NULL);
```



Parameter	Description
event_type	Select from the following event types:
	• CREATE_SESSION_EVENT
	• ATTACH_SESSION_EVENT
	CREATE_NAMESPACE_EVENT
	• GUEST_TO_USER_EVENT
	• PROXY_TO_USER_EVENT
	• REVERT_TO_USER_EVENT
	ENABLE_ROLE_EVENT
	• DISABLE_ROLE_EVENT
	<ul> <li>ENABLE_DYNAMIC_ROLE_EVENT</li> </ul>
	<ul> <li>DISABLE_DYNAMIC_ROLE_EVENT</li> </ul>
	• DETACH_SESSION_EVENT
	• TERMINATE_SESSION_EVENT
	• DIRECT_LOGIN_EVENT
callback_schema	Enter the name of the schema in which the global callback was created.
callback_package	Enter the name of the package in which the global callback was created.
callback_procedure	Enter the name of the procedure that defines the global callback.

### **Examples**

```
BEGIN

SYS.DBMS_XS_SESSIONS.DELETE_GLOBAL_CALLBACK (

DBMS_XS_SESSIONS.CREATE_SESSION_EVENT,

'APPS1_SCHEMA','APPS2_PKG','CREATE_SESSION_CB');

END:
```

# 11.2 XS\_ACL Package

The XS ACL package creates procedures to create and manage Access Control Lists (ACLs).

This section includes the following topics:

- Security Model for the XS\_ACL Package
- Constants
- Object Types, Constructor Functions, Synonyms, and Grants
- Summary of XS\_ACL Subprograms

# 11.2.1 Security Model for the XS\_ACL Package

The XS ACL package is created under the SYS schema.

The DBA role is granted the ADMIN\_ANY\_SEC\_POLICY privilege, which allows it to administer schema objects like ACLs, security classes, and security policies across all schemas.

Users can administer schema objects in their own schema if they have been granted the RESOURCE role for the schema. The RESOURCE role and the XS\_RESOURCE application role include the ADMIN\_SEC\_POLICY privilege, required to administer schema objects in the schema as well as administering the policy artifacts within the granted schema to achieve policy management within an application.

Users can administer policy enforcement on the schema if they have been granted the APPLY\_SEC\_POLICY privilege. With this privilege, the user can administer policy enforcement within granted schemas to achieve policy management within an application.

### 11.2.2 Constants

The following constants define the parent ACL type:

```
EXTENDED CONSTANT PLS_INTEGER := 1;
CONSTRAINED CONSTANT PLS INTEGER := 2;
```

### The following constants define the principal's type:

```
PTYPE_XS CONSTANT PLS_INTEGER := 1;
PTYPE_DB CONSTANT PLS_INTEGER := 2;
PTYPE_DN CONSTANT PLS_INTEGER := 3;
PTYPE_EXTERNAL CONSTANT PLS_INTEGER := 4;
```

The following constants define the parameter's value type:

```
TYPE_NUMBER CONSTANT PLS_INTEGER := 1;
TYPE VARCHAR CONSTANT PLS_INTEGER := 2;
```

### 11.2.3 Object Types, Constructor Functions, Synonyms, and Grants

The following object types, constructor functions, synonyms, and GRANT statements are defined for this package.

```
MEMBER PROCEDURE set_grant(granted IN BOOLEAN),
MEMBER FUNCTION is_granted RETURN BOOLEAN,
MEMBER PROCEDURE set_inverted_principal(inverted IN BOOLEAN),
MEMBER FUNCTION is_inverted_principal RETURN BOOLEAN,
MEMBER PROCEDURE set_principal(principal_name IN VARCHAR2),
MEMBER FUNCTION get_principal RETURN VARCHAR2,
MEMBER PROCEDURE set_principal_type (principal_type IN PLS_INTEGER),
MEMBER FUNCTION get_principal_type RETURN PLS_INTEGER,
MEMBER PROCEDURE set_start_date(start_date IN TIMESTAMP WITH TIME ZONE),
MEMBER FUNCTION get_start_date RETURN TIMESTAMP WITH TIME ZONE,
MEMBER PROCEDURE set_end_date(end_date IN TIMESTAMP WITH TIME ZONE),
MEMBER FUNCTION get_end_date RETURN TIMESTAMP WITH TIME ZONE)
);
CREATE OR REPLACE TYPE XS$ACE LIST AS VARRAY(1000) OF XS$ACE TYPE;
```

# 11.2.4 Summary of XS\_ACL Subprograms

Table 11-3 Summary of XS ACL Subprograms

Subprogram	Description
CREATE_ACL Procedure	Creates an Access Control List (ACL).
APPEND_ACES Procedure	Adds one or more Access Control Entries (ACEs) to an existing ACL.
REMOVE_ACES Procedure	Removes all ACEs from an ACL.
SET_SECURITY_CLASS Procedure	Sets or modifies the security class for an ACL.
SET_PARENT_ACL Procedure	Sets or modifies the parent ACL for an ACL.
ADD_ACL_PARAMETER Procedure	Adds an ACL parameter value for a data security policy.
REMOVE_ACL_PARAMETERS Procedure	Removes ACL parameters and values for an ACL.
SET_DESCRIPTION Procedure	Sets a description string for an ACL.
DELETE_ACL Procedure	Deletes the specified ACL.

This section describes the following XS ACL subprograms:

# 11.2.4.1 CREATE ACL Procedure

The CREATE ACL procedure creates a new Access Control List (ACL).

## **Syntax**



### **Parameters**

Parameter	Description
name	The name of the ACL to be created.
	The name is schema qualified, for example, SCOTT.ACL1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as ACL1, and the current schema is SCOTT, it would resolve to SCOTT.ACL1.
ace_list	The list of Access Control Entries (ACEs) in the ACL.
sec_class	The name of the security class that specifies the scope or type of the ACL. If no security class is specified, then the DML class is used as the default security class.
	The name is schema qualified, for example, SCOTT.ACL1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as ACL1, and the current schema is SCOTT, it would resolve to SCOTT.ACL1.
parent	The parent ACL name, if any.  The name is schema qualified, for example, SCOTT.ACL1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as ACL1, and the current schema is SCOTT, it would resolve to SCOTT.ACL1.
inherit_mode	The inheritance mode if a parent ACL is specified. The allowed values are: EXTENDED or CONSTRAINED.
description	An optional description for the ACL.

### **Examples**

The following example creates an ACL called HRACL. This ACL includes ACEs contained in ace\_list. The privileges used in ace\_list are part of the HRPRIVS security class.

# 11.2.4.2 APPEND\_ACES Procedure

The APPEND ACES procedure adds one or more ACE to an existing ACL.

### **Syntax**

#### **Parameters**

Parameter	Description
acl	The name of the ACL to which the ACE is to be added.
	The name is schema qualified, for example, SCOTT.ACL1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as ACL1, and the current schema is SCOTT, it would resolve to SCOTT.ACL1.
ace	The ACE to be added to the ACL.
ace_list	The list of ACEs to be added to the ACL.

## **Examples**

The following example adds an ACE to the  $\mbox{HRACL}$  ACL. The ACE grants the  $\mbox{SELECT}$  privilege to the  $\mbox{DB}_{\mbox{HR}}$  database user.

# 11.2.4.3 REMOVE\_ACES Procedure

The REMOVE ACES procedure removes all ACEs from an ACL.

## **Syntax**

```
XS_ACL.REMOVE_ACES (
   acl IN VARCHAR2);
```

Parameter	Description
acl	The name of the ACL from which the ACEs are to be removed.
	The name is schema qualified, for example, SCOTT.ACL1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as ACL1, and the current schema is SCOTT, it would resolve to SCOTT.ACL1.



The following example removes all ACEs from the ACL called HRACL:

```
BEGIN
   SYS.XS_ACL.REMOVE_ACES('HRACL');
END;
```

# 11.2.4.4 SET SECURITY CLASS Procedure

The  ${\tt SET\_SECURITY\_CLASS}$  procedure sets or modifies the security class for an ACL.

## **Syntax**

#### **Parameters**

Parameter	Description
acl	The name of the ACL for which the security class is to be set.
	The name is schema qualified, for example, SCOTT.ACL1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as ACL1, and the current schema is SCOTT, it would resolve to SCOTT.ACL1.
sec_class	The name of the security class that defines the ACL scope or type. The name is schema qualified, for example, SCOTT.ACL1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as ACL1, and the current schema is SCOTT, it would resolve to SCOTT.ACL1.

### **Examples**

The following example associates the HRPRIVS security class with the HRACL ACL:

```
BEGIN
   SYS.XS_ACL.SET_SECURITY_CLASS('HRACL','HRPRIVS');
END;
```

# 11.2.4.5 SET\_PARENT\_ACL Procedure

The SET PARENT ACL sets or modifies the parent ACL for an ACL.

### **Syntax**



### **Parameters**

Parameter	Description
acl	The name of the ACL whose parent needs to be set.
	The name is schema qualified, for example, SCOTT.ACL1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as ACL1, and the current schema is SCOTT, it would resolve to SCOTT.ACL1.
parent	The name of the parent ACL.  The name is schema qualified, for example, SCOTT.ACL1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as ACL1, and the current schema is SCOTT, it would resolve to SCOTT.ACL1.
inherit_mode	The inheritance mode. This can be one of the following values: EXTENDED (extends from), CONSTRAINED (constrained with)

## **Examples**

The following example sets the AllDepach ACL as the parent ACL for the HRACL ACL. The inheritance type is set to EXTENDED.

```
BEGIN
SYS.XS_ACL.SET_PARENT_ACL('HRACL','AllDepACL',XS_ACL.EXTENDED);
END:
```

# 11.2.4.6 ADD\_ACL\_PARAMETER Procedure

The ADD ACL PARAMETER adds an ACL parameter value for a data security policy.

## **Syntax**

```
XS_ACL.ADD_ACL_PARAMETER (
acl IN VARCHAR2,
policy IN VARCHAR2,
parameter IN VARCHAR2,
value IN NUMBER);

XS_ACL.ADD_ACL_PARAMETER (
acl IN VARCHAR2,
policy IN VARCHAR2,
parameter IN VARCHAR2,
value IN VARCHAR2);
```

Parameter	Description
acl	The name of the ACL to which the parameter is to be added.
	The name is schema qualified, for example, SCOTT.ACL1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as ACL1, and the current schema is SCOTT, it would resolve to SCOTT.ACL1.



Parameter	Description
policy	The name of the data security policy for which the ACL parameter has been created.
	The name is schema qualified, for example, SCOTT.ACL1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as ACL1, and the current schema is SCOTT, it would resolve to SCOTT.ACL1.
parameter	The name of the ACL parameter as defined by the data security policy.
value	The value of the ACL parameter to be used.

The following example adds the REGION parameter for ACL1. The name of the data security policy for which the ACL parameter is created is  ${\tt TEST\_DS}$ . The value of the REGION parameter is WEST.

```
BEGIN
   SYS.XS_ACL.ADD_ACL_PARAMETER('ACL1','TEST_DS','REGION', 'WEST');
END;
```

# 11.2.4.7 REMOVE ACL PARAMETERS Procedure

The REMOVE\_ACL\_PARAMETERS removes the specified ACL parameter for an ACL. If no parameter name is specified, then all ACL parameters for the ACL are removed.

### **Syntax**

#### **Parameters**

Parameter	Description
acl	The name of the ACL from which the parameter(s) are to be removed.
	The name is schema qualified, for example, SCOTT.ACL1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as ACL1, and the current schema is SCOTT, it would resolve to SCOTT.ACL1.
parameter	The name of the parameter that needs to be removed from the ACL.

### **Examples**

The following example removes the REGION parameter from the ACL1 ACL:

```
BEGIN
   XS_ACL.REMOVE_ACL_PARAMETERS('ACL1', 'REGION');
END;
```

The following example removes all ACL parameters for ACL1.



```
BEGIN
   SYS.XS_ACL.REMOVE_ACL_PARAMETERS('ACL1');
END:
```

# 11.2.4.8 SET\_DESCRIPTION Procedure

The SET DESCRIPTION procedure sets a description string for an ACL.

### **Syntax**

#### **Parameters**

Parameter	Description
acl	The name of the ACL for which the description is to be set.
	The name is schema qualified, for example, SCOTT.ACL1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as ACL1, and the current schema is SCOTT, it would resolve to SCOTT.ACL1.
description	A string description for the ACL.

### **Examples**

The following example sets a description for the HRACL ACL:

# 11.2.4.9 DELETE\_ACL Procedure

The DELETE ACL procedure deletes the specified ACL.

## **Syntax**

Parameter	Description
acl	The name of the ACL to be deleted.
	The name is schema qualified, for example, SCOTT.ACL1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as ACL1, and the current schema is SCOTT, it would resolve to SCOTT.ACL1.



Parameter	Description
delete_option	The delete option to use. To the data security policy, the behavior of the following options is the same:
	• DEFAULT_OPTION:
	The default option allows deleting an ACL only if it is not referenced elsewhere. If the ACL is referenced elsewhere, then the ACL cannot be deleted.
	For example, the delete operation fails if you try to delete an ACL that is part of a data security policy.  • CASCADE_OPTION:
	The cascade option deletes the ACL and also removes the ACL reference in a data realm constraint of a data security policy.  ALLOW_INCONSISTENCIES_OPTION:
	The allow inconsistencies option lets you delete the ACL even if other entities have late binding references to it. In this mode, the ACL will be removed but the references are not removed.

The following example deletes the HRACL ACL using the default delete option:

```
BEGIN
   SYS.XS_ACL.DELETE_ACL('HRACL');
END;
```

# 11.3 XS\_ADMIN\_UTIL Package

The XS\_ADMIN\_UTIL package contains helper subprograms to be used by other packages.

This section includes the following topics:

- Security Model
- Constants
- Object Types, Constructor Functions, Synonyms, and Grants
- Summary of XS\_ADMIN\_UTIL Subprograms

# 11.3.1 Security Model

The XS\_ADMIN\_UTIL package is created in the SYS schema. The caller has invoker's rights on this package. The SYS privilege is required to grant or revoke a Real Application Security system privilege to or from a user or role.

# 11.3.2 Constants

The following constants define the delete options:

```
DEFAULT_OPTION CONSTANT PLS_INTEGER := 1;
CASCADE_OPTION CONSTANT PLS_INTEGER := 2;
ALLOW_INCONSISTENCIES_OPTION CONSTANT PLS_INTEGER := 3;
```

The following constants define the principal's type:

```
PTYPE_XS CONSTANT PLS_INTEGER := 1;

PTYPE_DB CONSTANT PLS_INTEGER := 2;

PTYPE_DN CONSTANT PLS_INTEGER := 3;

PTYPE_EXTERNAL CONSTANT PLS_INTEGER := 4;
```

# 11.3.3 Object Types, Constructor Functions, Synonyms, and Grants

The following object types, constructor functions, synonyms, and GRANT statements are defined for this package.

```
CREATE OR REPLACE TYPE XS$LIST IS VARRAY(1000) OF VARCHAR2(4000);
CREATE OR REPLACE TYPE XS$NAME LIST IS VARRAY(1000) OF VARCHAR2(261);
```

# 11.3.4 Summary of XS\_ADMIN\_UTIL Subprograms

Table 11-4 Summary of XS\_ADMIN\_UTIL Subprograms

Subprogram	Brief Description
GRANT_SYSTEM_PRIVILEGE Procedure	Grant a Real Application Security system privilege to a user or role.
REVOKE_SYSTEM_PRIVILEGE Procedure	Revoke a Real Application Security system privilege from a user or role.

This section describes the following XS ADMIN UTIL subprograms:

# 11.3.4.1 GRANT SYSTEM PRIVILEGE Procedure

The GRANT\_SYSTEM\_PRIVILEGE procedure is used to grant a Real Application Security system privilege or schema privilege to a user or role. Only SYS or a user who has GRANT ANY PRIVILEGE privilege can perform this operation.

The audit action AUDIT\_GRANT\_PRIVILEGE, audits all GRANT\_SYSTEM\_PRIVILEGE calls for granting system privileges or schema privileges.

### **Syntax**

Parameter	Description
priv_name	Specifies the name of the Real Application Security system privilege or schema privilege to be granted.
user_name	Specifies the name of the user or role to which the Real Application Security system privilege or schema privilege is to be granted.
user_type	The type of user. By default the database user.

Parameter	Description
schema	The schema on which the privilege is granted. The value is NULL if the privilege is a system privilege.

The following example creates a database user, <code>dbuser1</code>, and grants Real Application Security privilege <code>ADMINISTER\_SESSION</code> to this database user and specifies the <code>user\_type</code> as <code>XS\_ADMIN\_UTIL.PTYPE\_DB</code>, though by default, this is the default value and need not be specified.

```
SQL> CREATE USER dbuser1 identified by password;

SQL> EXEC SYS.XS_ADMIN_UTIL.GRANT_SYSTEM_PRIVILEGE('ADMINISTER_SESSION',
'dbuser1', XS ADMIN UTIL.PTYPE DB, 'HR1');
```

The following example creates an application user, user1, and grants Real Application Security privilege ADMINISTER\_SESSION to this application user, specifies the user\_type as XS\_ADMIN\_UTIL.PTYPE\_XS, and specifies the schema as HR1.

```
SQL> EXEC SYS.XS_PRINCIPAL.CREATE_USER('user1','HR1');

SQL> EXEC SYS.XS_PRINCIPAL.SET_PASSWORD('user1', 'password');

SQL> EXEC SYS.XS_ADMIN_UTIL.GRANT_SYSTEM_PRIVILEGE('ADMINISTER_SESSION', 'user1', XS ADMIN_UTIL.PTYPE_XS, 'HR1');
```

# 11.3.4.2 REVOKE\_SYSTEM\_PRIVILEGE Procedure

The REVOKE\_SYSTEM\_PRIVILEGE is used to revoke a Real Application Security ststem privilege or schema privilege from a user or role. Only SYS privilege or a user with GRANT ANY PRIVILEGE privilege can perform this operation.

The audit action AUDIT\_REVOKE\_PRIVILEGE, audits all REVOKE\_SYSTEM\_PRIVILEGE calls for revoking system privileges or schema privileges.

### **Syntax**

Parameter	Description
priv_name	Specifies the name of the Real Application Security system privilege or schema privilege to be revoked.
user_name	Specifies the name of the user or role from which the Real Application Security system privilege or schema privilege is to be revoked.
user_type	The type of user. By default the database user.



Parameter	Description
schema	The schema on which the privilege is revoked. The value is ${\tt NULL}$ if the privilege is a system privilege.

The following example creates a database user, <code>dbuser1</code>, and revokes Real Application Security privilege <code>ADMINISTER\_SESSION</code> from this database user and specifies the <code>user\_type</code> as <code>XS\_ADMIN\_UTIL.PTYPE\_DB</code>, though by default, this is the default value and need not be specified.

```
CREATE USER dbuser1 identified by password;

SYS.XS_ADMIN_UTIL.REVOKE_SYSTEM_PRIVILEGE('ADMINISTER_SESSION','dbuser1',
XS ADMIN UTIL.PTYPE DB, 'HR1');
```

The following example creates an application user, user1, and revokes Real Application Security privilege ADMINISTER\_SESSION from this application user and specifies the user\_type as XS ADMIN UTIL.PTYPE XS.

```
SQL> EXEC SYS.XS_PRINCIPAL.CREATE_USER('user1', 'HR1');

SQL> EXEC SYS.XS_PRINCIPAL.SET_PASSWORD('user1', 'password');

SQL> EXEC SYS.XS_ADMIN_UTIL.REVOKE_SYSTEM_PRIVILEGE('ADMINISTER_SESSION', 'user1',
XS ADMIN UTIL.PTYPE XS, 'HR1');
```

# 11.4 XS\_DATA\_SECURITY Package

The XS\_DATA\_SECURITY package includes procedures to create, manage, and delete data security policies, associated data realm constraints, column constraints, and ACL parameters.

This section includes the following topics:

- Security Model
- Object Types, Constructor Functions, Synonyms, and Grants
- Summary of XS\_DATA\_SECURITY Subprograms

# 11.4.1 Security Model for the XS\_DATA\_SECURITY Package

The XS\_DATA\_SECURITY package is created under the SYS schema. The DBA role is granted the ADMIN\_ANY\_SEC\_POLICY, which allows it to administer schema objects like ACLs, security classes, and security policies across all schemas. In addition, users granted the ADMIN\_ANY\_SEC\_POLICY can call the following procedures: ENABLE\_OBJECT\_POLICY, DISABLE\_OBJECT\_POLICY, APPLY\_OBJECT\_POLICY, and REMOVE\_OBJECT\_POLICY.

Users can administer schema objects in their own schema if they have been granted the RESOURCE role for the schema. The RESOURCE role and the XS\_RESOURCE application role include the ADMIN\_SEC\_POLICY privilege, required to administer schema objects in the schema as well as administering the policy artifacts within the granted schema to achieve policy management within an application.

Users can administer policy enforcement on the schema if they have been granted the  ${\tt APPLY\_SEC\_POLICY}$  privilege. With this privilege, the user can administer policy enforcement within granted schemas to achieve policy management within an application.

# 11.4.2 Object Types, Constructor Functions, Synonyms, and Grants

The following object types, constructor functions, synonyms, and GRANT statements are defined for this package.

```
-- Create a type for key
CREATE OR REPLACE TYPE XS$KEY TYPE AS OBJECT (
primary_key VARCHAR2(130),
foreign_key VARCHAR2(4000),
-- Foreign key type; 1 = col name, 2 = col value
foreign key type NUMBER,
-- Constructor function
CONSTRUCTOR FUNCTION XS$KEY TYPE
                    foreign_key_type IN NUMBER)
                    RETURN SELF AS RESULT,
MEMBER FUNCTION GET PRIMARY KEY RETURN VARCHAR2,
MEMBER FUNCTION GET FOREIGN KEY RETURN VARCHAR2,
MEMBER FUNCTION GET FOREIGN KEY TYPE RETURN NUMBER,
CREATE OR REPLACE TYPE XS$KEY LIST AS VARRAY(1000) OF XS$KEY TYPE;
CREATE OR REPLACE TYPE XS$REALM CONSTRAINT TYPE AS OBJECT (
-- Member variables
realm type NUMBER,
-- Member evaluation rule
              VARCHAR2 (4000),
-- acl list of instance set
acl list XS$NAME LIST,
-- isStatic variable for instance set. Stored as INTEGER. No boolean datatype
-- for objects. False is stored as 0 and TRUE is stored as 1
is static INTEGER,
-- Indicate if the realm is parameterized.
parameterized INTEGER,
-- parent schema name for inherited from
parent schema VARCHAR2(130),
-- parent object name for inherited from
parent object VARCHAR2(130),
-- keys for inherited from
key list XS$KEY LIST,
-- when condition for inherited from
when condition VARCHAR2 (4000),
-- Constructor function - row level realm
CONSTRUCTOR FUNCTION XS$REALM CONSTRAINT TYPE
                    (realm IN VARCHAR2,
                    acl list
                                        IN XS$NAME LIST,
                    acl_list IN XSQNAME_LIST, is static IN BOOLEAN := FALSE)
                    RETURN SELF AS RESULT,
-- Constructor function - parameterized row level realm
CONSTRUCTOR FUNCTION XS$REALM CONSTRAINT TYPE
```

#### RETURN SELF AS RESULT,

```
-- Constructor function - master realm
CONSTRUCTOR FUNCTION XS$REALM CONSTRAINT TYPE
                     (parent_schema IN VARCHAR2,
                      parent_object IN VARCHAR2,
                      key list IN XS$KEY LIST,
                      when condition IN VARCHAR2:= NULL)
                      RETURN SELF AS RESULT,
-- Accessor functions
MEMBER FUNCTION GET TYPE RETURN NUMBER,
MEMBER FUNCTION GET REALM RETURN VARCHAR2,
MEMBER FUNCTION GET ACLS RETURN XS$NAME LIST,
MEMBER FUNCTION IS DYNAMIC REALM RETURN BOOLEAN,
MEMBER FUNCTION IS STATIC REALM RETURN BOOLEAN,
MEMBER FUNCTION IS PARAMETERIZED REALM RETURN BOOLEAN,
MEMBER FUNCTION GET KEYS RETURN XS$KEY LIST,
MEMBER FUNCTION GET PARENT SCHEMA RETURN VARCHAR2,
MEMBER FUNCTION GET PARENT OBJECT RETURN VARCHAR2,
MEMBER FUNCTION GET WHEN CONDITION RETURN VARCHAR2,
MEMBER PROCEDURE SET REALM(realm IN VARCHAR2),
MEMBER PROCEDURE ADD ACLS(acl IN VARCHAR2),
MEMBER PROCEDURE ADD ACLS (acl list IN XS$NAME LIST),
MEMBER PROCEDURE SET ACLS (acl list IN XS$NAME LIST),
MEMBER PROCEDURE SET DYNAMIC,
MEMBER PROCEDURE SET STATIC,
MEMBER PROCEDURE ADD KEYS(key IN XS$KEY TYPE),
MEMBER PROCEDURE ADD_KEYS(key_list IN XS$KEY_LIST),
MEMBER PROCEDURE SET KEYS(key list IN XS$KEY LIST),
MEMBER PROCEDURE SET PARENT SCHEMA(parent schema IN VARCHAR2),
MEMBER PROCEDURE SET PARENT OBJECT (parent object IN VARCHAR2),
MEMBER PROCEDURE SET WHEN CONDITION (when condition IN VARCHAR2)
-- Create a list of realm constraint type
CREATE OR REPLACE TYPE XS$REALM CONSTRAINT LIST AS VARRAY(1000)
                       OF XS$REALM CONSTRAINT TYPE;
-- Create a type for column(attribute) security
CREATE OR REPLACE TYPE XS$COLUMN CONSTRAINT TYPE AS OBJECT (
-- column list
column list
                 XS$LIST,
-- privilege for column security
privilege
                  VARCHAR2 (261),
-- Constructor function
CONSTRUCTOR FUNCTION XS$COLUMN CONSTRAINT TYPE
                    (column list IN XS$LIST,
                     privilege IN VARCHAR2)
                     return SELF AS RESULT,
MEMBER FUNCTION GET COLUMNS RETURN XS$LIST,
MEMBER FUNCTION GET PRIVILEGE RETURN VARCHAR2,
MEMBER PROCEDURE ADD COLUMNS (column IN VARCHAR2),
MEMBER PROCEDURE ADD_COLUMNS(column_list IN XS$LIST),
MEMBER PROCEDURE SET COLUMNS (column list IN XS$LIST),
MEMBER PROCEDURE SET PRIVILEGE (privilege IN VARCHAR2)
-- Create a list of column constraint for column security
CREATE OR REPLACE TYPE XS$COLUMN CONSTRAINT LIST
                       IS VARRAY(1000) of XS$COLUMN CONSTRAINT TYPE;
```

# 11.4.3 Summary of XS\_DATA\_SECURITY Subprograms

Table 11-5 Summary of XS\_DATA\_SECURITY Subprograms

Subprogram	Brief Description
CREATE_POLICY Procedure	Creates a new data security policy.
APPEND_REALM_CONSTRAINTS Procedure	Adds one or more data realm constraints to an existing data security policy.
REMOVE_REALM_CONSTRAINTS Procedure	Removes all data realm constraints for the specified data security policy.
ADD_COLUMN_CONSTRAINTS Procedure	Adds one or more column constraint to the specified data security policy.
REMOVE_COLUMN_CONSTRAINTS Procedure	Removes all column constraints from a data security policy.
CREATE_ACL_PARAMETER Procedure	Creates an ACL parameter for the specified data security policy.
DELETE_ACL_PARAMETER Procedure	Deletes an ACL parameter from the specified data security policy.
SET_DESCRIPTION Procedure	Sets a description string for the specified data security policy.
DELETE_POLICY Procedure	Deletes a data security policy.

Table 11-6 Summary of XS\_DATA\_SECURITY Subprograms for Managing Data Security Policies on Tables or Views

Subprogram	Brief Description
ENABLE_OBJECT_POLICY Procedure	Enables the data security policy for the specified table or view.
DISABLE_OBJECT_POLICY Procedure	Disables the data security policy for the specified table or view.
REMOVE_OBJECT_POLICY Procedure	Removes or drops the data security from the specified table or view without deleting it.
APPLY_OBJECT_POLICY Procedure	Enables or reenables the data security policy for the specified table or view.

This section describes the following XS\_DATA\_SECURITY subprograms:

# 11.4.3.1 CREATE\_POLICY Procedure

The CREATE POLICY procedure creates a new data security policy.

#### **Syntax**



#### **Parameters**

Parameter	Description
name	The name for the data security policy to be created.
	The name is schema qualified, for example, SCOTT.POLICY1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as POLICY1, and the current schema is SCOTT, it would resolve to SCOTT.POLICY1.
realm_constraint_list	The list of data realm constraints, which determine the rows to be protected by the data security policy.
column_constraint_list	This is optional. The list of attributes and the privileges protecting them.
description	An optional description for the data security policy.

#### **Examples**

The following example creates a data security policy called <code>USER1.EMPLOYEES\_DS</code>. It uses a data realm constraint to protect data related to department numbers 60 and 100. In addition, access to the <code>SALARY</code> column (attribute) is restricted using an column constraint.

```
realm cons XS$REALM CONSTRAINT LIST;
  column_cons XS$COLUMN_CONSTRAINT_LIST;
BEGIN
  realm cons :=
    XS$REALM CONSTRAINT LIST(
      XS$REALM CONSTRAINT TYPE(realm=> 'DEPARTMENT ID in (60, 100)',
                               acl list=> XS$NAME LIST('HRACL')));
  column cons :=
    XS$COLUMN CONSTRAINT LIST(
     XS$COLUMN_CONSTRAINT_TYPE(column_list=> XS$LIST('SALARY'),
                            privilege=> 'VIEW SENSITIVE INFO'));
  SYS.XS DATA SECURITY.CREATE POLICY(
          name=>'USER1.EMPLOYEES DS',
          realm constraint list=>realm cons,
          column constraint list=>column cons);
END;
```

# 11.4.3.2 APPEND\_REALM\_CONSTRAINTS Procedure

The APPEND\_REALM\_CONSTRAINTS procedure adds one or more data realm constraints to an existing data security policy.

### **Syntax**



### **Parameters**

Parameter	Description
policy	The name of the data security policy to which the data realm constraints are to be added.
	The name is schema qualified, for example, SCOTT.POLICY1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as POLICY1, and the current schema is SCOTT, it would resolve to SCOTT.POLICY1.
realm_constraint	The data realm constraint to be added to the data security policy.
realm_constraint_list	The list of data realm constraints to be added to the data security policy.

## **Examples**

The following example appends a new data realm constraint to the  ${\tt EMPLOYEES\_DS}$  data security policy.

# 11.4.3.3 REMOVE REALM CONSTRAINTS Procedure

The REMOVE\_REALM\_CONSTRAINTS procedure removes all data realm constraints from a data security policy.

### **Syntax**

```
XS_DATA_SECURITY.REMOVE_REALM_CONSTRAINTS (
  policy IN VARCHAR2);
```

Parameter	Description	
policy	The name of the data security policy from which the data realm constraints at to be removed.	
	The name is schema qualified, for example, SCOTT.POLICY1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as POLICY1, and the current schema is SCOTT, it would resolve to SCOTT.POLICY1.	



The following example removes all data realm constraints from the EMPLOYEES\_DS data security policy.

```
BEGIN
   SYS.XS_DATA_SECURITY.REMOVE_REALM_CONSTRAINTS('EMPLOYEES_DS');
END:
```

# 11.4.3.4 ADD\_COLUMN\_CONSTRAINTS Procedure

The ADD\_COLUMN\_CONSTRAINTS procedure adds one or more column constraint to a data security policy.

### **Syntax**

#### **Parameters**

Parameter	Description
policy	The name of the data security policy to which the attribute constraints are to be added.
	The name is schema qualified, for example, SCOTT.POLICY1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as POLICY1, and the current schema is SCOTT, it would resolve to SCOTT.POLICY1.
column_constraint	The column constraint to be added.
column_constraint_list	The list of column constraints to be added.

### **Examples**

The following example adds a column constraint on the  $COMMISSION\_PCT$  column in the EMPLOYEES DS data security policy:



# 11.4.3.5 REMOVE COLUMN CONSTRAINTS Procedure

The REMOVE\_COLUMN\_CONSTRAINTS procedure removes all column constraints from a data security policy.

### **Syntax**

```
XS_DATA_SECURITY.REMOVE_COLUMN_CONSTRAINTS (
   policy IN VARCHAR2,);
```

### **Parameters**

Parameter	Description	
policy	The name of the data security policy for which the column constraints are to removed.	
	The name is schema qualified, for example, SCOTT.POLICY1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as POLICY1, and the current schema is SCOTT, it would resolve to SCOTT.POLICY1.	

### **Examples**

The following example removes all column constraints from the  ${\tt EMPLOYEES\_DS}$  data security policy:

```
BEGIN
   SYS.XS_DATA_SECURITY.REMOVE_COLUMN_CONSTRAINTS('EMPLOYEES_DS');
END:
```

# 11.4.3.6 CREATE ACL PARAMETER Procedure

The CREATE\_ACL\_PARAMETER procedure creates an ACL parameter for a data security policy.

## **Syntax**

Parameter	Description
policy	The name of the data security policy for which the ACL parameter needs to be created.
	The name is schema qualified, for example, SCOTT.POLICY1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as POLICY1, and the current schema is SCOTT, it would resolve to SCOTT.POLICY1.
parameter	The name of the ACL parameter to be created.



Parameter	Description
param_type	The data type of the parameter. This can be 1 (NUMBER) or 2 (VARCHAR).

The following examples creates an ACL parameter, called <code>DEPT\_POLICY</code>, for the <code>EMPLOYEES\_DS</code> data security policy:

```
BEGIN
   SYS.XS_DATA_SECURITY.CREATE_ACL_PARAMETER('EMPLOYEES_DS','DEPT_POLICY',1);
END;
```

# 11.4.3.7 DELETE ACL PARAMETER Procedure

The DELETE ACL PARAMETER procedure deletes an ACL parameter for a data security policy.

## **Syntax**

### **Parameters**

Parameter	Description
policy	The name of the data security policy for which the ACL parameter is to be deleted.
	The name is schema qualified, for example, SCOTT.POLICY1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as POLICY1, and the current schema is SCOTT, it would resolve to SCOTT.POLICY1.
parameter	The name of the ACL parameter to be deleted.
delete_option	The delete option to use. The following options are available:  • DEFAULT_OPTION (default):  The default option allows deleting an ACL parameter only if it is not referenced elsewhere. If there are other entities that reference the ACL
	parameter, then the ACL parameter cannot be deleted.  • CASCADE OPTION:
	The cascade option deletes the ACL parameter together with any references to it. The user deleting the security class must have privileges to delete these references as well.
	• ALLOW_INCONSISTENCIES_OPTION:
	The allow inconsistencies option lets you delete the entity even if other entities have late binding references to it. If the entity is part of an early dependency, then the delete fails and an error is raised.

### **Examples**

The following example deletes the <code>DEPT\_POLICY</code> ACL parameter from the <code>EMPLOYEES\_DS</code> data security policy, using the default option.

```
BEGIN
SYS.XS_DATA_SECURITY.DELETE_ACL_PARAMETER('EMPLOYEES_DS','DEPT_POLICY',
XS_ADMIN_UTIL.DEFAULT_OPTION);
END;
```

# 11.4.3.8 SET\_DESCRIPTION Procedure

The  ${\tt SET\_DESCRPTION}$  procedure sets a description string for the specified data security policy.

### **Syntax**

#### **Parameters**

Parameter	Description
policy	The name of the data security policy for which the description is to be set.
	The name is schema qualified, for example, SCOTT.POLICY1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as POLICY1, and the current schema is SCOTT, it would resolve to SCOTT.POLICY1.
description	A description string for the specified data security policy.

### **Examples**

The following example sets a description string for the <code>EMPLOYEES\_DS</code> data security policy:

# 11.4.3.9 DELETE\_POLICY Procedure

The DELETE\_POLICY procedure deletes a data security policy.

#### **Syntax**



### **Parameters**

Description
The name of the data security policy to be deleted.  The name is schema qualified, for example, SCOTT.POLICY1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as POLICY1, and the current schema is SCOTT, it would resolve to SCOTT.POLICY1.
<ul> <li>The delete option to use. To the security policy, the behavior of the following options is the same:</li> <li>DEFAULT_OPTION:  The default option allows deleting a data security policy only if it is not referenced elsewhere. If there are other entities that reference the data security policy, then the data security policy cannot be deleted.</li> <li>CASCADE_OPTION:  The cascade option deletes the data security policy together with any references to it. The user deleting the data security policy deletes these references as well.</li> <li>ALLOW_INCONSISTENCIES_OPTION:  The allow inconsistencies option lets you delete the entity even if other entities have late binding references to it. If the entity is part of an early</li> </ul>
_

## **Examples**

The following example deletes the EMPLOYEES DS data security policy using the default option.

# 11.4.3.10 ENABLE OBJECT POLICY Procedure

The <code>ENABLE\_OBJECT\_POLICY</code> procedure enables the data security policy for the specified table or view. <code>ENABLE\_OBJECT\_POLICY</code> enables the ACL-based row level security policy for the table or view.

You may want to enable data security policies after you perform an import or export on the tables that it affects, or for debugging purposes.

To find the status of the data security policies for tables or views available for the current user, query the DBA\_XS\_APPLIED\_POLICIES data dictionary view.

Before enforcing policies, a check is made for the APPLY SEC POLICY privilege.

## **Syntax**

```
XS_DATA_SECURITY.ENABLE_OBJECT_POLICY (
  policy IN VARCHAR2,
  schema IN VARCHAR2,
  object IN VARCHAR2);
```



#### **Parameters**

Parameter	Description
policy	The name of the data security policy to be enabled.
	The name is schema qualified, for example, SCOTT.POLICY1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as POLICY1, and the current schema is SCOTT, it would resolve to SCOTT.POLICY1.
schema	The name of the schema that contains the table or view to enable.
object	The name of the table or view to enable the data security policy.

### **Examples**

The following example enables XDS for the products table in the sales schema.

```
BEGIN
   SYS.XS_DATA_SECURITY.ENABLE_OBJECT_POLICY(policy =>'CUST_DS', schema=>'sales',
object=>'products');
END;
```

# 11.4.3.11 DISABLE\_OBJECT\_POLICY Procedure

The <code>DISABLE\_OBJECT\_POLICY</code> procedure disables the data security policy for the specified table or view. <code>DISABLE\_OBJECT\_POLICY</code> disables the ACL-based row level security policy for the table or view.

You may want to disable Real Application Security if you are performing an import or export on the tables that it affects, or for debugging purposes.

To find the status of the data security policies for tables or views available for the current user, query the DBA\_XS\_APPLIED\_POLICIES data dictionary view.

Before enforcing policies, a check is made for the APPLY SEC POLICY privilege.

### **Syntax**

```
XS_DATA_SECURITY.DISABLE_OBJECT_POLICY (
policy IN VARCHAR2,
schema IN VARCHAR2,
object IN VARCHAR2);
```

Parameter	Description
policy	The name of the data security policy to be disabled.
	The name is schema qualified, for example, SCOTT.POLICY1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as POLICY1, and the current schema is SCOTT, it would resolve to SCOTT.POLICY1.
schema	The name of the schema that contains the table or view to disable.



Parameter	Description
object	The name of the table or view to disable a data security policy.

The following example disables XDS for the products table in the sales schema.

```
BEGIN
    SYS.XS_DATA_SECURITY.DISABLE_OBJECT_POLICY(policy =>'CUST_DS', schema=>'sales',
object=>'products');
END;
```

# 11.4.3.12 REMOVE OBJECT POLICY Procedure

The REMOVE\_OBJECT\_POLICY procedure drops the data security policy from the specified table or view without deleting it. REMOVE\_OBJECT\_POLICY drops the ACL Materialized View built by ENABLE XDS on a static data realm constraint.

To find the status of the data security policies for tables or views available for the current user, query the DBA\_XS\_APPLIED\_POLICIES data dictionary view.

Before enforcing policies, a check is made for the APPLY SEC POLICY privilege.

### **Syntax**

```
XS_DATA_SECURITY.REMOVE_OBJECT_POLICY (
policy IN VARCHAR2,
schema IN VARCHAR2,
object IN VARCHAR2);
```

## **Parameters**

Parameter	Description
policy	The name of the data security policy to be dropped.
	The name is schema qualified, for example, SCOTT.POLICY1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as POLICY1, and the current schema is SCOTT, it would resolve to SCOTT.POLICY1.
schema	The name of the schema that contains the table or view from which to remove the data security policy.
object	The name of the table or view from which to remove the data security. policy

#### **Examples**

The following example drops the  $\texttt{CUST\_DS}$  data security policy from the products table in the sales schema.

```
BEGIN
   SYS.XS_DATA_SECURITY.REMOVE_OBJECT_POLICY(policy=>'CUST_DS', schema=>'sales',
object=>'products');
END;
```



# 11.4.3.13 APPLY OBJECT POLICY Procedure

The APPLY\_OBJECT\_POLICY procedure enables or reenables the data security policy for the specified database table or view.

To find the status of the data security policies for tables or views available for the current user, query the DBA\_XS\_APPLIED\_POLICIES data dictionary view.

Before enforcing policies, a check is made for the  ${\tt APPLY\_SEC\_POLICY}$  privilege.

## **Syntax**

```
XS_DATA_SECURITY.APPLY_OBJECT_POLICY (
policy IN VARCHAR2,
schema IN VARCHAR2,
object IN VARCHAR2,
row_acl IN BOOLEAN DEFAULT FALSE,
owner_bypass IN BOOLEAN DEFAULT FALSE,
statement_types IN VARCHAR2 DEFAULT NULL,
aclmv IN VARCHAR2 DEFAULT NULL);
```

Parameter	Description
policy	Name of the data security policy to be enabled.
	The name is schema qualified, for example, SCOTT.POLICY1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as POLICY1, and the current schema is SCOTT, it would resolve to SCOTT.POLICY1.
schema	The name of the schema that contains the relational table or view to enable or re-enable.
object	The name of the relational table or view to enable or re-enable for the data security policy.
row_acl	The default is FALSE. When set to TRUE, creates the hidden column SYS_ACLOD.
owner_bypass	The owner of the object can bypass the data security policy. The default is ${\tt FALSE}.$
statement_types	The types can be: SELECT, INSERT, UPDATE, DELETE, and INDEX.
	Note that if your application security requires that you must update table rows and also restrict read access to certain columns in the same table, you must use two APPLY_OBJECT_POLICY procedures to enforce each data security policy to ensure precise enforcement of each policy. For example, one APPLY_OBJECT_POLICY procedure would enforce the DML statement_types required for updating table rows (for example, INSERT, UPDATE, DELETE), while the other APPLY_OBJECT_POLICY procedure would enforce only the statement_types of SELECT for the column constraint.
aclmv	Specifies a user-provided MV name that maintains static ACL information. The MV has two columns: TABLEROWID and ACLIDLIST. The default value for aclmv is NULL.



The following example enables the DEPT\_POLICY data security policy for the EMP table in the HR schema.

# 11.5 XS\_DATA\_SECURITY\_UTIL Package

The XS\_DATA\_SECURITY\_UTIL package is a utility package that schedules automatic refreshment for static ACL to a user table and changes the ACL refreshment mode to oncommit or on-demand refresh.

This section includes the following topics:

- Security Model
- Constants
- Summary of XS\_DATA\_SECURITY\_UTIL Subprograms

# 11.5.1 Security Model

The XS\_DATA\_SECURITY\_UTIL package is created in the SYS schema. You need EXECUTE privileges on the package to be able to run the programs contained in this package.

# 11.5.2 Constants

The following are valid values for ACLMV refresh modes:

```
ACLMV_ON_DEMAND CONSTANT VARCHAR2(9) := 'ON DEMAND';
ACLMV ON COMMIT CONSTANT VARCHAR2(9) := 'ON COMMIT';
```

The following are types of refresh on static ACLMV:

```
XS_ON_COMMIT_MV CONSTANT BINARY_INTEGER := 0;
XS_ON_DEMAND_MV CONSTANT BINARY_INTEGER := 1;
XS SCHEDULED MV CONSTANT BINARY INTEGER := 2;
```

The following are types of static ACLMV:

```
XS_SYSTEM_GENERATED_MV CONSTANT BINARY_INTEGER := 0;
XS_USER_SPECIFIED_MV CONSTANT_BINARY_INTEGER := 1;
```



# 11.5.3 Summary of XS\_DATA\_SECURITY\_UTIL Subprograms

Table 11-7 Summary of XS\_DATA\_SECURITY\_UTIL Subprograms

Subprogram	Brief Description
SCHEDULE_STATIC_ACL_REFRE SH Procedure	Schedules automatic refreshment for static ACL to a user table
ALTER_STATIC_ACL_REFRESH Procedure	Changes the ACL refreshment mode to on-commit or on-demand refresh.

This section describes the following XS\_DATA\_SECURITY\_UTIL subprograms:

# 11.5.3.1 SCHEDULE STATIC ACL REFRESH Procedure

The SCHEDULE\_STATIC\_ACL\_REFRESH procedure is used to invoke or schedule automatic refresh for static ACL to a user table. It can start the refresh immediately if NULL value is passed into the start date and repeat interval parameters.

To find the status of all latest static ACL refresh jobs done for tables or views available for the current user, query the ALL\_XDS\_LATEST\_ACL\_REFSTAT, DBA\_XDS\_LATEST\_ACL\_REFSTAT, and USER\_XDS\_LATEST\_ACL\_REFSTAT data dictionary views. All static ACL refresh job status history can be found in ALL\_XDS\_ACL\_REFSTAT, DBA\_XDS\_ACL\_REFSTAT, and USER\_XDS\_ACL\_REFSTAT data dictionary views.

### **Syntax**

Parameter	Description
schema_name	Specifies the name for the schema to which the table belongs.
table_name	The table name which is used with above schema name to uniquely identify a table for the static ACL refreshment.
start_date	This attribute specifies the first date on which this refresh is scheduled to run. If the function is called repeatedly, then the latest given start_date and repeat_interval is used to schedule the job. Each execution result of ACL refresh done by immediate call, on-commit, or refresh job is added into XDS_ACL_REFSTAT.
	If start_date and repeat_interval are left NULL, then the refresh is launched immediately and any existing refresh schedule is erased. For immediate refresh, no row will be added into XDS_ACL_REFRESH, as it does not change refresh mode.



Parameter	Description
repeat_interval	This attribute specifies how often the refresh should repeat. You can specify the repeat interval by using <code>DBMS_SCHEDULER</code> package calendaring syntax or using PL/SQL expressions. See <i>Oracle Database PL/SQL Packages and Types Reference</i> for more information about using calendering syntax.
	The expression specified is evaluated to determine the next time the refresh should run. If repeat_interval is not specified, the job runs only once at the specified start date.
	The start_date and repeat_interval are used to create a refresh job by using DBMS_SCHEDULER package with end_date default as NULL.
Comments	This attribute specifies a comment about the job. By default, this attribute is $\mathtt{NULL}\xspace$

```
SYS.XS_DATA_SECURITY_UTIL.SCHEDULE_STATIC_ACL_REFRESH('aclmvuser', 'sales', SYSTIMESTAMP, 'freq=hourly; interval=2');
```

# 11.5.3.2 ALTER\_STATIC\_ACL\_REFRESH Procedure

The ALTER\_STATIC\_ACL\_REFRESH procedure is used to change the ACL refresh mode to oncommit or on-demand refresh.

### **Syntax**

#### **Parameters**

Parameter	Description
schema_name	Specifies the name for the schema that the table belongs to.
table_name	The table name, which is used with the schema name to uniquely identify a table for altering the static ACL refreshment mode.
refresh_mode	ON COMMIT or ON DEMAND

## **Examples**

```
SYS.XS_DATA_SECURITY_UTIL.ALTER_STATIC_ACL_REFRESH('aclmvuser','sales',
refresh_mode=>'ON COMMIT');
```

# 11.6 XS\_DIAG Package

The XS\_DIAG package includes subprograms to diagnose potential problems in data security for principals, security classes, acls, data security policies, namespaces, and all objects in the work space. All subprograms return TRUE if the object is valid; otherwise, each returns FALSE. For each identified inconsistency, a row is inserted into the XS\$VALIDATION TABLE validation

table until the maximum number of inconsistencies you specify with the <code>error\_limit</code> parameter is reached. Users can query this validation table to determine the identified inconsistencies for information that includes the message code, the description about the error, the path leading to the invalid object, and any other helpful information that might assist you in identifying the nature of the inconsistency.

This section includes the following topics:

- Security Model
- Summary of XS\_DIAG Subprograms

# 11.6.1 Security Model

The XS\_DIAG package is created in the SYS schema. The caller has invoker's rights on this package and needs to have ADMIN\_ANY\_SEC\_POLICY system privilege to run the XS\_DIAG package. EXECUTE permission on the XS\_DIAG package is granted to PUBLIC. SELECT permission on the XS\$VALIDATION TABLE validation table is granted to PUBLIC.

# 11.6.2 Summary of XS\_DIAG Subprograms

Table 11-8 Summary of XS\_DIAG Subprograms

Subprogram	Description
VALIDATE_PRINCIPAL Function	Validates the principal.
VALIDATE_SECURITY_CLASS Function	Validates the security class.
VALIDATE_ACL Function	Validates the ACL.
VALIDATE_DATA_SECURITY Function	Validates the data security policy or validates the data security policy against a specific table.
VALIDATE_NAMESPACE_TEMPLATE Function	Validates the namespace template.
VALIDATE_WORKSPACE Function	Validates an entire workspace.

This section describes the following XS\_DIAG subprograms:

# 11.6.2.1 VALIDATE PRINCIPAL Function

The VALIDATE\_PRINCIPAL function validates the principal. This function returns TRUE if the object is valid; otherwise, it returns FALSE. For each identified inconsistency, a row is inserted into the XS\$VALIDATION\_TABLE validation table until the maximum number of inconsistencies that can be stored is reached. Users must query this validation table to find out what caused the validation failure.

#### **Syntax**



#### **Parameters**

Parameter	Description
name	The name of the object to be validated.
error_limit	The maximum number of inconsistencies that may be stored in the validation table.

#### **Examples**

Validate the principal, user user1, then query the validation table in case there are inconsistencies.

```
begin
  if sys.xs_diag.validate_principal('user1', 100) then
    dbms_output.put_line('The user is valid.');
  else
    dbms_output.put_line('The user is invalid.');
  end if;
end;
/
select * from xs$validation table;
```

Validate the principal, role role1, then query the validation table in case there are inconsistencies.

```
begin
  if sys.xs_diag.validate_principal('role1', 100) then
    dbms_output.put_line('The role is valid.');
  else
    dbms_output.put_line('The role is invalid.');
  end if;
end;
/
select * from xs$validation table;
```

# 11.6.2.2 VALIDATE\_SECURITY\_CLASS Function

The VALIDATE\_SECURITY\_CLASS function validates the security class. This function returns TRUE if the object is valid; otherwise, it returns FALSE. For each identified inconsistency, a row is inserted into the XS\$VALIDATION\_TABLE validation table until the maximum number of inconsistencies that can be stored is reached. Users must query this validation table to find out what caused the validation failure.

#### **Syntax**

Parameter	Description
name	The name of the object to be validated.



Parameter	Description
error_limit	The maximum number of inconsistencies that may be stored in the validation table.

Validate the security class, sec1, then query the validation table in case there are inconsistencies.

```
begin
  if sys.xs_diag.validate_security_class('sec1', 100) then
    dbms_output.put_line('The security class is valid.');
  else
    dbms_output.put_line('The security class is invalid.');
  end if;
end;
/
select * from xs$validation table;
```

# 11.6.2.3 VALIDATE ACL Function

The VALIDATE\_ACL function validates the ACL. This function returns TRUE if the object is valid; otherwise, it returns FALSE. For each identified inconsistency, a row is inserted into the XS\$VALIDATION\_TABLE validation table until the maximum number of inconsistencies that can be stored is reached. Users must query this validation table to find out what caused the validation failure.

### **Syntax**

#### **Parameters**

Parameter	Description
name	The name of the object to be validated.
error_limit	The maximum number of inconsistencies that may be stored in the validation table.

#### **Examples**

Validate the ACL, acl1, then query the validation table in case there are inconsistencies.

```
begin
  if sys.xs_diag.validate_acl('acl1', 100) then
    dbms_output.put_line('The ACL is valid.');
  else
    dbms_output.put_line('The ACL is invalid.');
  end if;
end;
/
select * from xs$validation_table;
```



# 11.6.2.4 VALIDATE DATA SECURITY Function

The VALIDATE\_DATA\_SECURITY function validates the data security. This function returns TRUE if the object is valid; otherwise, it returns FALSE. For each identified inconsistency, a row is inserted into the XS\$VALIDATION\_TABLE validation table until the maximum number of inconsistencies that can be stored is reached. Users must query this validation table to find out what caused the validation failure.

This function has three styles of policy validation.

- When policy is not NULL and table\_name is NULL, the function validates the policy
  against all the tables to which the policy is applied. Note that when table\_name is NULL,
  table\_owner is ignored even if it is not NULL.
- When both policy and table\_name are not NULL, the function validates the policy against the specific table. If table owner is not provided, the current schema is used.
- When policy is NULL and table\_name is not NULL, the function validates all policies
  applied to the table against the table. If table\_owner is not provided, the current schema
  is used.

### **Syntax**

### **Parameters**

Parameter	Description
policy	The name of the object to be validated.
table_owner	The name of the schema of the table or view.
table_name	The name of the table or view.
error_limit	The maximum number of inconsistencies that may be stored in the validation table.

#### **Examples**

Validate a policy, policy1 on all the applied tables, then query the validation table in case there are inconsistencies.



Validate a policy, policy1 on a given table, then query the validation table in case there are inconsistencies.

Validate all the policies applied to a given table, then query the validation table in case there are inconsistencies.

# 11.6.2.5 VALIDATE\_NAMESPACE\_TEMPLATE Function

The VALIDATE\_NAMESPACE\_TEMPLATE function validates the namespace. This function returns TRUE if the object is valid; otherwise, it returns FALSE. For each identified inconsistency, a row is inserted into the XS\$VALIDATION\_TABLE validation table until the maximum number of inconsistencies that can be stored is reached. Users must query this validation table to find out what caused the validation failure.

### **Syntax**

#### **Parameters**

Parameter	Description
name	The name of the object to be validated.
error_limit	The maximum number of inconsistencies that may be stored in the validation table.

#### **Examples**

Validate the namespace, ns1, then query the validation table in case there are inconsistencies.



```
begin
  if sys.xs_diag.validate_namespace_template('ns1', 100) then
    dbms_output.put_line('The namespace template is valid.');
  else
    dbms_output.put_line('The namespace template is invalid.');
  end if;
end;
/
select * from xs$validation table;
```

# 11.6.2.6 VALIDATE WORKSPACE Function

The VALIDATE\_WORKSPACE function validates all the artifacts, in other words, it validates all objects that exist in the work space by using this one function. This function returns TRUE if all the objects are valid; otherwise, it returns FALSE. For each identified inconsistency, a row is inserted into the XS\$VALIDATION\_TABLE validation table until the maximum number of inconsistencies that can be stored is reached. Users must query this validation table to find out what caused the validation failure.

### **Syntax**

```
validate_workspace(error_limit IN PLS_INTEGER := 1)
    RETURN BOOLEAN;
```

#### **Parameters**

Parameter	Description
error_limit	The maximum number of inconsistencies that may be stored in the validation table.

#### **Examples**

Validate all the objects in the workspace, then query the validation table in case there are inconsistencies.

```
begin
  if sys.xs_diag.validate_workspace(100) then
    dbms_output.put_line('The objects are valid.');
  else
    dbms_output.put_line('The objects are invalid.');
  end if;
end;
/
select * from xs$validation table;
```

# 11.7 XS\_NAMESPACE Package

The  ${\tt XS\_NAMESPACE}$  package includes subprograms to create, manage, and delete namespace templates and attributes.

This section includes the following topics:

- Security Model
- Constants
- Object Types, Constructor Functions, Synonyms, and Grants

### Summary of XS\_NAMESPACE Subprograms

# 11.7.1 Security Model

The XS\_NAMESPACE package is created under the SYS schema. The DBA role is granted the ADMIN\_ANY\_SEC\_POLICY, which allows it to administer namespace templates and attributes.

# 11.7.2 Constants

The following are attribute event constants:

```
NO_EVENT CONSTANT PLS_INTEGER := 0;
FIRSTREAD_EVENT CONSTANT PLS_INTEGER := 1;
UPDATE_EVENT CONSTANT PLS_INTEGER := 2;
FIRSTREAD PLUS UPDATE EVENT CONSTANT PLS_INTEGER := 3;
```

# 11.7.3 Object Types, Constructor Functions, Synonyms, and Grants

The following object types, constructor functions, synonyms, and GRANT statements are defined for this package.

```
-- Type definition for namespace template attribute
CREATE OR REPLACE TYPE XS$NS ATTRIBUTE AS OBJECT (
-- Member Variables
-- Name of the namespace template attribute
-- Must be unique within a namespace template
-- Cannot be null
name VARCHAR2(4000),
-- Default value assigned to the attribute
default value VARCHAR2(4000),
-- Trigger events associated with the attribute
-- Allowed values are :
-- 0 : NO EVENT
-- 1 : FIRST READ EVENT
-- 2 : UPDATE EVENT
-- 3 : FIRST READ PLUS UPDATE EVENT
attribute events NUMBER,
-- Constructor function
CONSTRUCTOR FUNCTION XS$NS_ATTRIBUTE
                   (name IN VARCHAR2,
                    attribute events IN NUMBER := 0)
                    RETURN SELF AS RESULT,
-- Return the name of the attribute
MEMBER FUNCTION GET NAME RETURN VARCHAR2,
-- Return the default value of the attribute
MEMBER FUNCTION GET_DEFAULT_VALUE RETURN VARCHAR2,
-- Return the trigger events associated with attribute
MEMBER FUNCTION GET ATTRIBUTE EVENTS RETURN NUMBER,
-- Mutator procedures
-- Set the default value for the attribute
MEMBER PROCEDURE SET DEFAULT VALUE(default value IN VARCHAR2),
-- Associate trigger events to the attribute
MEMBER PROCEDURE SET ATTRIBUTE EVENTS (attribute events IN NUMBER)
```



```
);
CREATE OR REPLACE TYPE XS$NS_ATTRIBUTE_LIST AS VARRAY(1000) OF XS$NS_ATTRIBUTE;
```

# 11.7.4 Summary of XS\_NAMESPACE Subprograms

Table 11-9 Summary of XS\_NAMESPACE Subprograms

Subprogram	Description
CREATE_TEMPLATE Procedure	Creates a new namespace template.
ADD_ATTRIBUTES Procedure	Adds one or more attributes to an existing namespace template.
REMOVE_ATTRIBUTES Procedure	Removes one or more attributes from a namespace template.
SET_HANDLER Procedure	Assigns a handler function for the specified namespace template.
SET_DESCRIPTION Procedure	Sets a description string for the specified namespace template.
DELETE_TEMPLATE Procedure	Deletes the specified namespace template.

This section describes the following XS\_NAMESPACE subprograms:

# 11.7.4.1 CREATE\_TEMPLATE Procedure

The CREATE\_TEMPLATE procedure creates a new namespace template.

## **Syntax**

Parameter	Description
name	The name of the namespace template to be created.
attr_list	The attributes contained in the namespace template together with their default values and associated attribute events, such as <code>UPDATE_EVENT</code> .
schema	The schema that contains the handler function for the namespace template.
package	The package that contains the handler function for the namespace template.
function	The handler function for the namespace template. The handler function is called when an attribute event occurs.
acl	The name of the ACL for this namespace template. If no ACL is provided, the default is the predefined ACL SYS.NS_UNRESTRICTED_ACL, which allows unrestricted attribute operations by the application user.
description	An optional description string for the namespace template.



The following example creates a namespace template called POAttrs. The namespace template contains a list of attributes defined by attrlist. The handler function for the namespace template is called Populate\_Order\_Func. This handler function is part of the Orders\_Pckg package, which is contained in the SCOTT schema. The namespace template has NS\_UNRESTRICTED\_ACL set on the template, which allows unrestricted operation on namespaces created from the template.

# 11.7.4.2 ADD ATTRIBUTES Procedure

The ADD\_ATTRIBUTES procedure adds one or more attributes to an existing namespace template.

#### **Syntax**

### **Parameters**

Parameter	Description
template	The name of the namespace templates to which the attribute(s) is/are to be added.
attribute	The name of the attribute to be added.
attr_list	The list of attributes to be added.
default_value	The default value of the attribute.
attribute_events	The attribute event associated with the attribute, such as update event.

### **Examples**

The following example adds an attribute called <code>item\_type</code> to the <code>POAttrs</code> namespace. It also specifies a default value and attribute event for the new attribute that is added.

### 11.7.4.3 REMOVE\_ATTRIBUTES Procedure

The REMOVE\_ATTRIBUTES procedure removes one or more attributes from a namespace template. If no attribute names are specified, then all attributes are removed from the namespace template.

### **Syntax**

```
XS_NAMESPACE.REMOVE_ATTRIBUTES (
  template IN VARCHAR2,
  attribute IN VARCHAR2);

XS_NAMESPACE.REMOVE_ATTRIBUTES (
  template IN VARCHAR2,
  attr_list IN XS$LIST);

XS_NAMESPACE.REMOVE_ATTRIBUTES (
  template IN VARCHAR2);
```

### **Parameters**

Parameter	Description
template	The name of the namespace template from which the attribute(s) is/are to be removed.
attribute	The name of the attribute to be removed.
attr_list	The list of attribute names to be removed.

### **Examples**

The following example removes the item type attribute from the POAttrs namespace.

```
BEGIN
    SYS.XS_NAMESPACE.REMOVE_ATTRIBUTES('POAttrs','item_type');
END;
```

The following example removes all attributes from the POAttrs namespace template.

```
BEGIN
   SYS.XS_NAMESPACE.REMOVE_ATTRIBUTES('POAttrs');
END;
```

## 11.7.4.4 SET\_HANDLER Procedure

The  $\mathtt{SET\_HANDLER}$  procedure assigns a handler function for the specified namespace template.

```
XS_NAMESPACE.SET_HANDLER (
template IN VARCHAR2,
schema IN VARCHAR2,
```



package	IN	VARCHAR2,
function	IN	VARCHAR2);

Parameter	Description
template	The name of the namespace template for which the handler function is to be set.
schema	The schema containing the handler package and function.
package	The name of the package that contains the handler function.
function	The name of the handler function for the namespace template.

### **Examples**

The following example sets a handler function, called Populate\_Order\_Func, for the POAttrs namespace template.

### 11.7.4.5 SET DESCRIPTION Procedure

The  ${\tt SET\_DESCRIPTION}$  procedure sets a description string for the specified namespace template.

### **Syntax**

### **Parameters**

Parameter	Description
template	The name of the namespace template whose description is to be set.
description	A description string for the specified namespace template.

### **Examples**

The following example sets a description string for the POAttrs namespace template.

```
BEGIN
    SYS.XS_NAMESPACE.SET_DESCRIPTION('POAttrs','Purchase Order Attributes');
END;
```

## 11.7.4.6 DELETE\_TEMPLATE Procedure

The Delete template procedure deletes the specified namespace template.



### **Syntax**

#### **Parameters**

Parameter	Description
template	The name of the namespace template to be deleted.
delete_option	The delete option to use. To the namespace template, the behavior of the following options is the same:
	• DEFAULT_OPTION:
	The default option allows deleting a namespace template only if it is not referenced elsewhere. If there are other entities that reference the namespace template, then the namespace template cannot be deleted.  • CASCADE_OPTION:
	The cascade option deletes the namespace template together with any references to it. The user deleting the namespace template deletes these references as well.  ALLOW_INCONSISTENCIES_OPTION:
	The allow inconsistencies option lets you delete the entity even if other entities have late binding references to it. If the entity is part of an early dependency, then the delete fails and an error is raised.

### **Examples**

The following example deletes the POAttrs namespace template using the default delete option.

```
BEGIN
SYS.XS_NAMESPACE.DELETE_TEMPLATE('POAttrs', XS_ADMIN_UTIL.DEFAULT_OPTION);
END;
```

# 11.8 XS\_PRINCIPAL Package

The XS\_PRINCIPAL package contains procedures used to create, manage, and delete application principals. These application principals include application users, regular application roles, and dynamic application roles.

This section includes the following topics:

- Security Model
- Constants
- Object Types, Constructor Functions, Synonyms, and Grants
- Summary of XS\_PRINCIPAL Subprograms

## 11.8.1 Security Model

The XS PRINCIPAL package is created under the SYS schema.

Users with Real Application Security PROVISION privilege can create, modify, or drop application users and roles. The privileges required to create, modify, or drop application users and roles are no longer governed by the same system privileges required to create, modify, or drop database users and roles.

### 11.8.2 Constants

The following constants define the user's status:

```
ACTIVE CONSTANT PLS_INTEGER := 1;
INACTIVE CONSTANT PLS_INTEGER := 2;
UNLOCKED CONSTANT PLS_INTEGER := 3;
EXPIRED CONSTANT PLS_INTEGER := 4;
LOCKED CONSTANT PLS_INTEGER := 5;
```

### The following constants define dynamic role scope:

```
SESSION_SCOPE CONSTANT PLS_INTEGER := 0;
REQUEST SCOPE CONSTANT PLS INTEGER := 1;
```

### The following constants define the verifier type:

## 11.8.3 Object Types, Constructor Functions, Synonyms, and Grants

The following object types, constructor functions, synonyms, and GRANT statements are defined for this package.

```
-- Type definition for roles granted to the principals
CREATE OR REPLACE TYPE XS$ROLE GRANT TYPE AS OBJECT (
-- Member Variables
-- Constants defined in other packages cannot be recognized in a type.
-- e.g. XS ADMIN UTIL.XSNAME MAXLEN
-- name VARCHAR2(XS ADMIN UTIL.XSNAME MAXLEN),
              VARCHAR2 (130),
-- Start date of the effective date
 start date TIMESTAMP WITH TIME ZONE,
-- End date of the effective date
  end date TIMESTAMP WITH TIME ZONE,
  CONSTRUCTOR FUNCTION XS$ROLE GRANT TYPE (
   name IN VARCHAR2,
   start date IN TIMESTAMP WITH TIME ZONE:= NULL,
   end date IN TIMESTAMP WITH TIME ZONE:= NULL)
  RETURN SELF AS RESULT,
 MEMBER FUNCTION get role name RETURN VARCHAR2,
 MEMBER PROCEDURE set start date(start date IN TIMESTAMP WITH TIME ZONE),
 MEMBER FUNCTION get start date RETURN TIMESTAMP WITH TIME ZONE,
 MEMBER PROCEDURE set end date (end date IN TIMESTAMP WITH TIME ZONE),
 MEMBER FUNCTION get end date RETURN TIMESTAMP WITH TIME ZONE
CREATE OR REPLACE TYPE XS$ROLE GRANT LIST AS VARRAY(1000) OF XS$ROLE GRANT TYPE;
```

# 11.8.4 Summary of XS\_PRINCIPAL Subprograms

Table 11-10 Summary of XS\_PRINCIPAL Subprograms

Subprogram	Description
CREATE_USER Procedure	Creates an application user.
CREATE_ROLE Procedure	Creates an application role.
CREATE_DYNAMIC_ROLE Procedure	Creates a dynamic application role.
GRANT_ROLES Procedure	Grants one or more application roles to an application principal.
REVOKE_ROLES Procedure	Revokes one or more roles from an application principa
ADD_PROXY_USER Procedure	Adds a proxy user for a target application user.
REMOVE_PROXY_USERS Procedure	Removes specified proxy user or all proxy users for a target application user.
ADD_PROXY_TO_DBUSER	Add a proxy application user to a database user.
REMOVE_PROXY_FROM_DBUSER Procedure	Remove a proxy application user from a database user.
SET_EFFECTIVE_DATES Procedure	Sets or modifies the effective dates for an application user or role.
SET_DYNAMIC_ROLE_DURATION Procedure	Sets or modifies the duration, in minutes, for a dynamic application role.
SET_DYNAMIC_ROLE_SCOPE Procedure	Sets or modifies the scope of a dynamic application role such as REQUEST_SCOPE or SESSION_SCOPE.
ENABLE_BY_DEFAULT Procedure	Enables or disables an application role.
ENABLE_ROLES_BY_DEFAULT Procedure	Enables or disables all directly granted roles for the specified user.
SET_USER_SCHEMA Procedure	Sets the database schema for an application user.
SET_GUID Procedure	Sets the GUID for an external user or role.
SET_ACL Procedure	Sets the Real Application Security session privilege for an application user or a dynamic role.
SET_PROFILE Procedure	Sets the application user's profile. This is a set of resource limits and password parameters that restrict database usage and database instance resources for a Real Application Security application user.
SET_USER_STATUS Procedure	Sets or modifies the status of an application user account, such as ACTIVE, INACTIVE, UNLOCK, LOCKED or EXPIRED.
SET_PASSWORD Procedure	Sets or modifies the password for an application user account.
SET_VERIFIER Procedure	Sets or modifies the verifier for an application user account.
SET_DESCRIPTION Procedure	Sets the description string for an application user or role
DELETE_PRINCIPAL Procedure	Drops an application user or role.

This section describes the following XS\_PRINCIPAL subprograms:

## 11.8.4.1 CREATE\_USER Procedure

The CREATE\_USER procedure creates a new application user. You need the CREATE USER system privilege to create an application user.

You can use the DBA\_XS\_USERS data dictionary view to get a list of all application users.

### **Syntax**

```
CREATE_USER (
name IN VARCHAR2,
schema IN VARCHAR2 := NULL,
status IN PLS_INTEGER := ACTIVE,
start_date IN TIMESTAMP WITH TIME ZONE := NULL,
end_date IN TIMESTAMP WITH TIME ZONE := NULL,
guid IN RAW := NULL,
external_source IN VARCHAR2 := NULL,
description IN VARCHAR2 := NULL,
acl IN VARCHAR2 := NULL);
```

### **Parameters**

Parameter	Description
name	The name of the application user to be created.
status	The status of the user on creation. This can be one of the following values:
	ACTIVE, INACTIVE.
	The default value is ACTIVE.
	The values PASSWORDEXPIRED and LOCKED are deprecated beginning with Oracle Database Release 12.1 (12.1.0.2).
schema	The database schema to be associated with the user. This is optional.
start_date	The date from which the user account becomes effective. This is optional.
end_date	The date on which the user account becomes ineffective. This is optional.
	If an end_date is specified, then the start_date must also be specified.
guid	GUID of the user. This is valid for external users only.
external_source	Name of the system that is the source for this user. This is optional.
description	A description for the user account. This is optional.



Parameter	Description
acl	The Real Application Security session privilege. The default value is NULL meaning no ACL is set on the principal. The ACL must reside in the SYS schema, or else an error is thrown.
	The Real Application Security session privilege to be set on the principal must follow the naming convention for Real Application Security objects and must exist before this procedure is called.
	The session privilege is enforced as per the ACL set on the Real Application Security application user involved in the session operation. For example, a create session operation requires the caller to have the CREATE SESSION privilege in the ACL set on the Real Application Security application user.
	Principal-specific ACL grants take precedence over existing system-level session privilege grants. A privilege check is first done in the ACL associated with the principal and if it succeeds, the operation proceeds. If the privilege check finds deny, the operation fails with an insufficient privilege error. If neither grant nor deny is found, the check is done in the system ACL associated with the SESSION_SC security class and the operation succeeds or fails based on this privilege check result.

### **Examples**

The following example creates a user:

```
BEGIN
SYS.XS_PRINCIPAL.CREATE_USER('TEST1');
END;
```

The following example creates a user, and also specifies a schema and start date for the user:

## 11.8.4.2 CREATE\_ROLE Procedure

The CREATE\_ROLE procedure creates a new application role. You need the CREATE ROLE system privilege to create a regular application role.

You can use the DBA\_XS\_ROLES data dictionary view to get the list of application roles together with their attributes, like start date and end date

```
CREATE_ROLE ( name IN VARCHAR2, enabled IN BOOLEAN := FALSE, start_date IN TIMESTAMP WITH TIME ZONE := NULL, end_date IN TIMESTAMP WITH TIME ZONE := NULL, guid IN RAW := NULL,
```



```
external_source IN VARCHAR2 := NULL,
description IN VARCHAR2 := NULL);
```

Parameter	Description
name	The name of the application role to be created.
enabled	Specifies whether the role is enabled on creation. The default value is FALSE, which means that the role is disabled on creation.
start_date	The date from which the role becomes effective. This is optional.
end_date	The date on which the role becomes ineffective. This is optional.
	If an end_date is specified, then the start_date must also be specified.
guid	GUID of the role. This is applicable for external roles only.
external_source	The name of the system that is the source for this role. This is optional.
description	An optional description for the role.

### **Examples**

The following example creates an application role, called hrmgr:

```
BEGIN
   SYS.XS_PRINCIPAL.CREATE_ROLE('hrmgr');
END;
```

The following example creates an application role called hrrep. It also enables the role, and assigns the current date as start date for the role.

### 11.8.4.3 CREATE\_DYNAMIC\_ROLE Procedure

The <code>CREATE\_DYNAMIC\_ROLE</code> procedure creates a new dynamic application role. Dynamic application roles can be dynamically enabled or disabled by an application, based on the criteria defined by the application. You need the <code>CREATE ROLE</code> system privilege to create an dynamic application role.

You can use the DBA\_XS\_DYNAMIC\_ROLES data dictionary view to get a list of all dynamic application roles together with their attributes, like duration.



```
description IN VARCHAR2 := NULL,
acl IN VARCHAR2 := NULL);
```

Parameter	Description
name	The name of the dynamic application role to be created.
duration	The duration (in minutes) of the dynamic application role. This is an optional attribute.
scope	The scope attribute of the dynamic application role. The possible values are SESSION_SCOPE and REQUEST_SCOPE. The default value is XS_PRINCIPAL.SESSION_SCOPE.
description	An optional description for the dynamic application role.
acl	The Real Application Security session privilege. The default value is <code>NULL</code> meaning no ACL is set on the principal. The ACL must reside in the <code>SYS</code> schema, or else an error is thrown.
	The Real Application Security session privilege to be set on the principal must follow the naming convention for Real Application Security objects and must exist before this procedure is called.
	The session privilege is enforced as per the ACL set on the Real Application Security dynamic role involved in the session operation. For example, the attach operation with dynamic role requires the <code>ENABLE_DYNAMIC_ROLE</code> privilege in the ACLs to be set on the dynamic roles.
	Principal-specific ACL grants take precedence over existing system-level session privilege grants. A privilege check is first done in the ACL associated with the principal and if it succeeds, the operation proceeds. If the privilege check finds deny, the operation fails with an insufficient privilege error. If neither grant nor deny is found, the check is done in the system ACL associated with the SESSION_SC security class and the operation succeeds or fails based on this privilege check result.

### **Examples**

The following example creates a dynamic application role, called sslrole:

```
BEGIN
   SYS.XS_PRINCIPAL.CREATE_DYNAMIC_ROLE('sslrole');
END;
```

The following example creates a dynamic application role called reprole. It also specifies a duration of 100 minutes for the role, and chooses the request scope for the role.

### 11.8.4.4 GRANT\_ROLES Procedure

The <code>GRANT\_ROLES</code> procedure grants one or more application roles to an application principal. You need the <code>GRANT\_ANY\_ROLE</code> system privilege to grant application roles.

You can use the DBA\_XS\_ROLE\_GRANTS data dictionary view to get the list of all role grants together with their details, like start date and end date.

### **Syntax**

#### **Parameters**

Parameter	Description
grantee	The name of the principal to which the role is granted.
role	The name of the role to be granted.
role_list	The list of roles to be granted.
start_date	The date on which the grant takes effect. This is an optional parameter.
end_date	The date until which the grant is in effect. This is an optional parameter.

### **Examples**

The following example grants the HRREP role to user SMAVRIS with a start date and an end date specified:

The following example grants the HRREP and HRMGR roles to user SMAVRIS:

The following example shows how to grant the role XSCONNECT to user XSUSER. This grant will allow user XSUSER using its password to connect to a database.

```
EXEC SYS.XS_PRINCIPAL.GRANT_ROLES('XSUSER', 'XSCONNECT');
```



### 11.8.4.5 REVOKE ROLES Procedure

The REVOKE\_ROLES procedure revokes the specified role(s) from the specified grantee. If no roles are specified, then all application roles are revoked from the grantee. You need the GRANT ANY ROLE system privilege to grant or revoke roles.

You can use the DBA\_XS\_ROLE\_GRANTS data dictionary view to get the list of all role grants together with their details, like start date and end date.

### **Syntax**

```
REVOKE_ROLES (
  grantee IN VARCHAR2,
  role IN VARCHAR2);

REVOKE_ROLES (
  grantee IN VARCHAR2,
  role_list IN XS$NAME_LIST);

REVOKE_ROLES (
  grantee IN VARCHAR2);
```

#### **Parameters**

Parameter	Description
grantee	The application principal from whom the role(s) are to be revoked.
role	The name of the application role that is to be revoked.
role_list	The list of role names that are to be revoked.

### **Examples**

The following example revokes the HRREP role from user SMAVRIS:

```
BEGIN
   XS_PRINCIPAL.REVOKE_ROLES('SMAVRIS','HRREP');
END;
```

The following example revokes the HRREP and HRMGR roles from user SMAVRIS:

```
DECLARE
  role_list XS$NAME_LIST;
BEGIN
  role_list := XS$NAME_LIST('HRREP','HRMGR');
  SYS.XS_PRINCIPAL.REVOKE_ROLES('SMAVRIS', role_list);
END;
```

The following example revokes all granted roles from user SMAVRIS:

```
BEGIN
   SYS.XS_PRINCIPAL.REVOKE_ROLES('SMAVRIS');
END;
```

### 11.8.4.6 ADD PROXY USER Procedure

The ADD\_PROXY\_USER adds a target user for the specified application user. This allows the application user to proxy as the target user. There are two signatures for this procedure. The

first signature allows you to specify a subset of roles of the target user using the target\_roles parameter that are to be assigned to the proxy user. For the second signature there is no target\_roles parameter, so all roles of the target user are assigned to the proxy user.

You need the ALTER USER system privilege to add or remove a proxy user.

### **Syntax**

```
ADD_PROXY_USER (
target_user IN VARCHAR2,
proxy_user IN VARCHAR2,
target_roles IN XS$NAME_LIST);

ADD_PROXY_USER (
target_user IN VARCHAR2,
proxy_user IN VARCHAR2);
```

#### **Parameters**

Parameter	Description
target_user	The name of the target application user that can be proxied to.
proxy_user	The name of the proxy application user.
target_roles	A list of target user roles that can be proxied by the proxy user. This parameter is mandatory. If you pass an explicit NULL value, then this would be a case of configuring the proxy user without any role of the target user; otherwise, the proxy_user parameter uses the value you specify for the target_roles parameter.

#### **Examples**

The following example enables user DJONES to proxy as target user SMAVRIS. The target roles granted to DJONES are HRREP and HRMGR.

```
DECLARE
   pxy_roles XS$NAME_LIST;
BEGIN
   pxy_roles := XS$NAME_LIST('HRREP','HRMGR');
   SYS.XS_PRINCIPAL.ADD_PROXY_USER('SMAVRIS','DJONES', pxy_roles);
END;
```

The following example passes an explicit NULL value for the target role; in other words, it assigns no roles of the target user 'SMAVRIS' to the proxy user 'DJONES'.

```
BEGIN
    SYS.XS_PRINCIPAL.ADD_PROXY_USER('SMAVRIS','DJONES', NULL);
END;
```

The following example assigns all roles of target user 'SMAVRIS' to proxy user 'DJONES'.

```
BEGIN
   SYS.XS_PRINCIPAL.ADD_PROXY_USER('SMAVRIS','DJONES');
END;
```



### 11.8.4.7 REMOVE PROXY USERS Procedure

The REMOVE\_PROXY\_USERS procedure disassociates one or all proxy users for a target application user. The associated proxy roles are automatically removed for the proxy users.

You need the ALTER USER system privilege to add or remove a proxy user.

### **Syntax**

```
REMOVE_PROXY_USERS (
  target_user IN VARCHAR2);

REMOVE_PROXY_USERS (
  target_user IN VARCHAR2,
  proxy user IN VARCHAR2);
```

#### **Parameters**

Parameter	Description
target_user	The target application user whose proxies are to be disassociated.
proxy_user	The proxy application user that needs to be disassociated from the target user.

### **Examples**

The following example removes all proxy users for target user SMAVRIS:

```
BEGIN
   SYS.XS_PRINCIPAL.REMOVE_PROXY_USERS('SMAVRIS');
END;
```

The following example disassociates the proxy user DJONES from the target user SMAVRIS:

```
BEGIN
   SYS.XS_PRINCIPAL.REMOVE_PROXY_USERS('SMAVRIS','DJONES');
END;
```

## 11.8.4.8 ADD\_PROXY\_TO\_DBUSER

The ADD\_PROXY\_TO\_DBUSER adds the specified target proxy application user to the specified database user. The application user must be a direct logon user. This allows the application user to proxy as the target database user. By default, all roles assigned to the target user can be used by the proxy user. Similar to Oracle Database, the default roles of the target database users would be enabled after connection. Other roles assigned to the target database user can be set by using the SET\_ROLE statement.

You need the ALTER USER system privilege to add a proxy user to a database user.



Parameter	Description
database_user	The name of the target database user that can be proxied to.
proxy_user	The name of the proxy application user.
is_external	The parameter to indicate whether the user is an external user or a regular Real Application Security application user.

### **Examples**

The following example enables application user DJONES to proxy as target database user SMAVRIS.

```
BEGIN
   SYS.XS_PRINCIPAL.ADD_PROXY_TO_DBUSER('SMAVRIS','DJONES', TRUE);
END:
```

### 11.8.4.9 REMOVE\_PROXY\_FROM\_DBUSER Procedure

The REMOVE\_PROXY\_FROM\_DBUSER procedure disassociates a proxy application user from a database user. The associated proxy roles are automatically removed from the application user.

You need the  ${\tt ALTER}\,$   ${\tt USER}$  system privilege to remove a proxy user from a database user.

### **Syntax**

```
REMOVE_PROXY_FROM_DBUSER (
  database_user IN VARCHAR2,
  proxy user IN VARCHAR2);
```

#### **Parameters**

Parameter	Description
database_user	The target database user whose proxies are to be disassociated.
proxy_user	The proxy application user that needs to be disassociated from the target database user.

### **Examples**

The following example disassociates the proxy user DJONES from the target database user SMAVRIS:

```
BEGIN
    SYS.XS_PRINCIPAL.REMOVE_PROXY_FROM_DBUSER('SMAVRIS','DJONES');
END;
```

### 11.8.4.10 SET EFFECTIVE DATES Procedure

The SET\_EFFECTIVE\_DATES procedure sets or modifies the effective dates for an application user or role. If the start date and end date values are specified as NULL

by default, then the application user is not currently effective, so the session for the particular application user cannot be created.

You need the ALTER USER system privilege to run this procedure for an application user. You need the ALTER ANY ROLE system privilege to run this procedure for an application role.

### **Syntax**

### **Parameters**

Parameter	Description
principal	The name of the application user or role for which effective dates are to be set.
start_date	The start date of the effective dates period.
end_date	The end date of the effective dates period.

#### **Examples**

The following example sets the effective dates for user DJONES.

## 11.8.4.11 SET DYNAMIC ROLE DURATION Procedure

The SET\_DYNAMIC\_ROLE\_DURATION procedure sets or modifies the duration for a dynamic application role. The duration is specified in minutes.

You need the ALTER ANY ROLE system privilege to modify a role.

### **Syntax**

### **Parameters**

Parameter	Description
role	The name of the dynamic application role.



Parameter	Description
duration	The duration of the dynamic application role in minutes. This cannot be a negative value.

### **Examples**

The following example sets the duration of the reprole dynamic application role to 60 minutes.

```
BEGIN
    SYS.XS_PRINCIPAL.SET_DYNAMIC_ROLE_DURATION('reprole',60);
END.
```

### 11.8.4.12 SET DYNAMIC ROLE SCOPE Procedure

The SET\_DYNAMIC\_ROLE\_SCOPE procedure sets or modifies the scope of a dynamic application role. The session (SESSION\_SCOPE) or request (REQUEST\_SCOPE) scopes can be chosen.

You need the ALTER ANY ROLE system privilege to modify a role.

### **Syntax**

### **Parameters**

Parameter	Description
role	The name of the dynamic application role.
scope	The scope of the dynamic application role to be set. The allowed values are XS_PRINCIPAL.REQUEST_SCOPE and XS_PRINCIPAL.SESSION_SCOPE.

### **Examples**

The following example sets the scope of the reprole dynamic application role to request scope:

```
begin
   SYS.XS_PRINCIPAL.SET_DYNAMIC_ROLE_SCOPE('reprole', XS_PRINCIPAL.REQUEST_SCOPE);
end;
```

## 11.8.4.13 ENABLE\_BY\_DEFAULT Procedure

The ENABLE BY DEFAULT procedure enables or disables a regular application role.

If enabled, then the application role is automatically enabled for the principal to which it is granted. If disabled, then the privileges associated with the application role are not enabled even if the application role is granted to a principal.

You need the ALTER ANY ROLE system privilege to modify an application role.

### **Syntax**

### **Parameters**

Parameter	Description
role	The name of the regular application role.
enabled	The <code>enabled</code> attribute of the application role. Setting this to <code>TRUE</code> marks the application role as being enabled by default. The default value is <code>TRUE</code> .

#### **Examples**

The following example sets the enabled attribute for the HRREP application role to TRUE:

```
BEGIN
   SYS.XS_PRINCIPAL.ENABLE_BY_DEFAULT('HRREP',TRUE);
END;
```

## 11.8.4.14 ENABLE\_ROLES\_BY\_DEFAULT Procedure

The <code>ENABLE\_ROLES\_BY\_DEFAULT</code> procedure enables or disables all application roles that have been directly granted to an application user.

You need the ALTER USER system privilege to run this procedure for an application user.

### **Syntax**

#### **Parameters**

Parameter	Description
user	The name of the application user.
enabled	The enabled attribute for all application roles that have been directly granted to the application user.
	Setting the enabled attribute to TRUE enables all directly granted application roles for the application user. The default value is TRUE.
	Setting the enabled attribute to ${\tt FALSE}$ disables all directly granted application roles for the application user.

### **Examples**

The following example enables all directly granted roles for application user SMAVRIS:

```
BEGIN
   SYS.XS_PRINCIPAL.ENABLE_ROLES_BY_DEFAULT('SMAVRIS',TRUE);
END;
```



### 11.8.4.15 SET USER SCHEMA Procedure

The SET\_USER\_SCHEMA procedure sets the database schema for an application user.

You need the ALTER USER system privilege to run this procedure for an application user.

### **Syntax**

```
SET_USER_SCHEMA (
user IN VARCHAR2,
schema IN VARCHAR2);
```

### **Parameters**

Parameter	Description
user	The name of the application user.
schema	The name of the database schema to be associated with the user. Setting this to $\mathtt{NULL}$ removes any schema association.

### **Examples**

The following example associates the HR schema with user DJONES.

```
BEGIN
   SYS.XS_PRINCIPAL.SET_USER_SCHEMA('DJONES','HR');
END;
```

## 11.8.4.16 SET\_GUID Procedure

The  $\mathtt{SET\_GUID}$  procedure sets the GUID for a principal. The principal must be an external user or role, and the current GUID must be  $\mathtt{NULL}$ .

You need the ALTER USER system privilege to run this procedure for an application user. You need the ALTER ANY ROLE system privilege to run this procedure for an application role.

### Note:

The  ${\tt external\_source}$  attribute for the user must have been set for SET GUID to work.

```
SET_GUID (
  principal IN VARCHAR2,
  guid IN RAW);
```



Parameter	Description
principal	The name of the external user or role.
guid	The GUID for the external user or role.

### **Examples**

The following example sets a GUID for user Alex:

```
BEGIN
   SYS.XS_PRINCIPAL.SET_GUID('ALEX','7b6cb3a98f8a4e20ac31a37419cc7fa3');
END;
```

### 11.8.4.17 SET ACL Procedure

### **Purpose**

The SET ACL procedure sets an ACL on the specified application user or dynamic role.

This procedure requires the caller to have the Real Application Security PROVISION privilege as the least privilege. Users with database ALTER USER privilege can also call the procedure if the principal is an application user. Users with the database role ALTER ROLE privilege can also call this procedure if the principal is a dynamic role.

### **Syntax**

#### **Parameters**

Parameter	Description
principal	The application user or dynamic role to which the ACL is to be set.
acl	The Real Application Security session privilege.

### **Usage Notes**

The ACLs must be created in the SYS schema.

An ACL set on an application user or dynamic role overrides a system-wide ACL.

The session privilege will be enforced as per the ACL set on Real Application Security application user or dynamic role involved in the session operation. For example, a create session operation requires the caller to have the <code>CREATE\_SESSION</code> privilege in the ACL set on the Real Application Security application user or the attach operation with dynamic role requires the <code>ENABLE\_DYNAMIC\_ROLE</code> privilege in the ACLs to be set on the dynamic roles.

Principal-specific ACL grants take precedence over existing system-level session privilege grants. A privilege check is first done in the ACL associated with the principal and if it succeeds, the operation proceeds. If the privilege check finds deny, the operation fails with an

insufficient privilege error. If neither grant nor deny is found, the check is done in the system ACL associated with the <code>SESSION\_SC</code> security class and the operation succeeds or fails based on this privilege check result.

### **Examples**

# Example 11-1 Set the ACL Privilege CREATE\_SESSION on Application User TEST1

The following example sets the ACL privilege <code>CREATE\_SESSION</code> on the specified application user <code>test1</code>.

```
BEGIN
   SYS.XS_PRINCIPAL.SET_ACL('test1','CREATE_SESSION');
END:
```

### 11.8.4.18 SET PROFILE Procedure

The SET\_PROFILE procedure sets the application user's profile. The profile is a set of resource limits and password parameters that restrict database usage and database instance resources for a Real Application Security application user. Both the application user and the profile must be existing entities.

The user executing this procedure must have the ALTER USER privilege.

If a profile that is assigned to an application user is dropped using the cascade option, then the default profile would automatically become activated for that user.

### **Syntax**

```
SET_PROFILE (
  user     IN VARCHAR2,
  profile     IN VARCHAR2);
```

### **Parameters**

Parameter	Description
user	The name of the Real Application Security application user. This must be an existing application user.
profile	The name of the profile.

### **Examples**

The following example creates a profile named prof and then sets the profile named prof to an application user named xsuser.

```
CREATE PROFILE prof LIMIT PASSWORD_REUSE_TIME 1/1440 PASSWORD_REUSE_MAX 3
PASSWORD_VERIFY_FUNCTION Verify_Pass;

BEGIN
SYS.XS_PRINCIPAL.SET_PROFILE('xsuser','prof');

END:
```



## 11.8.4.19 SET\_USER\_STATUS Procedure

The  ${\tt SET\_USER\_STATUS}$  procedure sets or modifies the status of an application user account.

You need the  ${\tt ALTER\_USER}$  privilege to run this procedure for an application user.

### **Syntax**

### **Parameters**

Parameter	Description
user	The name of the user account whose status needs to be set or updated.
status	The new status of the Real Application Security user account. The status values can be divided into several classes:
	<ul> <li>ACTIVE and INACTIVE - These two account status values will affect the user account's ability to create and attach to an application session.</li> </ul>
	When set to ACTIVE, it allows the application user to use a direct login account to log into the database with a valid password. The application user is allowed to create and attach to an application session if the account has the required application privileges.
	When set to INACTIVE, the application user cannot use a direct login account to log into the database even with a valid password and can not create and attach to an application session.
	<ul> <li>UNLOCK, LOCKED, or EXPIRED - These status values will be checked only for the direct login Real Application Security application user.</li> </ul>
	When set to UNLOCK, it opens the application user account when the account is LOCKED and allows the application user to use a direct login account to log into the database with a valid password.
	When set to LOCKED, it locks the account of the application user. This means user connections using a direct login account will not be allowed even with a valid password. Provided that the user account is ACTIVE, a direct login will not succeed when the account is locked, but the user cal create and attach to an application session.
	When set to EXPIRED, it expires the account of the application user. This means user connections using a direct login account will be allowed for valid passwords; however, the password must be changed at the time of logon.
	<ul> <li>PASSWORDEXPIRED (Deprecated) - This status value is deprecated beginning with Release 1 (12.1.0.2).</li> </ul>
	If you try to pass any other value for the parameter status, an ORA-46152: XS Security - invalid user status specified error is returned.

### **Examples**

The following example sets the user status to LOCKED for user DJONES.

```
BEGIN
   SYS.XS_PRINCIPAL.SET_USER_STATUS('DJONES', XS_PRINCIPAL.LOCKED);
END;
```



### 11.8.4.20 SET PASSWORD Procedure

The SET\_PASSWORD procedure sets or modifies the password for an application user account. When you use the SET\_PASSWORD procedure, it creates a verifier for you based on the password and the type parameter and then inserts the verifier and the value of the type parameter into the dictionary table.

A direct login Real Application Security user can change their own password by providing its value using the oldpass parameter. If value of the old password is incorrect, then the failed login count is incremented with each attempt, returning an ORA-28008: invalid old password error. The new password is not set until the old supplied password is correct.

You need the ALTER\_USER privilege to run this procedure for an application user or if you are changing the password of other Real Application Security users.

Native Real Application Security users synchronized from external ID stores are not allowed to change their own password. These users must change their password in the originating ID store. For example, if the Oracle Internet Directory 11g Release 1 (11.1.1) is the external store, for end-user self-service use the Oracle Identity Self Service interface provided by Oracle Identity Manager to manage your passwords. See Fusion Middleware Performing Self Service Tasks with Oracle Identity Manager for more information. You should contact your security administrator to determine if native Real Application Security users are synchronized from an external ID store, and if so, whether password management is provided in your directory server environment for end-user self-service.

The SET\_PASSWORD operation and the SQL\*Plus PASSWORD command are both blocked on the logical standby database.

#### **Syntax**

### **Parameters**

Parameter	Description
user	The name of the application user account for which the password is to be set.
password	The password to be set.
type	The verifier type to be used for the password. The default value is XS_SHA512. The verifier type must be one of the following types:
	XS_SHA512, XS_SALTED_SHA1
opassword	The old password. This parameter is required if the Real Application Security user is changing their own password. If not provided, then the user must have the required privilege to change their own password.



### **Examples**

The following example sets a password for application user SMAVRIS. It also specifies the XS\_SHA512 verifier type for the password.

```
BEGIN
   SYS.XS_PRINCIPAL.SET_PASSWORD('SMAVRIS','2Hrd2Guess',XS_PRINCIPAL.XS_SHA512);
END;
```

## 11.8.4.21 SET\_VERIFIER Procedure

The SET\_VERIFIER procedure sets or modifies the verifier for an application user account. When you use the SET\_VERIFIER procedure, the procedure directly inserts the verifier and the value of the type parameter into the dictionary table, XS\$VERIFIERS. This enables administrators to migrate users into Real Application Security with knowledge of the verifier and not the password.

You need the ALTER USER privilege to run this procedure for an application user.

The SET\_VERIFIER operation and the SQL\*Plus PASSWORD command are both blocked on the logical standby database.

### **Syntax**

#### **Parameters**

Parameter	Description
user	The name of the application user for whom the verifier is set.
verifier	A character string to be used as the verifier.
type	The verifier type to be used. This can be one of the following:
	XS_SHA512, XS_SALTED_SHA1

### **Examples**

Assume that a user by the name LWUSER3 is created and the password is set with a verifier type of  $XS\_SALTED\_SHA1$ .

Next, query the view DBA XS OBJECTS to obtain the ID value for user LWUSER3.



The value of the verifier includes its type as value "S" followed by a colon (:) to denote that it is a verifier type of XS SALTED SHA1, which is also indicated as being of type# 1.

Using the entire verifier value including "S:", set the verifier for user LWUSER3.

```
BEGIN
SYS.XS_PRINCIPAL.SET_VERIFIER('lwuser3','S:14DC0F5ABB72FC869549B1F845C548E0BEF7B8
63A116DB24DFAE22F0501E',
XS_PRINCIPAL.XS_SALTED_SHA1);
END;
/ 2 3 4 5

PL/SQL procedure successfully completed.
```

For this procedure to complete successfully, both the verifier value and its type must match the information in the VERIFIER column of the XS\$VERIFIERS dictionary table for the user whose verifier is being set. Note that when you change the password for an application user, it automatically changes its verifier value with the option of changing its verifier type.

The previous example set the verifier to its same exact value to show the steps involved. You have the option to set the verifier for a password to any verifier value that displays for an application user when you query the XS\$VERIFIERS dictionary table as long as the verifier value matches the verifier type that you set. For example, if you wanted to change the verifier value and the verifier type to XS\_SHA512, do the following.

```
SQL> BEGIN
SYS.XS_PRINCIPAL.SET_VERIFIER('lwuser3','T:9BA95FEF2C2630A2BAACF2E7C5E4
1B0D50CDC7B0B6
0C88AD4FE81F8155D002F99EEAF9D95477E4749870C67FDE870E154ED17809C359777F9
79E269010823FB
981B2A998915EB1439FE3C6C1542A239C',
XS_PRINCIPAL.XS_SHA512);
END;
/ 2 3 4
```

PL/SQL procedure successfully completed.



Note that this is the same verifier value and verifier type that was set for application user LWUSER1 as shown in Setting a Password Verifier for Direct Application User Accounts.

## 11.8.4.22 SET\_DESCRIPTION Procedure

The SET DESCRIPTION procedure is used to set the description for an application principal.

You need the ALTER USER system privilege to run this procedure for an application user. You need the ALTER ANY ROLE system privilege to run this procedure for an application role.

### **Syntax**

```
SET DESCRIPTION ( principal IN VARCHAR2, description IN VARCHAR2);
```

#### **Parameters**

Parameter	Description
principal	The name of the principal for which the description is set.
description	A descriptive string about the principal.

### **Examples**

The following example sets a description for the application role HRREP:

```
BEGIN
   SYS.XS_PRINCIPAL.SET_DESCRIPTION('HRREP','HR Representative role');
END:
```

## 11.8.4.23 DELETE\_PRINCIPAL Procedure

The DELETE PRINCIPAL procedure drops an application user or application role.

You need the DROP USER system privilege to run this procedure for an application user. You need the DROP ANY ROLE system privilege to run this procedure for an application role.

### **Syntax**

```
delete_principal (
  principal IN VARCHAR2,
  delete_option IN PLS_INTEGER:=XS_ADMIN_UTIL.DEFAULT_OPTION);
```

### **Parameters**

Parameter	Description
principal	The name of the application user or application role that is to be deleted.



Parameter	Description
delete_option	The delete option to use. The following options are available:
	• DEFAULT_OPTION:
	The default option allows deleting a principal only if it is not referenced elsewhere. If there are other entities that reference the principal, then the principal cannot be deleted.
	For example, the delete operation fails if you try to delete an application role that is granted to a principal.  • CASCADE_OPTION:
	The cascade option deletes the application user or application role together with any references to it. The user deleting the application user or application role must have privileges to delete these references as well.
	• ALLOW_INCONSISTENCIES_OPTION:
	The allow inconsistencies option lets you delete the entity even if other entities have late binding references to it. If the entity is part of an early dependency, then the delete fails and an error is raised.

### **Examples**

The following example deletes the user SMAVRIS using the DEFAULT\_OPTION:

```
BEGIN
   SYS.XS_PRINCIPAL.DELETE_PRINCIPAL('SMAVRIS');
END:
```

# 11.9 XS SECURITY CLASS Package

The XS\_SECURITY\_CLASS package includes procedures to create, manage, and delete security classes and their privileges. The package also includes procedures for managing security class inheritance.

This section includes the following topics:

- Security Model for the XS\_SECURITY\_CLASS Package
- Summary of XS\_SECURITY\_CLASS Subprograms

## 11.9.1 Security Model for the XS\_SECURITY\_CLASS Package

The XS\_SECURITY\_CLASS package is created under the SYS schema. The DBA role is granted the ADMIN\_ANY\_SEC\_POLICY, which allows it to administer schema objects like ACLs, security classes, and security policies across all schemas.

Users can administer schema objects in their own schema if they have been granted the RESOURCE role for the schema. The RESOURCE role and the XS\_RESOURCE application role include the ADMIN\_SEC\_POLICY privilege, required to administer schema objects in the schema as well as administering the policy artifacts within the granted schema to achieve policy management within an application.

Users can administer policy enforcement on the schema if they have been granted the  $\texttt{APPLY\_SEC\_POLICY}$  privilege. With this privilege, the user can administer policy enforcement within granted schemas to achieve policy management within an application.



# 11.9.2 Summary of XS\_SECURITY\_CLASS Subprograms

Table 11-11 Summary of XS\_SECURITY\_CLASS Subprograms

Subprogram	Description
CREATE_SECURITY_CLASS Procedure	Creates a new security class.
ADD_PARENTS Procedure	Adds one or more parent security classes for the specified security class.
REMOVE_PARENTS Procedure	Removes one or more parent security classes for the specified security class.
ADD_PRIVILEGES Procedure	Adds one or more privileges to the specified security class.
REMOVE_PRIVILEGES Procedure	Removes one or more privileges for the specified security class.
ADD_IMPLIED_PRIVILEGES Procedure	Adds one or more implied privileges for the specified aggregate privilege.
REMOVE_IMPLIED_PRIVILEGES Procedure	Removes one or more implied privileges from an aggregate privilege.
SET_DESCRIPTION Procedure	Sets a description string for the specified security class.
DELETE_SECURITY_CLASS Procedure	Deletes the specified security class.

This section describes the following XS\_SECURITY\_CLASS subprograms:

## 11.9.2.1 CREATE\_SECURITY\_CLASS Procedure

The  ${\tt CREATE\_SECURITY\_CLASS}$  creates a new security class.

### **Syntax**

### **Parameters**

Parameter	Description
name	The name of the security class to be created.
	The name is schema qualified, for example, SCOTT.SC1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as SC1, and the current schema is SCOTT, it would resolve to SCOTT.SC1.
priv_list	The list of privileges to include in the security class.
parent_list	The list of parent security classes from which the security class is inherited. This is optional.
description	An optional description for the security class.



### **Examples**

The following example creates a security class called HRPRIVS. The security class includes a set of privileges defined in <code>priv\_list</code>. The security class uses the <code>DML</code> class as its parent security class.

## 11.9.2.2 ADD\_PARENTS Procedure

The ADD\_PARENTS procedure adds one or more parent security classes for the specified security class.

### **Syntax**

### **Parameters**

Parameter	Description
sec_class	The name of the security class for which parent classes are to be added.
	The name is schema qualified, for example, SCOTT.SC1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as SC1, and the current schema is SCOTT, it would resolve to SCOTT.SC1.
parent	The name of the parent security class to be added.
parent_list	The list of parent classes to be added.

### **Examples**

The following example adds the parent security class GENPRIVS to the HRPRIVS security class.

```
BEGIN
   SYS.XS_SECURITY_CLASS.ADD_PARENTS('HRPRIVS','GENPRIVS');
END;
```



### 11.9.2.3 REMOVE PARENTS Procedure

The REMOVE\_PARENTS procedure removes one or more parent classes for the specified security class.

### **Syntax**

```
XS_SECURITY_CLASS.REMOVE_PARENTS (
   sec_class IN VARCHAR2);

XS_SECURITY_CLASS.REMOVE_PARENTS (
   sec_class IN VARCHAR2,
   parent IN VARCHAR2);

XS_SECURITY_CLASS.REMOVE_PARENTS (
   sec_class IN VARCHAR2,
   parent list IN XS$NAME LIST);
```

### **Parameters**

Parameter	Description
sec_class	The name of the security class whose parent classes are to be removed.
	The name is schema qualified, for example, SCOTT.SC1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as SC1, and the current schema is SCOTT, it would resolve to SCOTT.SC1.
parent	The parent security class that is to be removed.
parent_list	The list of parent security classes that are to be removed.

### **Examples**

The following example removes the parent security class <code>GENPRIVS</code> from the <code>HRPRIVS</code> security class.

```
BEGIN
   SYS.XS_SECURITY_CLASS.REMOVE_PARENTS('HRPRIVS','GENPRIVS');
END:
```

## 11.9.2.4 ADD\_PRIVILEGES Procedure

The ADD PRIVILEGES procedure adds one or more privileges to a security class.



Parameter	Description
sec_class	The name of the security class to which the privileges are to be added.
	The name is schema qualified, for example, SCOTT.SC1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as SC1, and the current schema is SCOTT, it would resolve to SCOTT.SC1.
priv	The name of the privilege to be added.
priv_list	The list of privileges to be added.
<pre>implied_priv_list</pre>	An optional list of implied privileges to be added.
description	An optional description of the privilege being added.

### **Examples**

The following example adds an aggregate privilege called <code>UPDATE\_INFO</code> to the <code>HRPRIVS</code> security class. The aggregate privilege contains the implied privileges, <code>UPDATE</code>, <code>DELETE</code>, and <code>INSERT</code>.

## 11.9.2.5 REMOVE\_PRIVILEGES Procedure

The REMOVE\_PRIVILEGES procedure removes one or more privileges from the specified security class. If no privilege name or list is specified, then all privileges are removed from the specified security class.



Parameter	Description
sec_class	The name of the security class for which the privileges are to be removed.
	The name is schema qualified, for example, SCOTT.SC1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as SC1, and the current schema is SCOTT, it would resolve to SCOTT.SC1.
priv	The name of the privilege to be removed.
priv_list	The list of privileges to be removed.

### **Examples**

The following example removes the UPDATE INFO privilege from the HRPRIVS security class.

```
BEGIN
    SYS.XS_SECURITY_CLASS.REMOVE_PRIVILEGES('HRPRIVS','UPDATE_INFO');
END;
```

The following example removes all privileges from the HRPRIVS security class.

```
BEGIN
   SYS.XS_SECURITY_CLASS.REMOVE_PRIVILEGES('HRPRIVS');
END;
```

## 11.9.2.6 ADD\_IMPLIED\_PRIVILEGES Procedure

The ADD\_IMPLIED\_PRIVILEGES procedure adds one or more implied privileges to an aggregate privilege.

### **Syntax**

```
XS_SECURITY_CLASS.ADD_IMPLIED_PRIVILEGES (
  sec_class     IN VARCHAR2,
  priv          IN VARCHAR2,
  implied_priv IN VARCHAR2);

XS_SECURITY_CLASS.ADD_IMPLIED_PRIVILEGES (
  sec_class          IN VARCHAR2,
  priv                IN VARCHAR2,
  implied_priv_list IN XS$NAME_LIST);
```

### **Parameters**

Parameter	Description
sec_class	The name of the security class to which the privileges are to be added.
	The name is schema qualified, for example, SCOTT.SC1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as SC1, and the current schema is SCOTT, it would resolve to SCOTT.SC1.
priv	Name of the aggregate privilege for which the implied privileges are to be added.

Parameter	Description
implied_priv	The implied privilege to be added.
<pre>implied_priv_list</pre>	A list of implied privileges to be added for the aggregate privilege.

### **Examples**

The following example adds a list of implied privileges for the aggregate privilege UPDATE INFO to the HRPRIVS security class:

```
BEGIN
   SYS.XS_SECURITY_CLASS.ADD_IMPLIED_PRIVILEGES(sec_class=>'HRPRIVS',
priv=>'UPDATE_INFO', implied_priv_list=>XS$NAME_LIST('"UPDATE"', '"DELETE"',
'"INSERT"'));
END;
```

### 11.9.2.7 REMOVE\_IMPLIED\_PRIVILEGES Procedure

The REMOVE\_IMPLIED\_PRIVILEGES procedure removes the specified implied privileges from an aggregate privilege. If no implied privileges are specified, then all implied privileges are removed from the aggregate privilege.

### **Syntax**

### **Parameters**

Parameter	Description
sec_class	The name of the security class for which the privileges are to be removed.
	The name is schema qualified, for example, SCOTT.SC1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as SC1, and the current schema is SCOTT, it would resolve to SCOTT.SC1.
priv	The name of the aggregate privilege from which the implied privileges are to be removed.
implied_priv	The implied privilege to be removed from the aggregate privilege.
<pre>implied_priv_list</pre>	The list of implied privileges to be removed from the aggregate privilege.



### **Examples**

The following example removes the implicit privilege DELETE from the aggregate privilege UPDATE INFO from the HRPRIVS security class:

```
BEGIN
SYS.XS_SECURITY_CLASS.REMOVE_IMPLIED_PRIVILEGES('HRPRIVS','UPDATE_INFO','"DELETE"');
END;
```

```
BEGIN
   SYS.XS_SECURITY_CLASS.REMOVE_IMPLIED_PRIVILEGES('HRPRIVS','UPDATE_INFO');
END;
```

## 11.9.2.8 SET\_DESCRIPTION Procedure

The SET DESCRIPTION procedure sets a description string for the specified security class.

### **Syntax**

```
XS_SECURITY_CLASS.SET_DESCRIPTION (
   sec_class    IN VARCHAR2,
   description IN VARCHAR2);
```

#### **Parameters**

Parameter	Description
sec_class	The name of the security class for which the description is to be set.
	The name is schema qualified, for example, SCOTT.SC1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as SC1, and the current schema is SCOTT, it would resolve to SCOTT.SC1.
description	A description string for the specified security class.

### **Examples**

The following example sets a description string for the HRPRIVS security class:

```
BEGIN
   SYS.XS_SECURITY_CLASS.SET_DESCRIPTION(
    'HRPRIVS','Contains privileges required to manage HR data');
END;
```

## 11.9.2.9 DELETE\_SECURITY\_CLASS Procedure

The  ${\tt DELETE\_SECURITY\_CLASS}$  procedure deletes the specified security class.



Parameter	Description
sec_class	The name of the security class to be deleted.  The name is schema qualified, for example, SCOTT.SC1. When the schema part of the name is missing, the current session schema is assumed. For example, in this same example, if the name is specified as SC1, and the current schema is SCOTT, it would resolve to SCOTT.SC1.
delete_option	<ul> <li>The delete option to use. The following options are available:</li> <li>DEFAULT_OPTION:         The default option allows deleting a security class only if it is not referenced elsewhere. If there are other entities that reference the security class, then the security class cannot be deleted.     </li> <li>CASCADE OPTION:</li> </ul>
	The cascade option deletes the security class together with any references to it. The user deleting the security class must have privileges to delete these references as well.  ALLOW_INCONSISTENCIES_OPTION:  The allow inconsistencies option lets you delete the entity even if other entities have late binding references to it.

### **Examples**

The following example deletes the  ${\tt HRPRIVS}$  security class using the default option:

BEGIN

SYS.XS\_SECURITY\_CLASS.DELETE\_SECURITY\_CLASS('HRPRIVS',XS\_ADMIN\_UTIL.DEFAULT\_OPTIO N); END;



# Real Application Security HR Demo

This chapter describes the following topics:

- Overview of the Security HR Demo
- · What Each Script Does
- Setting Up the Security HR Demo Components
- Running the Security HR Demo Using Direct Logon
- Running the Security HR Demo Attached to a Real Application Security Session
- Running the Security HR Demo Cleanup Script
- Running the Security HR Demo in the Java Interface

# 12.1 Overview of the Security HR Demo

This Human Resources (HR) Demonstration shows how to use basic Real Application Security (RAS) features. This tutorial is an end-to-end use case scenario. PL/SQL scripts, a Java program source file, and log files can be found in Real Application Security HR Demo Files.

The HR demo secures the  $\mbox{HR.EMPLOYEE}$  table by applying a data security policy that has three realms:

- 1. An employee's own record realm. The ACL, EMP\_ACL controls this realm, which grants application role employee privileges to access the realm, including the SALARY column.
- 2. All the records in the IT department realm. The ACL, IT\_ACL controls this realm, which grants application role it\_engineer privileges to access the realm, but excluding the SALARY column.
- 3. All the employee records realm. The ACL, HR\_ACL controls this realm, which grants application role hr\_representative privileges to access the realm, including the SALARY column.

The HR Demo defines two application users to demonstrate the effects of the policy:

- DAUSTIN, an application user in the IT department. He has application roles employee and it\_engineer. So, he can access realm #1 and realm #2 mentioned previously; that is, he can view employee records in the IT department, but he cannot view the SALARY column, except for his own salary record.
- SMAVRIS, an application user in HR department. She has application roles employee and hr\_representative. So, she can access realm #1 and realm #3 mentioned previously; that is, she can view and update all the employee records.

The HR Demo scripts show:

 How to create Real Application Security objects: application user, application role, ACL, security class, and data security policy.



- How to use the data security policy to secure rows (using realm constraints) and columns (using a column constraint) of a table.
- How to directly logon to a database with application users (requiring a password), and how to create, attach, detach, and destroy a Real Application Security session.
- How to enable and disable an application role in a Real Application Security session.

# 12.2 What Each Script Does

The Security HR demo use case runs the following set of PL/SQL scripts to set up components and run the demo:

- hrdemo setup.sql: sets up the demo components by:
  - Creating a database user as the Real Application Security Administrator and then connecting as the Real Application Security Administrator to create the components.
  - Creating a database role, DB EMP.
  - Creating an IT application user, DAUSTIN.
  - Creating an HR application user, SMAVRIS.
  - Creating application roles: employee, it\_engineer, and hr\_representative, and then granting the database role DB EMP to each of these application roles.
  - Granting application roles employee and it\_engineer to application user DAUSTIN.
  - Granting application roles employee and hr\_representative to application user SMAVRIS.
  - Creating the VIEW\_SALARY privilege and creating the hr\_privileges security class in which to scope the privilege.
  - Creating three ACLs: EMP ACL, IT ACL, and HR\_ACL, in which:
    - \* EMP\_ACL grants the employee role the SELECT database privilege and VIEW\_SALARY application privilege to view an employee's own record, including the SALARY column.
    - \* IT\_ACL grants the it\_engineer role only the SELECT database privilege to view the employee records in the IT department, but it does not grant the VIEW SALARY privilege that is required for access to the SALARY column.
    - \* HR\_ACL grants the hr\_representative role SELECT, INSERT, UPDATE, and DELETE database privileges to view and update all employee's records, and granting the VIEW\_SALARY application privilege to view the SALARY column.
  - The HR demo secures the HR.EMPLOYEE table by creating and applying the data security policy, EMPLOYEES\_DS, that has the following three realms and column constraint:
    - \* An employee's own record realm. The ACL, EMP\_ACL controls this realm, which grants application role employee privileges to access the realm, including the SALARY column.



- \* All the records in the IT department realm. The ACL, IT\_ACL controls this realm, which grants application role it\_engineer privileges to access the realm, but excluding the SALARY column.
- \* All the employee records realm. The ACL, HR\_ACL controls this realm, which grants application role hr\_representative privileges to access the realm, including the SALARY column.
- \* A column constraint that protects the SALARY column by requiring the VIEW SALARY privilege to view its sensitive data.
- Validating all the objects that have been created to ensure that all configurations are correct.
- Setting up the mid-tier related configuration by creating a DISPATCHER user, setting
  the password for this user, and granting the roles, XSCONNECT and xsdispatcher to
  this DISPATCHER user.
- hrdemo.sql: runs the demo with direct logon, demonstrating:
  - That the IT application user, DAUSTIN, can view the records in the IT department, but can only view his own salary record, and cannot update his own record.
  - That the HR application user, SMAVRIS, can view all the records, including all salary rows in the SALARY column, and can update any record.
- hrdemo\_session.sql: runs the demo creating and attaching to a Real Application Security session, demonstrating:
  - Connecting as the Real Application Security Administrator and creating an application session for application user SMAVRIS and attaching to it.
  - Displaying the current user as SMAVRIS.
  - Displaying the enabled database roles as DB\_EMP and application roles as employee,
     hr representative, and XSPUBLIC for the current user SMAVRIS.
  - That SMAVRIS application user can view all records including all salary rows in the SALARY column.
  - Disabling the hr\_representative and thus limiting application user SMAVRIS to viewing only her own employee record.
  - Enabling the hr\_representative, thus allowing SMAVRIS application user to view all records, including all salary rows in the SALARY column again.
  - Detaching from the application session.
  - Destroying the application session.
- hrdemo\_clean.sql: performs a cleanup operation that removes: application roles, application users, ACLs, the data security policy, the database role, the Real Application Security administrative user, and the mid-tier dispatcher user.
- hrdemo.java: runs the HR Demo using the Java interface.

"Setting Up the Security HR Demo Components" describes in more detail how each of the Real Application Security components is created along with performing some other important tasks.



# 12.3 Setting Up the Security HR Demo Components

Before you can create Real Application Security components, you must first connect as SYS/ user as SYSDBA.

```
define passwd=&1
connect sys/&passwd as sysdba
```

This sections includes the following topics:

- About Creating Roles and Application Users
- About Creating the Security Class and ACLs
- About Creating the Data Security Policy
- About Validating the Real Application Security Objects
- About Setting Up the Mid-Tier Related Configuration

## 12.3.1 About Creating Roles and Application Users

Create the application roles EMPLOYEE, IT\_ENGINEER, and HR\_REPRESENTATIVE, and the database role DB\_EMP. The DB\_EMP role is used to grant the required object privileges to the two application users that are created, DAUSTIN and SMAVRIS. Finally, grant the HR user the policy administration privilege, ADMIN ANY SEC POLICY.

Connect as SYS/ user as SYSDBA.

```
define passwd=&1
connect sys/&passwd as sysdba
```

Create the application role **EMPLOYEE** for common employees.

```
exec sys.xs_principal.create_role(name => 'employee', enabled => true);
```

Create an application role IT ENGINEER for the IT department.

```
exec sys.xs_principal.create_role(name => 'it_engineer', enabled => true);
```

Create an application role HR REPRESENTATIVE for the HR department.

```
exec sys.xs principal.create role(name => 'hr representative', enabled => true);
```

Create the database role, DB EMP, for object privilege grants.

```
create role db_emp;
```

Grant the  $DB\_EMP$  database role to the three application roles, so they each have the required object privilege to access the table.

```
grant db_emp to employee;
grant db_emp to it_engineer;
grant db emp to hr representative;
```

Create the application users.

Create application user DAUSTIN (in the IT department) and grant this user application roles EMPLOYEE and IT ENGINEER.



```
exec sys.xs_principal.create_user(name => 'daustin', schema => 'hr');
exec sys.xs_principal.set_password('daustin', 'welcomel');
exec sys.xs_principal.grant_roles('daustin', 'XSCONNECT');
exec sys.xs_principal.grant_roles('daustin', 'employee');
exec sys.xs_principal.grant_roles('daustin', 'it engineer');
```

Create application user SMAVRIS (in the HR department) and grant this user application roles EMPLOYEE and HR REPRESENTATIVE.

```
exec sys.xs_principal.create_user(name => 'smavris', schema => 'hr');
exec sys.xs_principal.set_password('smavris', 'welcome1');
exec sys.xs_principal.grant_roles('smavris', 'XSCONNECT');
exec sys.xs_principal.grant_roles('smavris', 'employee');
exec sys.xs_principal.grant_roles('smavris', 'hr_representative');
```

Grant the HR user the policy administration privilege, ADMIN ANY SEC POLICY.

```
exec sys.xs admin util.grant system privilege('ADMIN ANY SEC POLICY', 'HR');
```

## 12.3.2 About Creating the Security Class and ACLs

First, grant the necessary table privileges to the DB EMP role.

Next, create a security class <code>HR\_PRIVILEGES</code> based on the predefined DML security class. <code>HR\_PRIVILEGES</code> has a new privilege <code>VIEW\_SALARY</code>, which controls access to the <code>SALARY</code> column. Finally, create the three ACLs, <code>EMP\_ACL</code>, <code>IT\_ACL</code>, and <code>HR\_ACL</code>.

Connect as the HR user.

```
connect hr/hr;
```

Grant the necessary object privileges to the  $DB\_EMP$  role. This role is used to grant the required object privileges to application users.

Create three ACLs, EMP\_ACL, IT\_ACL, and HR\_ACL to grant privileges for the data security policy to be defined later.



```
sec class => 'hr privileges');
  -- IT ACL: This ACL grants IT ENGINEER role the privilege to view the employee
             records in IT department, but it does not grant the VIEW SALARY
             privilege that is required for access to SALARY column.
  aces(1) := xs$ace type(privilege list => xs$name list('select'),
                        principal name => 'it engineer');
                                => 'it_acl',
 sys.xs_acl.create_acl(name
                   ace list => aces,
                    sec class => 'hr privileges');
  -- HR ACL: This ACL grants HR REPRESENTITIVE role the privileges to view and
update all
              employees' records including SALARY column.
 aces(1):= xs$ace_type(privilege_list => xs$name_list('select', 'insert',
                                         'update', 'delete', 'view_salary'),
                       principal name => 'hr representative');
                                => 'hr acl',
  sys.xs acl.create acl(name
                   ace list => aces,
                   sec class => 'hr privileges');
end;
```

## 12.3.3 About Creating the Data Security Policy

Create the data security policy for the EMPLOYEE table. The policy defines three realm constraints and a column constraint that protects the SALARY column.

```
declare
 realms xs$realm constraint list := xs$realm constraint list();
          xs$column constraint list := xs$column constraint list();
begin
  realms.extend(3);
  -- Realm #1: Only the employee's own record.
               The EMPLOYEE role can view the realm including SALARY column.
  realms(1) := xs$realm constraint type(
          => 'email = xs sys context(''xs$session'',''username'')',
   acl list => xs$name list('emp acl'));
  -- Realm #2: The records in the IT department.
             The IT ENGINEER role can view the realm excluding SALARY column.
  realms(2) := xs$realm constraint type(
   realm => 'department id = 60',
   acl list => xs$name list('it acl'));
  -- Realm #3: All the records.
             The HR REPRESENTATIVE role can view and update the realm
including SALARY column.
  realms(3) := xs$realm constraint type(
   realm => '1 = 1',
   acl list => xs$name list('hr acl'));
  -- Column constraint protects SALARY column by requiring VIEW SALARY
  -- privilege.
  cols.extend(1);
  cols(1) := xs$column constraint type(
    column list => xs$list('salary'),
```

```
privilege => 'view_salary');

sys.xs_data_security.create_policy(
   name => 'employees_ds',
   realm_constraint_list => realms,
   column_constraint_list => cols);
end;
//
```

Apply the data security policy to the EMPLOYEES table.

```
begin
  sys.xs_data_security.apply_object_policy(
    policy => 'employees_ds',
    schema => 'hr',
    object =>'employees');
end;
//
```

## 12.3.4 About Validating the Real Application Security Objects

After you create these Real Application Security objects, validate them to ensure they are all properly configured.

```
begin
   if (sys.xs_diag.validate_workspace()) then
      dbms_output.put_line('All configurations are correct.');
   else
      dbms_output.put_line('Some configurations are incorrect.');
   end if;
end;
/
-- XS$VALIDATION_TABLE contains validation errors if any.
-- Expect no rows selected.
select * from xs$validation table order by 1, 2, 3, 4;
```

## 12.3.5 About Setting Up the Mid-Tier Related Configuration

Set up the mid-tier configuration to be used later. This involves creating a session administrator, hr\_session, who only has Real Application Security administrative privileges (XS\_SESSION\_ADMIN and CREATE SESSION), but no data privileges. The session administrator is responsible for managing the Real Application Security session for each application user. In addition, it involves creating a DISPATCHER user and password and granting this user the XSCONNECT and XSDISPATCHER Real Application Security administrator privileges.

```
grant xs_session_admin, create session to hr_session identified by hr_session; grant create session to hr common identified by hr common;
```

Create a dispatcher user for the Java demo to set up a session for the application user.

```
exec sys.xs_principal.create_user(name=>'dispatcher', schema=>'HR');
exec sys.xs_principal.set_password('dispatcher', 'welcome1');
exec sys.xs_principal.grant_roles('dispatcher', 'XSCONNECT');
exec sys.xs_principal.grant_roles('dispatcher', 'xsdispatcher');
```



# 12.4 Running the Security HR Demo Using Direct Logon

To run the HR Demo, first connect as application user DAUSTIN, who has only the EMPLOYEE and IT ENGINEER application roles.

```
conn daustin/welcome1;
```

Customize how secured column values are to be displayed in SQL\*Plus using the default indicator asterisks (\*\*\*\*\*\*\*) in place of column values.

```
SET SECUREDCOL ON UNAUTH ******
```

Perform a query to show that application user DAUSTIN can view the records in the IT department, but can only view his own SALARY column.

```
select email, first_name, last_name, department_id, manager_id, salary
from employees order by email;
```

SQL> select email, first\_name, last\_name, department\_id, manager\_id, salary
2 from employees order by email;

EMAIL FIRST_NAME LAST_NAME DEPARTMENT_ID MANAG	GER_ID	SALARY
AHUNOLD Alexander Hunold 60	102	*****
BERNST Bruce Ernst 60	102	*****
DAUSTIN David Austin 60	103	4800
DLORENTZ Diana Lorentz 60	103	*****
VPATABAL Valli Pataballa 60	103	*****

<sup>5</sup> rows selected.

Set to the default display for how secured column values are to be displayed in SQL\*Plus by displaying null values in place of column values for application users without authorization, and in place of column values where the security level is unknown.

```
SET SECUREDCOL OFF
```

Perform an update operation to show that application user is not authorized to update the record.

```
update employees set manager_id = 102 where email = 'DAUSTIN';
SQL> update employees set manager_id = 102 where email = 'DAUSTIN';
0 rows updated.
```

Perform a query to show that the record is unchanged.

```
select email, first_name, last_name, department_id, manager_id, salary
from employees where email = 'DAUSTIN';
```

SQL> select email, first\_name, last\_name, department\_id, manager\_id, salary
2 from employees where email = 'DAUSTIN';

EMAIL	FIRST_NAME	LAST_NAME	DEPARTMENT_	ID MANAGER	L_ID	SALARY
DAUSTIN	David	Austin	6	60	103	4800



1 row selected.

Connect as application user SMAVRIS, who has both EMPLOYEE and  $HR_REPRESENTATIVE$  application roles.

conn smavris/welcome1;

Perform a query to show that application user SMAVRIS can view all the records including SALARY column.

```
select email, first_name, last_name, department_id, manager_id, salary
from employees where department_id = 60 or department_id = 40
order by department_id, email;
```

SQL> select email, first\_name, last\_name, department\_id, manager\_id, salary
2 from employees where department\_id = 60 or department\_id = 40

3 order by department id.	der by department id, email	ı;
---------------------------	-----------------------------	----

EMAIL	FIRST_NAME	LAST_NAME	DEPARTMENT_ID	MANAGER_ID	SALARY
SMAVRIS	Susan	Mavris	40	101	6500
AHUNOLD	Alexander	Hunold	60	102	9000
BERNST	Bruce	Ernst	60	103	6000
DAUSTIN	David	Austin	60	103	4800
DLORENTZ	Diana	Lorentz	60	103	4200
VPATABAL	Valli	Pataballa	60	103	4800

6 rows selected.

Perform a guery to show that application user SMAVRIS can access all the records.

```
select count(*) from employees;

SQL> select count(*) from employees;

COUNT(*)
-----
107
```

1 row selected.

Perform an update of the record to show that application user SMAVRIS can update the record.

```
update employees set manager_id = 102 where email = 'DAUSTIN';

SQL> update employees set manager_id = 102 where email = 'DAUSTIN';

1 row updated.
```

## Perform a query to show that the record is changed.

```
select email, first_name, last_name, department_id, manager_id, salary
from employees where email = 'DAUSTIN';
```

SQL> select email, first\_name, last\_name, department\_id, manager\_id, salary
2 from employees where email = 'DAUSTIN';

EMAIL	FIRST_NAME	LAST_NAME	DEPARTMENT_ID	MANAGER_ID	SALARY
DAUSTIN	David	Austin	60	102	4800

```
1 row selected.
```

Update the record to change it back to its original state.

```
update employees set manager_id = 103 where email = 'DAUSTIN';

SQL> update employees set manager_id = 103 where email = 'DAUSTIN';

1 row updated.
```

# 12.5 Running the Security HR Demo Attached to a Real Application Security Session

To run the demo attached to a Real Application Security session, the Real Application Security administrator must first create the session for an application user and attach to it. In the process, create a variable to remember the session ID.

```
connect hr_session/hr_session;
var gsessionid varchar2(32);

declare
   sessionid raw(16);
begin
   sys.dbms_xs_sessions.create_session('SMAVRIS', sessionid);
   :gsessionid := rawtohex(sessionid);
   sys.dbms_xs_sessions.attach_session(sessionid, null);
end;
//
```

#### Display the current user.

```
select xs_sys_context('xs$session','username') from dual;

SQL> select xs_sys_context('xs$session','username') from dual;

XS_SYS_CONTEXT('XS$SESSION','USERNAME')

SMAVRIS

1 row selected.
```

### Display the enabled database and application roles for the current application user.



# Perform a query to show that application user SMAVRIS can view all the records including SALARY column.

```
select email, first_name, last_name, department_id, manager_id, salary
from employees where department_id = 60 or department_id = 40
order by department_id, email;
```

SQL> select email, first\_name, last\_name, department\_id, manager\_id, salary 2 from employees where department id = 60 or department id = 40

3 order by department\_id, email;

EMAIL	FIRST_NAME	LAST_NAME	DEPARTMENT_ID	MANAGER_ID	SALARY
SMAVRIS	Susan	Mavris	40	101	6500
AHUNOLD	Alexander	Hunold	60	102	9000
BERNST	Bruce	Ernst	60	103	6000
DAUSTIN	David	Austin	60	103	4800
DLORENTZ	Diana	Lorentz	60	103	4200
VPATABAL	Valli	Pataballa	60	103	4800

6 rows selected.

#### Perform a guery to show that application user SMAVRIS can access all the records.

```
select count(*) from employees;

SQL> select count(*) from employees;

COUNT(*)
-----
107
```

1 row selected.

# Disable the HR\_REPRESENTATIVE role. This will limit application user SMAVRIS to only be able to see her own record.

```
exec dbms xs sessions.disable role('hr representative');
```

## Perform a query

```
select email, first_name, last_name, department_id, manager_id, salary
from employees where department_id = 60 or department_id = 40
order by department id, email;
```

```
SQL> select email, first_name, last_name, department_id, manager_id, salary 2 from employees where department_id = 60 or department_id = 40
```

3 order by department\_id, email;

EMAIL	FIRST_NAME	LAST_NAME	DEPARTMENT_I	D MANAGER_ID	SALARY
SMAVRIS	Susan	Mavris	4	101	6500

1 row selected.

# Enable the $\mbox{HR}_{\mbox{\scriptsize REPRESENTATIVE}}$ role so the application user can view all the records including salary column.

exec dbms xs sessions.enable role('hr representative');



# Perform a query to show that application user can view all the records including SALARY column.

EMAIL	FIRST_NAME	LAST_NAME	DEPARTMENT_ID	MANAGER_ID	SALARY
SMAVRIS	Susan	Mavris	40	101	6500
AHUNOLD	Alexander	Hunold	60	102	9000
BERNST	Bruce	Ernst	60	103	6000
DAUSTIN	David	Austin	60	103	4800
DLORENTZ	Diana	Lorentz	60	103	4200
VPATABAL	Valli	Pataballa	60	103	4800

<sup>6</sup> rows selected.

### Perform a query to show that application user SMAVRIS can access all the records.

```
select count(*) from employees;

SQL> select count(*) from employees;

COUNT(*)
-----
107
```

1 row selected.

#### Detach and destroy the application session.

```
declare
   sessionid raw(16);
begin
   sessionid := hextoraw(:gsessionid);
   sys.dbms_xs_sessions.detach_session;
   sys.dbms_xs_sessions.destroy_session(sessionid);
end;
//
```

# 12.6 Running the Security HR Demo Cleanup Script

After running the HR demo, you can run the clean up script to remove all of the Real Application Security components.

To start, connect as the Real Application Security Administrator and then begin removing components.

```
define passwd=&1
connect hr/hr;
```

Remove the data security policy from the EMPLOYEES table.



### Delete the security class and the ACLs.

```
exec sys.xs_security_class.delete_security_class('hrprivs',
xs_admin_util.cascade_option);
exec sys.xs_acl.delete_acl('emp_acl', xs_admin_util.cascade_option);
exec sys.xs_acl.delete_acl('it_acl', xs_admin_util.cascade_option);
exec sys.xs acl.delete acl('hr acl', xs_admin_util.cascade_option);
```

#### Delete the data security policy.

```
exec sys.xs data security.delete policy('employees ds', xs admin util.cascade option);
```

#### Connect as SYS/ user as SYSDBA.

```
connect sys/&passwd as sysdba
```

#### Delete the application roles and application users.

```
exec sys.xs_principal.delete_principal('employee', xs_admin_util.cascade_option);
exec sys.xs_principal.delete_principal('hr_representative',
    xs_admin_util.cascade_option);
exec sys.xs_principal.delete_principal('it_engineer', xs_admin_util.cascade_option);
exec sys.xs_principal.delete_principal('smavris', xs_admin_util.cascade_option);
exec sys.xs_principal.delete_principal('daustin', xs_admin_util.cascade_option);
```

#### Delete the database role.

```
drop role db_emp;
```

#### Delete the Real Application Security session administrator.

```
drop user hr session;
```

#### Delete the common user used to connect to the database.

```
drop user hr_common;
```

#### Delete the DISPATCHER user used by the mid-tier.

```
exec sys.xs principal.delete principal('dispatcher', xs admin util.cascade option);
```

# 12.7 Running the Security HR Demo in the Java Interface

See the Output section in "Human Resources Administration Use Case: Implementation in Java" for a description of the two queries that are returned from running the Security HR Demo in the Java interface.

A

# Predefined Objects in Real Application Security

This appendix describes the following predefined objects in Real Application Security:

- Users
- Roles
- Namespaces
- Security Classes
- ACLs

## A.1 Users

XSGUEST - A system-defined Real Application Security user typically reserved for anonymous access.

## A.2 Roles

Real Application Security provides predefined application roles for regular application roles, dynamic application roles, and database roles.

This section includes the following topics:

- Regular Application Roles
- Dynamic Application Roles
- Database Roles

## A.2.1 Regular Application Roles

Real Application Security provides the following predefined regular application roles:

- XSPUBLIC This application role is similar to the PUBLIC role in the database. It is granted
  to all Real Application Security application users.
- XSBYPASS A role used to bypass the restrictions imposed by a system constraining ACL.
- XSPROVISIONER A role used to grant PROVISION and CALLBACK privileges.
- XSSESSIONADMIN A role used for session administration.
- XSNAMESPACEADMIN A role used for namespace attribute administration.
- XSCACHEADMIN A role used for middle tier cache administration.
- XSDISPATCHER A role used for session administration, namespace administration, and middle tier cache administration by a dispatcher.



XSCONNECT — A role used to control whether a Real Application Security
application user with a password can connect to the database or not.

# A.2.2 Dynamic Application Roles

Real Application Security provides the following predefined dynamic application roles:

DBMS AUTH

This application role depends on the authentication state of the application user. It is enabled whenever the application user is authenticated in the Real Application Security system as a direct-logon application user using any of the database authentication methods.

• EXTERNAL\_DBMS\_AUTH

This application role depends on the authentication state of the external application user. It is enabled whenever the external application user is authenticated in the Real Application Security system as an external direct-logon application user using any of the database authentication methods.

DBMS\_PASSWD

This application role depends on the authentication state of the application user. It is enabled whenever the application user is authenticated in the Real Application Security system as a direct-logon application user using a password authentication method.

MIDTIER AUTH

This application role depends on the authentication state of the application user. It is enabled whenever the application user is authenticated in the Real Application Security system through the middle tier. The middle tier explicitly passes this application role to the database indicating that the application user has been authenticated by the middle tier.

XSAUTHENTICATED

This application role depends on the authentication state of the application user. It is enabled whenever the application user is authenticated in the Real Application Security system (either directly or through the middle tier).

XSSWITCH

This application role depends on the session state of the application user. It is enabled whenever the Real Application Security session for an application user is created as a result of a switch\_user operation, that is, if the proxy user in the original Real Application Security session is switched to an application user.

## A.2.3 Database Roles

Real Application Security provides the following database roles.

- PROVISIONER A database role that has the PROVISION and CALLBACK privileges.
- XS\_SESSION\_ADMIN A database role that has the ADMINISTER\_SESSION privilege.
- XS\_NAMESPACE\_ADMIN A database role that has the ADMIN\_ANY\_NAMESPACE privilege.



XS CACHE ADMIN - A database role that can be used for middle tier cache administration.

# A.3 Namespaces

Real Application Security provides the following predefined namespaces:

- XS\$GLOBAL\_VAR Contains the following NLS Attributes: NLS\_LANGUAGE, NLS\_TERRITORY, NLS\_SORT, NLS\_DATE\_LANGUAGE, NLS\_DATE\_FORMAT, NLS\_CURRENCY, NLS\_NUMERIC\_CHARACTERS, NLS\_ISO\_CURRENCY, NLS\_CALENDAR, NLS\_TIME\_FORMAT, NLS\_TIMESTAMP\_FORMAT, NLS\_TIME\_TZ\_FORMAT, NLS\_TIMESTAMP\_TZ\_FORMAT, NLS\_DUAL CURRENCY, NLS\_COMP, NLS\_LENGTH\_SEMANTICS, and NLS\_NCHAR\_CONV\_EXCP.
  - The XS\$GLOBAL\_VAR namespace can be loaded in to a Real Application Security session without requiring any privileges.
- XS\$SESSION Contains the following attributes: CREATED\_BY, CREATE\_TIME, COOKIE, CURRENT\_XS\_USER, CURRENT\_XS\_USER\_GUID, INACTIVITY\_TIMEOUT, LAST\_ACCESS\_TIME, LAST\_AUTHENTICATION\_TIME, LAST\_UPDATED\_BY, PROXY\_GUID, SESSION\_ID, SESSION\_SIZE, SESSION XS USER, SESSION XS USER GUID, USERNAME, and USER ID.

# A.4 Security Classes

Real Application Security provides the following predefined security classes and application privileges:

- DML DML Privileges security class. If an ACL does not specify its security class, DML is
  the default security class for the ACL. See "DML Security Class" for more information.
  Contains the following common application privileges for object manipulation.
  - SELECT Privilege to read an object.
  - INSERT Privilege to insert an object.
  - UPDATE Privilege to update an object.
  - DELETE Privilege to delete an object.
- SYSTEM System security class. Contains the following application privileges:
  - PROVISION Privilege for updating principal documents from FIDM. The PROVISION privilege is also extended for creating, deleting, and modifying Real Application Security principals (users or roles) beginning in Release 12.2. This Real Application Security system privilege is intended to replace the traditional use of database create user, alter user privileges, and so forth to create and alter Real Application Security application users and roles.
  - CALLBACK Privilege to register and update global callbacks.
  - ADMIN ANY SEC POLICY Privilege for any administrative operation.
  - ADMIN SEC POLICY Privilege for administering objects in its own schema.
  - ADMIN NAMESPACE Privilege for administering any namespace.
- SESSION SC Session security class. Contains the following application privileges:
  - CREATE SESSION Privilege to create a Real Application Security user session.
  - TERMINATE SESSION Privilege to terminate a Real Application Security user session.
  - ATTACH SESSION Privilege to attach to a Real Application Security user session.



- MODIFY\_SESSION Privilege to modify contents of a Real Application Security user session.
- ASSIGN\_USER Privilege to assign user to an anonymous Real Application Security user session.
- ADMINISTER\_SESSION Privilege for Real Application Security user session administration, aggregate of CREATE\_SESSION, TERMINATE\_SESSION, ATTACH SESSION, MODIFY SESSION, and SET DYNAMIC ROLES.
- SET\_DYNAMIC\_ROLES Privilege to protect Real Application Security
  enablement and disablement of a dynamic role as part of the attach session
  and assign user operations.
- NSTEMPLATE\_SC Namespace template security class. Contains the following application privileges:
  - MODIFY NAMESPACE Privilege to modify session namespace.
  - MODIFY ATTRIBUTE Privilege to modify session namespace attribute.
  - ADMIN\_NAMESPACE Privilege for namespace administration, aggregate of MODIFY NAMESPACE and MODIFY ATTRIBUTE.

## A.5 ACI s

Real Application Security provides the following predefined ACLs:

- SYSTEMACL ACL for granting SYSTEM security class privileges.
  - Grants PROVISION and CALLBACK privileges to PROVISIONER database role and XSPROVISIONER Real Application Security role.
  - Grants ADMIN ANY SEC POLICY privilege to DBA database role.
  - Grants ADMIN SEC POLICY privilege to RESOURCE and XS RESOURCE database roles.
  - Grants ADMIN\_ANY\_NAMESPACE privilege to DBA and XS\_NAMESPACE\_ADMIN database roles and XSNAMESPACEADMIN and MIDTIER AUTH Real Application Security roles.
- SESSIONACL ACL for granting SESSION SC security class privileges.
  - Grants ADMINISTER\_SESSION privilege to XS\_SESSION\_ADMIN database role and XSSESSIONADMIN Real Application Security role.
- NS\_UNRESTRICTED\_ACL ACL to grant ADMIN\_NAMESPACE privilege to PUBLIC database role and XSPUBLIC Real Application Security role.



B

# Configuring OCI and JDBC Applications for Column Authorization

### This appendix contains:

- About Using OCI to Retrieve Column Authorization Indicators
- About Using JDBC to Retrieve Column Authorization Indicators

# B.1 About Using OCI to Retrieve Column Authorization Indicators

Oracle Call Interface (OCI) applications can access database tables that have data security policies enabled and then test columns for authorization indicators.

- If the column is determined to be unauthorized to the user, a null column value is returned to the user with indicator "unauthorized".
- If the column authorization cannot be determined, the evaluated column (or column expression) value will be returned to the user along with the indicator "unknown." If any of the underlying table columns involved in the top column expression evaluation is unauthorized, the authorization indicator can be "unknown" and a null value will be used as the underlying column value for expression valuation.
- If the column is determined as authorized to the user, the evaluated column value and indicator will be returned to the user without authorization indicator.

The OCI return code is to communicate column authorization information to the user. To obtain the authorization information for a column, you must provide a return-code buffer when the column buffer is bound or defined. After the column data is returned to the user buffer, you can check the return code associated with the column for authorization information. The column authorization indicator is applicable to define variables or out-bind variables defined by the application. The return code buffer does not have to be provided if the application is not retrieving the column authorization indicator.

This section describes the following topics:

- Example of Obtaining the Return Code
- About Using the Return Code and Indicator with Authorization Indicator
- · About the Warning for Unknown Authorization Indicator
- Using OCI Describe for Column Security

# B.1.1 Example of Obtaining the Return Code

The following return codes are used to find the column authorizations:

- ORA-24530: column value is unauthorized to the user
- ORA-24531: column value authorization is unknown



#### ORA-24536: column authorization unknown

If the unknown value authorization indicator (ORA-24531) is returned for any column, the OCI function status will be <code>OCI\_SUCCESS\_WITHINFO</code> and the error ORA-24536 will be returned in the error handle as warning. To suppress the warning, the application can set attribute, <code>OCI\_ATTR\_NO\_COLUMN\_AUTH\_WARNING</code> to <code>TRUE</code> in the statement handle before fetching:

The default boolean value of OCI ATTR NO COLUMN AUTH WARNING is FALSE.

Example B-1 shows OCI code that retrieves the return codes.

### Example B-1 Retrieving Return Codes from OCI for a Column Authorization

# B.1.2 About Using the Return Code and Indicator with Authorization Indicator

To access tables with column security, you should access the return code at least when the column is bound or defined. If the return code is not accessed, the application needs to know if the column value is authorized with other means so that it can correctly interpret the indicator and the value.

You should also provide the indicator for the bind or define if column security is enabled. If the indicator is not provided and the column value is not authorized or unknown, Oracle Database returns error ORA-1405.

If column value authorization is unknown, the authorization indicator (for ORA-24531) will take precedence over the regular return codes that may otherwise be returned to the user. For example, column null fetch (ORA-1405) and column truncation (ORA-1406) may occur at the same time when a non-null column value is returned along with unknown authorization indicator. In that case, the application gets ORA-24531 as the return code for this column, instead of getting ORA-1405 or ORA-1406. Hence the application should not rely on column return code ORA-1405 or ORA-1406 to find the exact column that is null fetched or truncated.

Table B-1 and Table B-2 summarizes the behavior of the authorization indicator, return code, indicator, and return status.



## B.1.3 About the Warning for Unknown Authorization Indicator

If the unknown authorization indicator (ORA-24531) is returned for any column, an OCI warning is returned to the application, that is, the OCI function status will be <code>OCI\_SUCCESS\_WITH\_INFO</code>, instead of <code>OCI\_SUCCESS</code>. At the same time, ORA-24536 will be set in the error handle returned to the application. You must check this warning, examine the SQL being executed, and take appropriate action. The error ORA-24536 takes precedence over the error that is returned when the column is authorized or column security is not enabled.

If a column value is unauthorized or authorized, the OCI function status code will not be changed.

By default the column authorization warning is turned on for unknown authorizations. The application should be designed to handle the error. If the application is prepared for column security and wants to ignore any unknown authorization indicator, the OCI warning can be turned off by setting the OCI attribute, OCI\_ATTR\_NO\_COLUMN\_AUTH\_WARNING to TRUE in the OCI statement handle before the column value is fetched.

Table B-1 describes the default authorization behavior for OCI return indicators.

Table B-1 Authorization Indicator Behavior (By Default)

Column Authorization	Column Value	IND Provided RC Provided	IND Not Provided RC Provided	IND Provided RC Not Provided	IND Not Provided RC Not Provided
Unauthorized	Any	OCI_SUCCESS	OCI_SUCCESS	OCI_SUCCESS	OCI_SUCCESS
		Error = 0	Error = 1405	Error = 0	Error = 1405
		IND = -1	IND = N/A	IND = -1	IND =-N/A
		RC = 24530	RC = 24530	RC = N/A	RC = N/A
Unknown	Null	SUCCESS_WITH_ INFO	SUCCESS_WITH_I NFO	SUCCESS_WITH_ INFO	SUCCESS_WITH _INFO
		Error = 24536 (0) IND = -1	Error = 24536 (1405)	Error = 24536 (0) IND = -1	Error = 24536 (1405)
		RC = 24531 (0)	IND = N/A	RC = N/A	IND = N/A
		(-,	RC = 24531 (1405)		RC = N/A
Unknown	Not Null and Not	SUCCESS_WITH_ INFO	SUCCESS_WITH_I NFO	SUCCESS_WITH_ INFO	SUCCESS_WITH _INFO
	Truncated	Error = 24536 (0)	Error = 24536 (0)	Error = 24536 (0)	Error = 24536 (0)
		IND = 0	IND = N/A	IND = 0	IND = N/A
		RC = 24531 (0)	RC = 24531 (0)	RC = N/A	RC = N/A
Unknown	Not Null and	SUCCESS_WITH_ INFO	SUCCESS_WITH_I NFO	SUCCESS_WITH_ INFO	SUCCESS_WITH _INFO
	Truncated	Error = 24536 (24345)	Error = 24536 (24345)	Error = 24536 (1406)	Error = 24536 (1406)
		IND = data_len	IND = N/A	IND = data_len	IND = N/A
		RC = 24531 (1406)	RC = 24531 (1406)	RC = N/A	RC = N/A



## See Also:

Oracle Call Interface Programmer's Guide Table 2-4 shows the default fetch behavior without column security

Table B-2 describes the default behavior when the <code>OCI\_ATTR\_NO\_AUTH\_WARNING</code> parameter is set to <code>TRUE</code>.

Table B-2 Authorization Indicator Behavior (By Default) - OCI\_ATTR\_NO\_AUTH\_WARNING=TRUE

Column Authorization	Column Value	IND Provided RC Provided	IND Not Provided RC Provided	IND Provided RC Not Provided	IND Not Provided RC Not Provided
Unknown	Null	Error = 0	Error = 1405	Error = 0	Error = 1405
		IND = -1	IND = N/A	IND = -1	IND = N/A
		RC = 24531 (0)	RC = 24531 (1405)	RC = N/A)	RC = N/A
Unknown	Not Null and	Error = 0	Error = 0	Error = 0	Error = 0
	Not Truncated	IND = 0	IND = N/A	IND = 0	IND = N/A
		RC = 24531 (0)	RC = 24531 (0)	RC = N/A	RC = N/A
Unknown	Not Null and Truncated	SUCCESS_WITH_I NFO	SUCCESS_WITH_IN FO	Error = 1406 IND =	Error = 1406 IND = N/A
		Error = 24345	Error = 24345	data_len	RC = N/A
		IND = data_len	IND = N/A	RC = N/A)	
		RC = 24531 (1406)	RC = 24531 (1406)		

## B.1.4 Using OCI Describe for Column Security

The <code>OCIDescribeAny()</code> function enables an explicit describe of schema objects. Applications sometimes need to know if a column is protected by a column constraint before fetching data. You can use this information to guide the application to process the data and indicators. This is especially useful to applications that handle dynamic SQL. The attribute <code>OCI\_ATTR\_XDS\_POLICY\_STATUS</code> for the OCI parameter handle is of data type <code>ub4</code> and has the following possible values:

- OCI XDS POLICY NONE: No XDS policy for the column or the policy is not enabled
- OCI\_XDS\_POLICY\_ENABLED: policy is enabled for the column
- OCI XDS POLICY UNKNOWN: policy unknown

If the column status is  $OCI\_XDS\_POLICY\_NONE$ , then the column values will always be "authorized." If the column status is  $OCI\_XDS\_POLICY\_ENABLED$ , then the column values will be either "authorized" or "unauthorized." If the column status is  $OCI\_XDS\_POLICY\_UNKNOWN$ , the column value authorization will always be "unknown."

Example B-2 shows how to use the OCIDescribeAny() function to perform an explicit describe on a set of schema objects.



See Also:

Oracle Call Interface Programmer's Guide

### Example B-2 Using the OCIDescribeAny Function to Enable an Explicit Describe

```
void desc explicit()
 const char *table = "col sec tab";
 ub4 pos;
 ub2 numcol;
 OCIParam *paramh;
 OCIParam *collst;
 OCIParam *col;
 ub4 colnamelen, colseclen;
 ub1 colname[20];
 ub1 *colnm;
 ub4 colsec;
  ub4 tablen = strlen((char *)table);
 checkerr(errhp, OCIDescribeAny(svchp, errhp, (dvoid *)table, tablen,
                               OCI OTYPE NAME, 0, OCI PTYPE TABLE, deschp));
  checkerr(errhp, OCIAttrGet(deschp, OCI_HTYPE_DESCRIBE, &paramh, 0,
                           OCI_ATTR_PARAM, errhp));
  checkerr(errhp, OCIAttrGet(paramh, OCI DTYPE PARAM, &numcol, 0,
                           OCI ATTR NUM COLS, errhp));
  checkerr(errhp, OCIAttrGet(paramh, OCI DTYPE PARAM, &collst, 0,
                           OCI ATTR LIST COLUMNS, errhp));
  printf("Number of columns = %d\n\n", numcol);
 printf(" -----\n\n");
  for (pos = 1; (ub4) pos <= numcol; pos++)</pre>
   checkerr(errhp, OCIParamGet (collst, OCI DTYPE PARAM, errhp,
                               (dvoid **)&col, pos));
   checkerr(errhp, OCIAttrGet ((dvoid *)col, (ub4) OCI DTYPE PARAM,
                              (dvoid **) &colnm, (ub4 *) &colnamelen,
                              (ub4) OCI ATTR NAME, errhp));
   memset (colname, ' ', 20);
   strncpy((char *)colname, (char *)colnm, colnamelen);
   colname[10] = '\0';
   checkerr(errhp, OCIAttrGet ((dvoid *)col, (ub4) OCI DTYPE PARAM,
                              (dvoid **) &colsec, (ub4 *) &colseclen,
                              (ub4) OCI_ATTR_XDS_POLICY_STATUS, errhp));
   printf(" %d
                         %S
                              %s\n", pos, colname,
                         ((colsec == OCI XDS POLICY ENABLED) ? "ENABLED" :
```



# B.2 About Using JDBC to Retrieve Column Authorization Indicators

JDBC applications can access database tables that have data security policies enabled, and test columns for authorization indicators. You can use the JDBC APIs described in this section to check the security attributes and user authorization for a table column.

This section contains:

- About Checking Security Attributes for a Table Column
- About Checking User Authorization for a Table Column
- Example of Checking Security Attributes and User Authorization

## B.2.1 About Checking Security Attributes for a Table Column

The <code>getSecurityAttribute</code> method of the <code>oracle.jdbc.OracleResultSetMetaData</code> interface enables you to check the data security policy attribute for a column. The security attribute has the following definition:

```
public static enum SecurityAttribute
{
   NONE,
   ENABLED,
   UNKNOWN;
}
```

SecurityAttribute can have the following values:

- NONE implies that no column data security policy is enabled for the column. This
  means that the column either does not have a policy applied to it, or the policy is
  not enabled.
- ENABLED implies that column data security policy is enabled for the column.
- UNKNOWN implies that the column data security policy for the column is unknown.
   This could happen, for example, if the column is a union of two columns but only one of the columns has data security attributes.

The getSecurityAttribute method has the following signature:

```
public SecurityAttribute getSecurityAttribute(int indexOfColumnInResultSet)
throws SQLException;
```

The getSecurityAttribute method returns the SecurityAttribute value for the column.





Example B-3 for an example of using the getSecurityAttribute method

## B.2.2 About Checking User Authorization for a Table Column

The <code>getAuthorizationIndicator</code> method of the <code>oracle.jdbc.OracleResultSet</code> interface enables you to check the <code>AuthorizationIndicator</code> attribute for a column. The <code>AuthorizationIndicator</code> attribute has the following definition:

```
public static enum AuthorizationIndicator
{
   NONE,
   UNAUTHORIZED,
   UNKNOWN;
}
```

AuthorizationIndicator can have the following values:

- NONE implies that access to column data is authorized. The user might have explicit authorization or the column could be lacking security attributes.
- UNAUTHORIZED implies that access to column data is not authorized.

When the column value is retrieved, the authorization indicator is evaluated based on the enabled column constraint policy for the column. If the user is not authorized to access the column value, a NULL value is returned to the application along with the authentication indicator, AuthorizationIndicator.UNAUTHORIZED.

If there is a column expression involving the unauthorized base column, the evaluated value is returned to the application along with the <code>AuthorizationIndicator.UNAUTHORIZED</code> indicator. The application should examine the authorization indicator before interpreting the returned data.

UNKNOWN implies that the authorization indicator cannot be determined.

Sometimes, the server fails to determine the authorization indicator for a SELECT item due to functionality limitations or performance constrains. This can happen if the query involves a column expression, for example, and the server is unable to compute whether the top operator is supposed to be authorized. In such a scenario, the server returns the authorization indicator, AuthorizationIndicator.UNKNOWN to the application. The returned value can be NULL or not NULL depending on how the column expression operates on the underlying column value.

If the application sees an <code>UNKNOWN</code> authorization indictor, it should determine whether or not the returned value should be accessed. If the query and its column expressions are designed to handle unauthorized NULL values from the underlying columns, then the application can use the returned value. Otherwise the application may have to take appropriate actions for the returned value.

The getAuthorizationIndicator method has the following forms:

```
/**
  * Accepts the column index number as an argument and retrieves the corresponding
column security AuthorizationIndicator value
  */
public AuthorizationIndicator getAuthorizationIndicator(int columnIndex) throws
```



#### SQLException;

/\*\*

 $^{\star}$  Accepts the column name as a string and retrieves the column security AuthorizationIndicator value

\* /

 $\verb|public AuthorizationIndicator getAuthorizationIndicator(String columnName)| throws SQLException;\\$ 



- The preceding methods throw a SQLException if the index specified in the argument is invalid.
- If a column is masked, the JDBC user sees it as a NULL value. An
  exception is not thrown for this.



**Example B-3 for an example of using the** getAuthorizationIndicator method

# B.2.3 Example of Checking Security Attributes and User Authorization

Example B-3 illustrates the use of the <code>getSecurityAttribute</code> and <code>getAuthorization</code> methods to check security attributes and user authorization. The program uses the sample <code>EMP</code> table to illustrate the procedure.

The EMP table is configured as follows:

Column No.	Column Title	Security Attribute
1	EMPNO	No security attribute
2	ENAME	Active security
3	JOB	No security attribute
4	MGR	Active security
5	HIREDATE	Unknown security attribute
6	SAL	Active security
7	COMM	No security attribute
6	DEPTNO	Active security

The program performs the following actions:

- 1. Selects rows from the EMP table
- 2. Uses the getSecurityAttribute method to extract the security setting for each column in the result set. It prints these as column headings



3. Uses the getAuthorizationIndicator method to check the user authorization for returned column values. The program prints these values and formats them as follows:

An unauthorized value that is returned as NULL is represented by four asterisk characters (\*\*\*\*).

### **Example B-3** Check Security Attributes and User Authorization

```
PreparedStatement pstmt = conn.prepareStatement("SELECT * FROM EMP");
  ResultSet rs = pstmt.executeQuery();
  OracleResultSetMetaData metaData =
  (OracleResultSetMetaData) rs.getMetaData();
  int nbOfColumns = metaData.getColumnCount();
  OracleResultSetMetaData.SecurityAttribute[] columnSecurity
   = new OracleResultSetMetaData.SecurityAttribute[nbOfColumns];
  // display which columns are protected:
  for(int i=0;i<nbOfColumns;i++)</pre>
   columnSecurity[i] = metaData.getSecurityAttribute(i+1);
   System.out.print(columnSecurity[i]);
   System.out.print("\t");
  System.out.println();
  System.out.println("-----");
  while(rs.next())
    for(int colIndex=0;colIndex<nbOfColumns;colIndex++)</pre>
     OracleResultSet.AuthorizationIndicator visibility
       = ((OracleResultSet)rs).getAuthorizationIndicator(colIndex+1);
     if(visibility == OracleResultSet.AuthorizationIndicator.UNAUTHORIZED)
       System.out.print("****");
       System.out.print(rs.getString(colIndex+1));
     System.out.print("\t");
   System.out.println("");
  rs.close();
 pstmt.close();
```

#### The program generates the following output:

7369 SMITH CLERK 7902 1980-12-17 **** null 20 7499 ALLEN SALESMAN 7698 1981-02-20 **** 300 30 7521 WARD SALESMAN 7698 1981-02-22 **** 500 30 7566 JONES MANAGER 7839 1981-04-02 **** null 20 7654 MARTIN SALESMAN 7698 1981-09-28 **** 1400 30 7698 BLAKE MANAGER 7839 1981-05-01 **** null 30 7782 CLARK MANAGER 7839 1981-06-09 **** null 10 7788 SCOTT ANALYST 7566 1987-04-19 **** null 20 7839 KING PRESIDENT null 1981-11-17 **** null 10 7844 TURNER SALESMAN 7698 1981-09-08 **** null 20 7876 ADAMS CLERK 7788 1987-05-23 **** null 20 7900 JAMES CLERK 7698 1981-12-03 **** null 30 7902 FORD ANALYST 7566 1981-12-03 **** null 20 7934 MILLER CLERK 7782 1982-01-23 **** null 10	NONE	ENABLED	NONE	ENABLED	UNKNOWN	ENABLED	NONE	ENABLED
7521 WARD SALESMAN 7698 1981-02-22 **** 500 30 7566 JONES MANAGER 7839 1981-04-02 **** null 20 7654 MARTIN SALESMAN 7698 1981-09-28 **** 1400 30 7698 BLAKE MANAGER 7839 1981-05-01 **** null 30 7782 CLARK MANAGER 7839 1981-06-09 **** null 10 7788 SCOTT ANALYST 7566 1987-04-19 **** null 20 7839 KING PRESIDENT null 1981-11-17 **** null 10 7844 TURNER SALESMAN 7698 1981-09-08 **** 0 30 7876 ADAMS CLERK 7788 1987-05-23 **** null 20 7900 JAMES CLERK 7698 1981-12-03 **** null 30 7902 FORD ANALYST 7566 1981-12-03 **** null 20	7369	SMITH	CLERK	7902	1980-12-17	***	null	20
7566 JONES MANAGER 7839 1981-04-02 **** null 20 7654 MARTIN SALESMAN 7698 1981-09-28 **** 1400 30 7698 BLAKE MANAGER 7839 1981-05-01 **** null 30 7782 CLARK MANAGER 7839 1981-06-09 **** null 10 7788 SCOTT ANALYST 7566 1987-04-19 **** null 20 7839 KING PRESIDENT null 1981-11-17 **** null 10 7844 TURNER SALESMAN 7698 1981-09-08 **** 0 30 7876 ADAMS CLERK 7788 1987-05-23 **** null 20 7900 JAMES CLERK 7698 1981-12-03 **** null 30 7902 FORD ANALYST 7566 1981-12-03 **** null 20	7499	ALLEN	SALESMAN	7698	1981-02-20	****	300	30
7654 MARTIN SALESMAN 7698 1981-09-28 **** 1400 30 7698 BLAKE MANAGER 7839 1981-05-01 **** null 30 7782 CLARK MANAGER 7839 1981-06-09 **** null 10 7788 SCOTT ANALYST 7566 1987-04-19 **** null 20 7839 KING PRESIDENT null 1981-11-17 **** null 10 7844 TURNER SALESMAN 7698 1981-09-08 **** 0 30 7876 ADAMS CLERK 7788 1987-05-23 **** null 20 7900 JAMES CLERK 7698 1981-12-03 **** null 30 7902 FORD ANALYST 7566 1981-12-03 **** null 20	7521	WARD	SALESMAN	7698	1981-02-22	****	500	30
7698 BLAKE MANAGER 7839 1981-05-01 **** null 30 7782 CLARK MANAGER 7839 1981-06-09 **** null 10 7788 SCOTT ANALYST 7566 1987-04-19 **** null 20 7839 KING PRESIDENT null 1981-11-17 **** null 10 7844 TURNER SALESMAN 7698 1981-09-08 **** 0 30 7876 ADAMS CLERK 7788 1987-05-23 **** null 20 7900 JAMES CLERK 7698 1981-12-03 **** null 30 7902 FORD ANALYST 7566 1981-12-03 **** null 20	7566	JONES	MANAGER	7839	1981-04-02	****	null	20
7782 CLARK MANAGER 7839 1981-06-09 **** null 10 7788 SCOTT ANALYST 7566 1987-04-19 **** null 20 7839 KING PRESIDENT null 1981-11-17 **** null 10 7844 TURNER SALESMAN 7698 1981-09-08 **** 0 30 7876 ADAMS CLERK 7788 1987-05-23 **** null 20 7900 JAMES CLERK 7698 1981-12-03 **** null 30 7902 FORD ANALYST 7566 1981-12-03 **** null 20	7654	MARTIN	SALESMAN	7698	1981-09-28	****	1400	30
7788 SCOTT ANALYST 7566 1987-04-19 **** null 20 7839 KING PRESIDENT null 1981-11-17 **** null 10 7844 TURNER SALESMAN 7698 1981-09-08 **** 0 30 7876 ADAMS CLERK 7788 1987-05-23 **** null 20 7900 JAMES CLERK 7698 1981-12-03 **** null 30 7902 FORD ANALYST 7566 1981-12-03 **** null 20	7698	BLAKE	MANAGER	7839	1981-05-01	****	null	30
7839 KING PRESIDENT null 1981-11-17 **** null 10 7844 TURNER SALESMAN 7698 1981-09-08 **** 0 30 7876 ADAMS CLERK 7788 1987-05-23 **** null 20 7900 JAMES CLERK 7698 1981-12-03 **** null 30 7902 FORD ANALYST 7566 1981-12-03 **** null 20	7782	CLARK	MANAGER	7839	1981-06-09	****	null	10
7844 TURNER SALESMAN 7698 1981-09-08 **** 0 30 7876 ADAMS CLERK 7788 1987-05-23 **** null 20 7900 JAMES CLERK 7698 1981-12-03 **** null 30 7902 FORD ANALYST 7566 1981-12-03 **** null 20	7788	SCOTT	ANALYST	7566	1987-04-19	****	null	20
7876 ADAMS CLERK 7788 1987-05-23 **** null 20 7900 JAMES CLERK 7698 1981-12-03 **** null 30 7902 FORD ANALYST 7566 1981-12-03 **** null 20	7839	KING	PRESIDENT	null	1981-11-17	****	null	10
7900 JAMES CLERK 7698 1981-12-03 **** null 30 7902 FORD ANALYST 7566 1981-12-03 **** null 20	7844	TURNER	SALESMAN	7698	1981-09-08	****	0	30
7902 FORD ANALYST 7566 1981-12-03 **** null 20	7876	ADAMS	CLERK	7788	1987-05-23	****	null	20
7502 FORD ANALIST 7500 1501 12 05 Hull 20	7900	JAMES	CLERK	7698	1981-12-03	****	null	30
7934 MILLER CLERK 7782 1982-01-23 **** null 10	7902	FORD	ANALYST	7566	1981-12-03	****	null	20
	7934	MILLER	CLERK	7782	1982-01-23	****	null	10



C

# Real Application Security HR Demo Files

This appendix contains both the source files and log files. A detailed description of the HR Demo can be found in Real Application Security HR Demo.



The Real Application Security HR demo uses the HR Sample Schema provided by Oracle. To install the HR Sample schema, refer Installing the HR Schema.

This appendix describes the following topics:

- How to Run the Security HR Demo
- Scripts for the Security HR Demo
- Generated Log Files for Each Script

# C.1 How to Run the Security HR Demo

To run the Security HR demo, run the following scripts in the order shown:

- 1. Run the setup script hrdemo setup.sql, which creates the log file: hrdemo setup.log.
- 2. Run the demo script hrdemo.sql with direct logon, which creates the log file: hrdemo.log.
- 3. Run the demo script to explicitly create and attach to the Real Application Security session hrdemo session.sql, which creates the log file: hrdemo session.log.
- 4. Run the Java demo hrdemo.java file, which creates the log file: hrdemo.log.
- 5. Run the clean up script hrdemo clean.sql, which creates the log file: hrdemo clean.log.

# C.2 Scripts for the Security HR Demo

Table C-1 lists the scripts and generated log files with links to the content of each file.

Table C-1 HR Demo Scripts and Log Files

Scripts	Log Files
hrdemo_setup.sql	hrdemo_setup.log
hrdemo.sql	hrdemo.log
hrdemo_session.sql	hrdemo_session.log
hrdemo.java	hrdemo.log
hrdemo_clean.sql	hrdemo_clean.log

This section includes the following script files:

## C.2.1 hrdemo\_setup.sql

The source file for the set up script hrdemo\_setup.sql.

```
SET ECHO OFF
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100
SET ECHO ON
define passwd=&1
-- Introduction
______
-- The HR Demo shows how to use basic Real Application Security features.
-- The demo secures HR.EMPLOYEES table by creating a data security
-- policy that grants the table access to:
-- Data Security Policy
--(1) An employee can view their own record including SALARY column.
--(2) An IT engineer can view all employee records in IT department,
-- but cannot view employee's salaries.
--(3) An HR representative can view and update all employee records.
-- Sample Users and Their Role Grants:
-- 1) DAUSTIN, an application user in IT department. He has role employee
-- and it engineer. He can view employee records in IT department, but he
   cannot view the salary column except for his own.
-- 2) SMAVRIS, an application user in HR department. She has role employee
-- and hr representative. She can view and update all the employee records.
-- 1. SETUP - User and Roles
connect sys/&passwd as sysdba
-- Create an application role employee for common employees.
exec sys.xs principal.create role(name => 'employee', enabled => true);
-- Create an application role it engineer for IT department.
exec sys.xs principal.create role(name => 'it engineer', enabled => true);
-- Create an application role hr representative for HR department.
exec sys.xs principal.create role(name => 'hr representative', enabled => true);
-- create a database role for object privilege grants
create role db emp;
-- Grant DB EMP to the three application roles, so they have the required
-- object privileges to access the table.
grant db emp to employee;
grant db emp to it engineer;
grant db emp to hr representative;
```



```
-- Create two application users:
-- DAUSTIN (in IT department), granted employee and it engineer.
exec sys.xs principal.create user(name => 'daustin', schema => 'hr');
exec sys.xs principal.set password('daustin', 'welcome1');
exec sys.xs_principal.grant_roles('daustin', 'XSCONNECT');
exec sys.xs principal.grant roles('daustin', 'employee');
exec sys.xs_principal.grant_roles('daustin', 'it_engineer');
-- SMAVRIS (in HR department), granted employee and hr representative.
exec sys.xs principal.create user(name => 'smavris', schema => 'hr');
exec sys.xs_principal.set_password('smavris', 'welcome1');
exec sys.xs_principal.grant_roles('smavris', 'XSCONNECT');
exec sys.xs principal.grant roles('smavris', 'employee');
exec sys.xs principal.grant roles('smavris', 'hr representative');
-- Grant HR user policy adminisration privilege
exec sys.xs admin util.grant system privilege('ADMIN ANY SEC POLICY','HR');
-- 2. SETUP - Security class and ACL
-- Connect as HR
connect hr/hr;
-- Grant necessary object privileges to db emp role
-- This role will be used to grant the required object privileges to
-- application users.
grant select, insert, update, delete on hr.employees to db emp;
-- Create a security class hr privileges and include privileges from the predefined
DML security class.
-- hr privileges has a new privilege VIEW SALARY, which is used to control the
-- access to SALARY column.
declare
begin
 sys.xs security class.create security class(
   name => 'hr privileges',
   parent list => xs$name list('sys.dml'),
   priv list => xs$privilege list(xs$privilege('view salary')));
end;
-- Create three ACLs to grant privileges for the policy defined later.
 aces xs$ace list := xs$ace list();
begin
 aces.extend(1);
  -- EMP ACL: This ACL grants employee the privileges to view an employee's
             own record including SALARY column.
  aces(1) := xs$ace type(privilege list => xs$name list('select','view salary'),
                        principal_name => 'employee');
                                => 'emp_acl',
  sys.xs acl.create acl(name
                   ace list => aces,
                    sec class => 'hr privileges');
  -- IT ACL: This ACL grants it engineer the privilege to view the employee
             records in IT department, but it does not grant the VIEW SALARY
```

```
privilege that is required for access to SALARY column.
  aces(1) := xs$ace type(privilege list => xs$name list('select'),
                        principal name => 'it engineer');
                               => 'it_acl',
  sys.xs_acl.create_acl(name
                   ace list => aces,
                   sec class => 'hr privileges');
  -- HR ACL: This ACL grants hr representative the privileges to view and
update all
             employees' records including SALARY column.
  aces(1):= xs$ace_type(privilege_list => xs$name_list('select', 'insert',
                                  'update', 'delete', 'view salary'),
                       principal name => 'hr representative');
                               => 'hr acl',
  sys.xs acl.create acl(name
                   ace_list => aces,
                   sec class => 'hr privileges');
end:
/
-- 3. SETUP - Data security policy
-- Create data security policy for EMPLOYEE table. The policy defines three
-- realm constraints and a column constraint that protects SALARY column.
 realms xs$realm constraint list := xs$realm constraint list();
          xs$column constraint list := xs$column constraint list();
begin
 realms.extend(3);
  -- Realm #1: Only the employee's own record.
  -- employee can view the realm including SALARY column.
  realms(1) := xs$realm constraint type(
   realm => 'email = xs sys context(''xs$session'',''username'')',
   acl list => xs$name list('emp acl'));
  -- Realm #2: The records in the IT department.
  -- it engineer can view the realm excluding SALARY column.
  realms(2) := xs$realm constraint type(
  realm => 'department id = 60',
   acl_list => xs$name_list('it acl'));
  -- Realm #3: All the records.
  -- hr representative can view and update the realm including SALARY
  realms(3) := xs$realm constraint type(
   realm => '1 = 1',
   acl list => xs$name_list('hr_acl'));
  -- Column constraint protects SALARY column by requiring VIEW SALARY
  -- privilege.
  cols.extend(1);
  cols(1) := xs$column constraint type(
   column list => xs$list('salary'),
   privilege => 'view salary');
  sys.xs_data_security.create_policy(
                         => 'employees ds',
   realm_constraint_list => realms,
```

```
column constraint list => cols);
end;
-- Apply the data security policy to the table.
  sys.xs_data_security.apply_object_policy(
    policy => 'employees ds',
    schema => 'hr',
    object =>'employees');
end;
-- 4. SETUP - Validate the objects we have set up.
set serveroutput on;
  if (sys.xs diag.validate workspace()) then
    dbms output.put line('All configurations are correct.');
   dbms_output.put_line('Some configurations are incorrect.');
  end if;
end;
-- XS$VALIDATION TABLE contains validation errors if any.
-- Expect no rows selected.
select * from xs$validation table order by 1, 2, 3, 4;
-- 5. SETUP - Mid-Tier related configuration.
connect sys/&passwd as sysdba
-- create a session administrator who has only
-- RAS session administration privilege (no data privilege),
-- and is responsible to manage RAS session for each application user.
grant xs_session_admin, create session to hr_session identified by hr_session;
grant create session to hr common identified by hr common;
-- craete a dispatcher user for java demo, to set up session for application user
exec sys.xs principal.create user(name=>'dispatcher', schema=>'HR');
exec sys.xs_principal.set_password('dispatcher', 'welcome1');
exec sys.xs principal.grant roles('dispatcher', 'XSCONNECT');
exec sys.xs principal.grant roles('dispatcher', 'xsdispatcher');
exit
```

# C.2.2 hrdemo.sql

The source file for the hrdemo.sql script. This script runs the demo with direct logon.

```
SET ECHO OFF
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100
```

```
COLUMN EMAIL FORMAT A10
COLUMN FIRST NAME FORMAT A15
COLUMN LAST NAME FORMAT A15
COLUMN DEPARTMENT ID FORMAT 9999
COLUMN MANAGER ID FORMAT 9999
COLUMN SALARY FORMAT 999999
SET ECHO ON
-- HR Demo - PL/SQL with RAS direct logon user
   ______
-- This demo shows RAS runtime, using RAS direct logon user.
-- Each user directly connects to database and accesses employee table.
-- RAS policy is automatically enforced.
-- Connect as DAUSTIN, who has only employee and it engineer role
conn daustin/welcome1;
SET SECUREDCOL ON UNAUTH ******
-- DAUSTIN can view the records in IT department, but can only view his own
-- SALARY column.
select email, first name, last name, department id, manager id, salary
from employees order by email;
SET SECUREDCOL OFF
-- DAUSTIN cannot update the record.
update employees set manager_id = 102 where email = 'DAUSTIN';
-- Record is not changed.
select email, first name, last name, department id, manager id, salary
from employees where email = 'DAUSTIN';
-- Connect as SMAVRIS, who has both employee and hr representative role.
conn smavris/welcome1;
-- SMAVRIS can view all the records including SALARY column.
select email, first name, last name, department id, manager id, salary
from employees where department id = 60 or department id = 40
order by department id, email;
-- EMPLOYEES table has 107 rows, we expect to see all of them.
select count(*) from employees;
-- SMAVRIS can update the record.
update employees set manager_id = 102 where email = 'DAUSTIN';
-- Record is changed.
select email, first_name, last_name, department_id, manager_id, salary
from employees where email = 'DAUSTIN';
-- change the record back to the original.
update employees set manager id = 103 where email = 'DAUSTIN';
exit
```



# C.2.3 hrdemo\_session.sql

The source file for the hrdemo\_session.sql script. This script explicitly creates and attaches a Real Application Security session.

```
SET ECHO OFF
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100
COLUMN EMAIL FORMAT A10
COLUMN FIRST NAME FORMAT A15
COLUMN LAST NAME FORMAT A15
COLUMN DEPARTMENT ID FORMAT 9999
COLUMN MANAGER ID FORMAT 9999
COLUMN SALARY FORMAT 999999
SET ECHO ON
-- HR Demo - PL/SQL with Session API
-- This demo shows RAS runtime, using RAS user as application user.
-- The user does not logon to database, but a RAS session is created
-- and attached for each user before accessing employee table.
______
-- Connect as RAS session administrator.
connect hr session/hr session;
-- Variable used to remember the session ID.
var gsessionid varchar2(32);
-- Create an application session for SMARVIS and attach to it.
declare
  sessionid raw(16);
begin
  sys.dbms xs sessions.create session('SMAVRIS', sessionid);
  :gsessionid := rawtohex(sessionid);
  sys.dbms xs sessions.attach session(sessionid, null);
end ;
-- Display the current user, it should be SMAVRIS now.
select xs_sys_context('xs$session', 'username') from dual;
-- Display the enabled application roles and database roles.
select role name from v$xs session roles union
select role from session roles order by 1;
-- SMAVRIS can view all the records including SALARY column.
select email, first name, last name, department id, manager id, salary
from employees where department id = 60 or department id = 40
order by department id, email;
-- EMPLOYEES table has 107 rows, we expect to see all of them.
select count(*) from employees;
```



```
-- Disable hr representative role.
exec dbms xs sessions.disable role('hr representative');
-- SMAVRIS should only be able to see her own record.
select email, first name, last name, department id, manager id, salary
from employees where department id = 60 or department id = 40
order by department id, email;
-- Enable hr representative role.
exec sys.dbms xs sessions.enable role('hr representative');
-- SMAVRIS can view all the records again.
select email, first name, last name, department id, manager id, salary
from employees where department id = 60 or department id = 40
order by department id, email;
-- EMPLOYEES table has 107 rows, we expect to see all of them.
select count(*) from employees;
-- Detach and destroy the application session.
declare
  sessionid raw(16);
begin
  sessionid := hextoraw(:gsessionid);
  sys.dbms xs sessions.detach session;
 sys.dbms xs sessions.destroy session(sessionid);
end;
exit
```

## C.2.4 hrdemo.java

#### The source file for the Java demo is hrdemo.java.

```
import java.security.GeneralSecurityException;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.spec.InvalidKeySpecException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import oracle.jdbc.OracleDriver;
import oracle.jdbc.OracleResultSet;
import oracle.jdbc.OracleResultSet.AuthorizationIndicator;
import oracle.security.xs.AccessDeniedException;
import oracle.security.xs.InvalidXSNamespaceException;
import oracle.security.xs.InvalidXSUserException;
import oracle.security.xs.Role;
import oracle.security.xs.Session;
import oracle.security.xs.XSAccessController;
import oracle.security.xs.XSException;
```



```
import oracle.security.xs.XSSessionManager;
/**
 * A simple java application implemented using RAS.
* It shows:
 * - How to setup RAS session manager
 * - How to manage RAS sessions
 ^{\star}\, - How to use Column authorization indicator
   - How to check privileges using "checkAcl" function
public class hrdemo {
  // application connection, should be created with unprivileged user
  // in RAS case, the user only needs DB connection privilege
 private Connection appConnection = null;
  // RAS dispatcher's connection, should be create with a RAS dispatcher user
 private Connection mgrConnection = null;
  // RAS session manager, to manage session for application user
  // Must be instanciated with disptcher's connection
 private XSSessionManager manager = null;
 public static void main(String[] args) {
    try {
     DriverManager.registerDriver(new OracleDriver());
     if (args.length != 1) {
        System.out.println("Usage hrdemo dbURL");
        System.exit(1);
     hrdemo demo = new hrdemo();
     demo.setupConnection(args[0]);
     demo.queryAsUser("DAUSTIN");
     demo.queryAsUser("SMAVRIS");
     demo.cleanupConnection();
    } catch (Exception e) {
      // we don't handle exception for now
     e.printStackTrace();
  private void queryAsUser(String user) throws SQLException, XSException {
    System.out.println("\nQuery HR.EMPLOYEES table as user \"" + user + "\"");
     Session lws = manager.createSession(appConnection, user, null, null);
     manager.attachSession(appConnection, lws, null, null, null, null, null);
     queryEmployees(lws);
     manager.detachSession(lws);
     manager.destroySession(appConnection, lws);
  public void setupConnection(String url) throws SQLException, XSException,
```

```
GeneralSecurityException {
    // dispatcher's connection
    mgrConnection =
        DriverManager.getConnection(url, "dispatcher", "welcome1");
    // RAS session manager
    manager = XSSessionManager.getSessionManager(mgrConnection, 30, 2048000);
    // connection used for application query
    appConnection = DriverManager.getConnection(url, "hr common", "hr common");
  public void cleanupConnection() throws SQLException {
    mgrConnection.close();
    appConnection.close();
  public void queryEmployees(Session lws) throws SQLException, XSException {
    // using DB connection that has been attached to a RAS session
    Connection conn = lws.getConnection();
    String query = " select email, first_name, last_name, department_id, salary,
ora get aclids(emp) from hr.employees emp where department id in (40, 60, 100)
order by email";
    Statement stmt = null;
    ResultSet rs = null;
    System.out.printf(" EMAIL | FIRST NAME | LAST NAME | DEPT | SALARY |
UPDATE | VIEW SALARY\n");
    try {
      stmt = conn.createStatement();
      rs = stmt.executeQuery(query);
      while (rs.next()) {
        String email = rs.getString("EMAIL");
        String first name = rs.getString("FIRST NAME");
        String last name = rs.getString("LAST NAME");
        String department id = rs.getString("DEPARTMENT ID");
        String salary;
        if (((OracleResultSet)rs).getAuthorizationIndicator("SALARY") ==
AuthorizationIndicator.NONE) {
          salary = rs.getString("SALARY");
        else {
         salary = "****";
        byte[] aclRaw = rs.getBytes(6);
        String update, viewSalary;
    // call checkAcl to determine whether can update the database record
    if (XSAccessController.checkAcl(lws, aclRaw, "UPDATE")) {
          update = "true";
        else {
```

## C.2.5 hrdemo\_clean.sql

The source file for the cleanup script is hrdemo clean.sql.

```
SET ECHO OFF
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100
SET ECHO ON
define passwd=&1
connect hr/hr;
-- Remove policy from the table.
  sys.xs data security.remove object policy(policy=>'employees ds',
                                        schema=>'hr', object=>'employees');
end;
-- Delete security class and ACLs
exec sys.xs security class.delete security class('hr privileges',
xs admin util.cascade option);
exec sys.xs_acl.delete_acl('emp_acl', xs_admin_util.cascade_option);
exec sys.xs_acl.delete_acl('it_acl', xs_admin_util.cascade_option);
exec sys.xs_acl.delete_acl('hr_acl', xs_admin_util.cascade_option);
-- Delete data security policy
exec sys.xs data security.delete policy('employees ds', xs admin util.cascade option);
connect sys/&passwd as sysdba
-- Delete application users and roles
exec sys.xs principal.delete principal('employee', xs admin util.cascade option);
exec sys.xs principal.delete principal('hr representative',
xs admin util.cascade option);
exec sys.xs principal.delete principal('it engineer', xs admin util.cascade option);
```

```
exec sys.xs_principal.delete_principal('smavris', xs_admin_util.cascade_option);
exec sys.xs_principal.delete_principal('daustin', xs_admin_util.cascade_option);

-- Delete database role
drop role db_emp;

-- Delete session administrator
drop user hr_session;
-- Delete the common user used to connect to DB
drop user hr_common;

-- Delete dispatcher user used by mid-tier
exec sys.xs_principal.delete_principal('dispatcher',
xs_admin_util.cascade_option);

exit
```

# C.3 Generated Log Files for Each Script

This section includes the following log files that are generated from running the scripts listed in Table C-1:

- hrdemo\_setup.log
- hrdemo.log
- hrdemo\_run\_sess.log
- hrdemo.log
- hrdemo clean.log

## C.3.1 hrdemo\_setup.log

The hrdemo setup.log file.

```
SQL> @hrdemo setup
SQL>
SQL> define passwd=&1
Enter value for 1: sample
SQL>
______
SQL> -- Introduction
SQL>
______
SQL> -- The HR Demo shows how to use basic Real Application Security
features.
SQL> -- The demo secures HR.EMPLOYEE table by creating a data security
SQL> -- policy that grants the table access to.
SQL> -- Data Security Policy
SQL> --
SQL> -- (1) An employee can view their own record including SALARY
SQL> -- (2) An IT engineer can view all employee records in IT
department,
SQL> -- but cannot view employee's salaries.
```

```
SQL> --(3) An HR representative can view and update all employee records.
SQL> --
SOL> --
SQL> --Sample Users and Their Role Grants:
SQL> --1) DAUSTIN, an application user in IT department. He has role employee
SQL> -- and it engineer. He can view employee records in IT department,
but he
SQL> -- cannot view the salary column except for his own.
SQL> --2) SMAVRIS, an application user in HR department. She has role
SQL> -- and hr representative. She can view and update all the employee
records
SQL> --
SQL> -----
SQL> -- 1. SETUP - User and Roles
SOL> -----
SQL>
SQL> connect sys/&passwd as sysdba
SQL> -- Create an application role employee for common employees.
SQL> exec xs principal.create role(name => 'employee', enabled => true);
PL/SQL procedure successfully completed.
SQL>
SQL> -- Create an application role it engineer for IT department.
SQL> exec xs principal.create role(name => 'it engineer', enabled => true);
PL/SQL procedure successfully completed.
SOL>
SQL> -- Create an application role hr representative for HR department.
SQL> exec xs principal.create role(name => 'hr representative', enabled =>
true);
PL/SQL procedure successfully completed.
SOL>
SQL> -- create a database role for object privilege grants
SQL> create role db emp;
Role created.
SQL>
SQL> -- Grant DB EMP to the three application roles, so they have the
required
SQL> -- object privileges to access the table.
SQL> grant db emp to employee;
Grant succeeded.
SQL> grant db emp to it engineer;
Grant succeeded.
```

```
SQL> grant db emp to hr representative;
Grant succeeded.
SOL>
SQL> -- Create two application users:
SQL> -- DAUSTIN (in IT department), granted employee and it engineer.
SQL> exec xs principal.create user(name => 'daustin', schema => 'hr');
PL/SQL procedure successfully completed.
SQL> exec xs principal.set password('daustin', 'welcome1');
PL/SQL procedure successfully completed.
SQL> exec xs principal.grant roles('daustin', 'XSCONNECT');
PL/SQL procedure successfully completed.
SQL> exec xs principal.grant roles('daustin', 'employee');
PL/SQL procedure successfully completed.
SQL> exec xs principal.grant roles('daustin', 'it engineer');
PL/SQL procedure successfully completed.
SQL>
SQL> -- SMAVRIS (in HR department), granted employee and
hr representative.
SQL> exec xs principal.create user(name => 'smavris', schema => 'hr');
PL/SQL procedure successfully completed.
SQL> exec xs principal.set password('smavris', 'welcome1');
PL/SQL procedure successfully completed.
SQL> exec xs principal.grant roles('smavris', 'XSCONNECT');
PL/SQL procedure successfully completed.
SQL> exec xs principal.grant roles('smavris', 'employee');
PL/SQL procedure successfully completed.
SQL> exec xs principal.grant roles('smavris', 'hr representative');
PL/SQL procedure successfully completed.
SOL>
SQL> -- Grant HR user policy administration privilege
SQL> exec
xs admin util.grant system privilege('ADMIN ANY SEC POLICY','HR');
```

```
PL/SQL procedure successfully completed.
SQL>
SQL> -----
SQL> -- 2. SETUP - Security class and ACL
SOL> -----
SQL>
SQL>
SQL> -- Connect as HR
SQL> connect hr/hr;
Connected.
SQL>
SQL> -- Grant necessary object privileges to db emp role
SQL> -- This role will be used to grant the required object privileges to
SQL> -- application users.
SOL>
SQL> grant select, insert, update, delete on hr.employees to db emp;
Grant succeeded.
SOL>
SQL>
SQL> -- Create a security class hr privileges and include privileges from
the predefined DML security class.
SQL> -- hr privileges has a new privilege VIEW SALARY, which is used to
control the
SQL> -- access to SALARY column.
SQL> declare
 2 begin
    xs security class.create security class(
                  => 'hr privileges',
       name
       parent list => xs$name list('sys.dml'),
        priv list => xs$privilege list(xs$privilege('view salary')));
 6
 7 end;
 8 /
PL/SQL procedure successfully completed.
SQL>
SQL>
SQL> -- Create three ACLs to grant privileges for the policy defined later.
SQL> declare
      aces xs$ace list := xs$ace list();
     aces.extend(1);
     -- EMP ACL: This ACL grants employee the privileges to view an
employee's
 7
                 own record including SALARY column.
      aces(1) := xs$ace type(privilege list =>
xs$name list('select','view salary'),
 9
                           principal name => 'employee');
 10
 11
     xs acl.create acl(name => 'emp acl',
```

```
12
                       ace list => aces,
                       sec class => 'hr privileges');
 13
14
15
    -- IT ACL: This ACL grants it engineer the privilege to view
the employee
16
                 records in IT department, but it does not grant the
VIEW SALARY
17 --
                privilege that is required for access to SALARY
column.
    aces(1) := xs$ace type(privilege list => xs$name list('select'),
 19
                            principal name => 'it engineer');
 20
 21
   xs acl.create acl(name => 'it acl',
 22
                       ace list => aces,
 2.3
                       sec class => 'hr privileges');
 24
      -- HR ACL: This ACL grants hr representative the privileges to
view and update all
                 employees' records including SALARY column.
      aces(1):= xs$ace type(privilege list => xs$name list('select',
'insert',
28
                                            'update', 'delete',
'view salary'),
 29
                           principal name => 'hr representative');
 31
     xs acl.create acl(name => 'hr acl',
 32
                       ace list => aces,
 33
                       sec class => 'hr privileges');
34 end;
35 /
PL/SQL procedure successfully completed.
SQL>
SQL>
SQL>
SQL>
______
SQL> -- 3. SETUP - Data security policy
SQL> -- Create data security policy for EMPLOYEE table. The policy
defines three
SQL> -- realm constraints and a column constraint that protects SALARY
column.
SQL> declare
 2 realms xs$realm constraint list := xs$realm constraint list();
             xs$column constraint list :=
xs$column constraint list();
 4 begin
 5
     realms.extend(3);
    -- Realm #1: Only the employee's own record.
                  employee can view the realm including SALARY
column.
```

```
9
     realms(1) := xs$realm constraint type(
              => 'email = xs sys context(''xs$session'',''username'')',
 10
       realm
 11
       acl list => xs$name list('emp acl'));
 12
 13
      -- Realm #2: The records in the IT department.
14
                 it engineer can view the realm excluding SALARY column.
 15
      realms(2) := xs$realm constraint type(
 16
      realm => 'department id = 60',
 17
      acl list => xs$name list('it acl'));
 18
 19
     -- Realm #3: All the records.
 20
                hr representative can view and update the realm
including SALARY column.
 21
     realms(3) := xs$realm constraint type(
 22
      realm => '1 = 1',
 23
      acl list => xs$name list('hr acl'));
 24
    -- Column constraint protects SALARY column by requiring VIEW SALARY
 26 -- privilege.
 27 cols.extend(1);
 28
     cols(1) := xs$column constraint type(
    column list => xs$list('salary'),
 30
      privilege => 'view salary');
 31
 32
    xs data security.create policy(
 33
                           => 'employees ds',
      name
       realm constraint list => realms,
 35
        column constraint list => cols);
 36 end;
 37 /
PL/SQL procedure successfully completed.
SQL>
SQL>
SQL>
SQL> -- Apply the data security policy to the table.
SQL> begin
    xs data security.apply object policy(
 3
      policy => 'employees ds',
       schema => 'hr',
 5
       object =>'employees');
 6 end;
 7 /
PL/SQL procedure successfully completed.
SQL>
SOL>
SQL>
SQL> -----
SQL> -- 4. SETUP - Validate the objects we have set up.
SQL> -----
SQL> set serveroutput on;
SQL> begin
```

```
if (xs diag.validate workspace()) then
       dbms output.put line('All configurations are correct.');
 3
 4
 5
      dbms output.put line('Some configurations are incorrect.');
    end if;
 7 end;
Some configurations are incorrect.
PL/SQL procedure successfully completed.
SQL> -- XS$VALIDATION TABLE contains validation errors if any.
SQL> -- Expect no rows selected.
SQL> select * from xs$validation table order by 1, 2, 3, 4;
    CODE
-----
DESCRIPTION
______
______
NOTE
   -1020
No ACE in the ACL
[ACL "SYS"."NETWORK ACL 30D45882EF095A86E053B0AAE80AF5F8"]
1 row selected.
SQL>
SQL>
SQL>
______
SQL> -- 5. SETUP - additional configuration for Java demo.
______
SOL>
SQL> connect sys/&passwd as sysdba
Connected.
SQL>
SQL> -- create a session administrator who has only
SQL> -- RAS session administration privilege (no data privilege),
SQL> -- and is responsible to manage RAS session for each application
SQL> grant xs session admin, create session to hr session identified
by hr session;
Grant succeeded.
SQL> grant create session to hr common identified by hr common;
```

```
Grant succeeded.

SQL>
SQL> -- craete a dispatcher user for java demo, to set up session for application user
SQL> exec xs_principal.create_user(name=>'dispatcher', schema=>'HR');
PL/SQL procedure successfully completed.

SQL> exec xs_principal.set_password('dispatcher', 'welcome1');
PL/SQL procedure successfully completed.

SQL> exec xs_principal.grant_roles('dispatcher', 'XSCONNECT');
PL/SQL procedure successfully completed.

SQL> exec xs_principal.grant_roles('dispatcher', 'xsdispatcher');
PL/SQL procedure successfully completed.

SQL> exec xs_principal.grant_roles('dispatcher', 'xsdispatcher');
PL/SQL procedure successfully completed.
```

### C.3.2 hrdemo.log

The hrdemo.log file.

```
SQL> @hrdemo
SQL>
SQL>
SOL> -----
SQL> -- HR Demo - PL/SQL with RAS direct logon user
SOL> -----
SQL> -- This demo shows RAS runtime, using RAS direct logon user.
SQL> -- Each user directly connects to database and accesses employee table.
SQL> -- RAS policy is automatically enforced.
SQL> -----
SQL>
SQL> -- Connect as DAUSTIN, who has only employee and it engineer role
SQL> conn daustin/welcome1;
Connected.
SOL>
SQL> SET SECUREDCOL ON UNAUTH ******
SQL> -- DAUSTIN can view the records in IT department, but can only view his
SQL> -- SALARY column.
SQL> select email, first name, last name, department id, manager id, salary
 2 from employees order by email;
      FIRST_NAME LAST_NAME DEPARTMENT_ID MANAGER_ID SALARY
EMAIL
```

```
AHUNOLD Alexander
                    Hunold
                                             60
                                                     102
*****
        Bruce
BERNST
                     Ernst
                                             60
                                                     103
*****
DAUSTIN David Austin
                                             60
                                                     103
4800
DLORENTZ Diana Lorentz
                                            60
                                                     103
*****
VPATABAL Valli
                     Pataballa
                                           60
                                                    103
*****
5 rows selected.
SQL>
SQL>
SQL> SET SECUREDCOL OFF
SQL>
SQL>
SQL> -- DAUSTIN cannot update the record.
SQL> update employees set manager id = 102 where email = 'DAUSTIN';
0 rows updated.
SQL>
SQL> -- Record is not changed.
SQL> select email, first name, last name, department id, manager id,
salary
 2 from employees where email = 'DAUSTIN';
EMAIL FIRST NAME
                    LAST NAME
                                DEPARTMENT ID MANAGER ID
SALARY
DAUSTIN David Austin
                                           60 103
4800
1 row selected.
SQL>
SQL>
SQL> -- Connect as SMAVRIS, who has both employee and
hr representative role.
SQL> conn smavris/welcome1;
Connected.
SQL> -- SMAVRIS can view all the records including SALARY column.
SQL> select email, first name, last name, department id, manager id,
 2 from employees where department id = 60 or department id = 40
 3 order by department id, email;
       FIRST NAME LAST NAME
                                  DEPARTMENT ID MANAGER ID
EMAIL
SALARY
  ----- ---- ----- -----
```

```
-----
AHUNOLD Alexander Hunold
BERNST Bruce Ernst
SMAVRIS Susan
                     Mavris
                                              40
                                                      101 6500
                                                       102 9000
                                              60
                                             60
                                                      103 6000
DAUSTIN David
                     Austin
                                             60
                                                      103 4800
DLORENTZ Diana
VPATABAL Valli
                      Lorentz
                                             60
                                                     103 4200
                                             60 103 4800
                     Pataballa
6 rows selected.
SQL>
SQL> -- EMPLOYEES table has 107 rows, we expect to see all of them.
SQL> select count(*) from employees;
 COUNT(*)
-----
     107
1 row selected.
SQL>
SQL>
SQL>
SQL> -- SMAVRIS can update the record.
SQL> update employees set manager id = 102 where email = 'DAUSTIN';
1 row updated.
SQL>
SQL> -- Record is changed.
SQL> select email, first name, last name, department id, manager id, salary
 2 from employees where email = 'DAUSTIN';
EMAIL FIRST_NAME LAST_NAME DEPARTMENT_ID MANAGER_ID SALARY
DAUSTIN David
                                             60 102 4800
                     Austin
1 row selected.
SQL>
SQL> -- change the record back to the original.
SQL> update employees set manager id = 103 where email = 'DAUSTIN';
1 row updated.
SQL>
SQL> exit
```

### C.3.3 hrdemo\_run\_sess.log

```
The hrdemo_run_sess.log file.
SQL> @hrdemo_session
SQL>
```

```
SQL>
SQL>
______
SQL> -- HR Demo - PL/SQL with Session API
SQL> -- This demo shows RAS runtime, using RAS user as application
SQL> -- The user does not logon to database, but a RAS session is
SQL> -- and attached for each user before accessing employee table.
______
SQL> -- Connect as RAS session administrator
SQL> connect hr session/hr session;
Connected.
SQL>
SQL> -- Variable used to remember the session ID;
SQL> var gsessionid varchar2(32);
SQL> -- Create an application session for SMARVIS and attach to it.
SQL> declare
     sessionid raw(16);
 3 begin
    dbms xs sessions.create session('SMAVRIS', sessionid);
     :gsessionid := rawtohex(sessionid);
     dbms xs sessions.attach session(sessionid, null);
 7 end ;
 8 /
PL/SQL procedure successfully completed.
SQL>
SQL> -- Display the current user, it should be SMAVRIS now.
SQL> select xs sys context('xs$session','username') from dual;
XS SYS CONTEXT ('XS$SESSION', 'USERNAME')
SMAVRIS
1 row selected.
SQL>
SQL> -- Display the enabled application roles and database roles.
SQL> select role name from v$xs session roles union
 2 select role from session roles order by 1;
ROLE NAME
DB EMP
EMPLOYEE
HR REPRESENTATIVE
```

```
XSCONNECT
XSPUBLIC
XS CONNECT
6 rows selected.
SQL> -- SMAVRIS can view all the records including SALARY column.
SQL> select email, first name, last name, department id, manager id, salary
 2 from employees where department id = 60 or department id = 40
 3 order by department id, email;
EMAIL FIRST_NAME LAST_NAME DEPARTMENT_ID MANAGER_ID SALARY
SMAVRIS Susan
                       Mavris
                                               4.0
                                                       101 6500
AHUNOLD Alexander
BERNST Bruce
DAUSTIN David
                                              60
                                                       102 9000
                     Hunold
                      Ernst
                                               60
                                                       103 6000
                     Austin
                                              60
                                                        103 4800
                                              60
                                                      103 4200
DLORENTZ Diana
                      Lorentz
VPATABAL Valli Pataballa
                                             60 103 4800
6 rows selected.
SQL>
SQL> -- EMPLOYEES table has 107 rows, we expect to see all of them.
SQL> select count(*) from employees;
 COUNT(*)
     107
1 row selected.
SQL>
SQL> -- Disable hr representative role
SQL> exec dbms xs sessions.disable role('hr representative');
PL/SQL procedure successfully completed.
SQL>
SQL> -- SMAVRIS should only be able to see her own record.
SQL> select email, first name, last name, department id, manager id, salary
 2 from employees where department id = 60 or department id = 40
 3 order by department id, email;
EMAIL FIRST_NAME LAST_NAME DEPARTMENT_ID MANAGER_ID SALARY
40 101 6500
SMAVRIS Susan
                      Mavris
1 row selected.
SQL>
SOL>
SQL> -- Enable HR ROLE
SQL> exec dbms xs sessions.enable role('hr representative');
```

```
PL/SQL procedure successfully completed.
SQL>
SQL> -- SMAVRIS can view all the records again.
SQL> select email, first name, last name, department id, manager id,
salary
 2 from employees where department id = 60 or department id = 40
 3 order by department id, email;
EMAIL
        FIRST NAME
                     LAST NAME
                                   DEPARTMENT ID MANAGER ID
SALARY
SMAVRIS Susan Mavris
                                              40
                                                       101
6500
                                              60
AHUNOLD Alexander
                     Hunold
                                                       102
9000
BERNST Bruce
                                                       103
                     Ernst
                                              60
6000
DAUSTIN David Austin
                                              60
                                                       103
4800
DLORENTZ Diana
                     Lorentz
                                             60
                                                       103
4200
VPATABAL Valli
                     Pataballa
                                             60
                                                       103
4800
6 rows selected.
SQL>
SQL> -- EMPLOYEES table has 107 rows, we expect to see all of them.
SQL> select count(*) from employees;
 COUNT(*)
-----
     107
1 row selected.
SQL>
SQL> -- Detach and destroy the application session.
SQL> declare
 2 sessionid raw(16);
 3 begin
 4 sessionid := hextoraw(:gsessionid);
 5 dbms xs sessions.detach session;
    dbms xs sessions.destroy session(sessionid);
 6
 7 end;
 8 /
PL/SQL procedure successfully completed.
SOL>
SQL> exit
```

### C.3.4 hrdemo.log

The Java hrdemo.log file.

Query HR.EMPLOYEES table as user "DAUSTIN"						
EMAIL	FIRST_NAME	LAST_NAME	DEPT	SALARY	UPDATE	VIEW_SALARY
AHUNOLD	Alexander	Hunold	60	*****	false	false
		Ernst				
		Austin				
		Lorentz				
VPATABAL	Valli	Pataballa	60	*****	false	false
Query HR.E	MPLOYEES table	e as user "SM	AVRIS"			
EMAIL	FIRST_NAME	LAST_NAME	DEPT	SALARY	UPDATE	VIEW_SALARY
AHUNOLD	Alexander	Hunold	60	9000	true	true
		Ernst				
DAUSTIN	David	Austin	60	4800	true	true
DFAVIET	Daniel	Faviet	100	9000	true	true
DLORENTZ	Diana	Lorentz	60	4200	true	true
ISCIARRA	Ismael	Sciarra	100	7700	true	true
JCHEN	John	Chen	100	8200	true	true
JMURMAN	Jose Manuel	Urman	100	7800	true	true
		Popp				
NGREENBE	Nancy	Greenberg	100	12008	true	true
SMAVRIS	Susan	Mavris	40	6500	true	true
VPATABAL	Valli	Pataballa	60	4800	true	true

### C.3.5 hrdemo\_clean.log

The hrdemo clean.log file.

```
SQL> @hrdemo clean
SQL>
SQL> define passwd=&1
Enter value for 1: test
SQL>
SQL> connect hr/hr;
Connected.
SQL> -- Remove policy from the table.
SQL> begin
       xs data security.remove object policy(policy=>'employees ds',
                                             schema=>'hr',
object=>'employees');
 4 end;
PL/SQL procedure successfully completed.
SQL>
SQL> -- Delete security class and ACLs
SQL> exec xs security class.delete security class('hr privileges',
xs admin util.cascade option);
PL/SQL procedure successfully completed.
```

```
SQL> exec xs acl.delete acl('emp acl', xs admin util.cascade option);
PL/SQL procedure successfully completed.
SQL> exec xs acl.delete acl('it acl', xs admin util.cascade option);
PL/SQL procedure successfully completed.
SQL> exec xs acl.delete acl('hr_acl', xs_admin_util.cascade_option);
PL/SQL procedure successfully completed.
SQL> -- Delete data security policy
SQL> exec xs data security.delete policy('employees ds',
xs admin util.cascade option);
PL/SQL procedure successfully completed.
SOL>
SQL> connect sys/&passwd as sysdba
Connected.
SQL> -- Delete application users and roles
SQL> exec xs principal.delete principal('employee',
xs admin util.cascade option);
PL/SQL procedure successfully completed.
SQL> exec xs principal.delete principal('hr representative',
xs admin util.cascade option);
PL/SQL procedure successfully completed.
SQL> exec xs principal.delete principal('it engineer',
xs admin util.cascade option);
PL/SQL procedure successfully completed.
SQL> exec xs principal.delete principal('smavris',
xs admin util.cascade option);
PL/SQL procedure successfully completed.
SQL> exec xs principal.delete principal('daustin',
xs admin util.cascade option);
PL/SQL procedure successfully completed.
SQL>
SQL> -- Delete database role
SQL> drop role db emp;
Role dropped.
```

```
SQL>
SQL> -- Delete session administrator
SQL> drop user hr_session;

User dropped.

SQL> -- Delete the common user used to connect to DB
SQL> drop user hr_common;

User dropped.

SQL>
SQL> -- Delete dispatcher
SQL> exec xs_principal.delete_principal('dispatcher', xs_admin_util.cascade_option);

PL/SQL procedure successfully completed.

SQL>
SQL>
SQL> exit
```



D

### Troubleshooting Oracle Database Real Application Security

#### This appendix contains:

- About Real Application Security Diagnostics
- About Event-Based Tracing of Real Application Security Components
- About Exception State Dump Information
- About Session Statistics
- Using Middle-Tier Tracing

### D.1 About Real Application Security Diagnostics

Real Application Security uses an integrated infrastructure that spans across back-end databases, application servers, and application instances. Real Application Security components include diagnostic capabilities that enable you to locate, diagnose, and resolve problems in a Real Application Security system.

Real Application Security diagnostics make use of the database Diagnostic Framework (DFW) available in Oracle Database 12c Release 1 (12.1) and later. Functionality diagnostics allow you to track, investigate, and resolve functionality failures. You can use exception state dumps, event-based tracing, or default tracing to study and resolve functionality issues. Performance diagnostics enable you to identify and resolve performance issues.

The following sections discuss functionality and performance diagnostic techniques used in Real Application Security systems:

- About Using Validation APIs
- How to Check Which ACLs Are Associated with a Row for the Current User
- · How to Find If a Privilege Is Granted in an ACL to a User
- About Exception State Dumps
- About Event-Based Tracing
- About In-Memory Tracing
- About Statistics

### D.1.1 About Using Validation APIs

You should always validate objects after they are created. This includes objects, such as principals, security classes, ACLs, data security policies, and namespaces. You can also validate all these objects that exist in a workspace in a single operation. The XS\_DIAG package includes subprograms that you can use to diagnose potential problems in any of these created objects. See "XS\_DIAG Package" for more information. These packages are briefly described in the following table with links to each validation subprogram where examples of their usage are shown.

Subprogram	Description
VALIDATE_PRINCIPAL Function	Validates the principal.
VALIDATE_SECURITY_CLASS Function	Validates the security class.
VALIDATE_ACL Function	Validates the ACL.
VALIDATE_DATA_SECURITY Function	Validates the data security policy or validates the data security policy against a specific table.

Validates the namespace template.

Validates an entire workspace.

Table D-1 Summary of XS\_DIAG Subprograms

VALIDATE NAMESPACE TEMPLATE

VALIDATE\_WORKSPACE Function

**Function** 

## D.1.2 How to Check Which ACLs Are Associated with a Row for the Current User

To find which ACLs are associated with a particular row for the current user, use the <code>ORA\_GET\_ACLIDS</code> function. The <code>ORA\_GET\_ACLIDS</code> function returns a list of ACL IDS associated with a row instance of data security policy enabled tables for the current application user. If access to the current row has been granted, this function captures all ACL identifiers that are associated with the matching data realm constraints. See <code>"ORA\_GET\_ACLIDS Function"</code> for reference information and "About Checking ACLs for a Privilege" for tutorial information.

### D.1.3 How to Find If a Privilege Is Granted in an ACL to a User

To find if a privilege is granted in an ACL, use the <code>ORA\_CHECK\_ACL</code> function. The <code>ORA\_CHECKACL</code> function checks whether an application user has the queried application privileges according to a list of ACLs. If the specified application privileges have been granted to the application user, <code>ORA\_CHECKACL</code> returns 1. If they are not granted to the application user, then it returns 0. See "<code>ORA\_CHECK\_ACL</code> Function" for reference information and "About Checking ACLs for a Privilege" for tutorial information.

To list the ACLIDs associated with each row of a table, for example, the EMPLOYEE table, the user can use the following query:

```
select ORA GET ACLIDS(emp) from EMPLOYEE emp;
```

To list the result if a privilege, for example SELECT, is granted for each row of the EMPLOYEE table, the user can perform the following query:

```
select ORA CHECK ACL(ORA GET ACLIDS(emp), 'SELECT') from EMPLOYEE emp;
```

### D.1.4 About Exception State Dumps

When an exception occurs, the state information for Real Application Security components is dumped into trace files. Exception state dumps are analogous to crash site evidence for a plane crash.

A failure, like an internal error or server crash, causes a Diagnostic Data Extraction (DDE) routine to be invoked for each component. This dumps the current system,

session, and process state information into trace files. You can later analyze the cause of failure using the state information dumped into trace files.

### D.1.5 About Event-Based Tracing

Event-based tracing can be used to track events related to specific Real Application Security components. Event-based tracing helps in tracing the events that led up to a failure. For example, event number 46148 is used to trace application session events, such as createSession and attachSession.

### D.1.6 About In-Memory Tracing

In-memory tracing is a proactive tracing mechanism that is used do diagnose intermittent and hard to replicate errors. The in-memory tracing mechanism records component state changes and events in memory buffers. This is dumped to a trace file when a failure occurs. In-memory tracing is analogous to black box data that is used for plane crash investigation.

#### D.1.7 About Statistics

Real Application Security component statistics help identify performance issues in a Real Application Security system. Statistics include key data like the number of session create operations, principal invalidations, role-enabling operations, and so on.

# D.2 About Event-Based Tracing of Real Application Security Components

Event-based tracing can be used to track events related to specific Real Application Security components. Table D-2 lists the events assigned to Real Application Security components.

Table D-2 Real Application Security Components and Events

Real Application Security Components	Event (Oracle Error #)
Application Sessions (XSSESSION)	46148
Application Principals (XSPRINCIPAL)	46150
Security Classes (XSSECCLASS)	46149
ACLs (XSACL)	46110
Data Security (XSXDS)	46049
Mid-Tier Caches (XS_MIDTIER)	46151
Data Security VPD Rewrite (XSVPD)	10730



The following sections describe event-based tracing for individual Real Application Security components:

- About Application Sessions (XSSESSION) Event-Based Tracing
- About Application Principals (XSPRINCIPAL) Event-Based Tracing
- About Security Classes (XSSECCLASS) Event-Based Tracing
- About ACL (XSACL) Event-Based Tracing
- About Data Security (XSXDS and XSVPD) Event-Based Tracing

### D.2.1 About Application Sessions (XSSESSION) Event-Based Tracing

Use the following SQL statement to enable event-based tracing for the  ${\tt XSSESSION}$  component:

```
ALTER SESSION SET EVENTS '46148 trace name context forever, level="1", level="2", level="3";
```

Here, 46148 is the Oracle Database error number associated with XSSESSION events. You can set a trace level of 1 (low), 2 (medium), or 3 (high). Table D-3 describes the trace levels.

Alternatively, you can use the following statement:

```
ALTER SESSION SET EVENTS 'TRACE [XSSESSION] disk=[low, medium, high]'
```

You can find the location of this trace file by using the following SQL statement:

```
SHOW PARAMETER USER DUMP DEST;
```

Table D-3 shows the XSSESSION trace contents for each trace level.

Table D-3 XSSESSION Trace Contents

Event	Trace Level 1 (Low)	Trace Level 2 (Medium)	Trace Level 3 (High)
createSession	Includes the following:  User name Session Id for the session	Includes the following in addition to trace level 1 items:  User GUID  Session attributes such as create time, last authentication time, global variable namespace, and cookie information	Same as level 2



Table D-3 (Cont.) XSSESSION Trace Contents

Event	Trace Level 1 (Low)	Trace Level 2 (Medium)	Trace Level 3 (High)
attachSession	Includes the following:  User name  Session Id for the session	Includes the following in addition to trace level 1 items:  Roles	Includes the following in addition to trace level 1 and 2 items:  Application namespace with attribute values
detachSession	<ul><li>Includes the following:</li><li>User name</li><li>Session Id for the session before detaching</li></ul>	Same as level 1	Same as levels 1 and 2
createNamespace	Includes the following:  User name Session Id Application namespace with attribute values	Includes the following in addition to trace level 1 items:  Session attributes such as create time, last authentication time, global variable namespace, and cookie information	Includes the following in addition to trace level 1 and 2 items:  Namespace handler
switchUser	<ul><li>Includes the following:</li><li>User name</li><li>Session Id for the session</li></ul>	Includes the following in addition to trace level 1 items:  Roles	Includes the following in addition to trace level 1 and 2 items:  Application namespace with attribute values
assignUser	<ul><li>Includes the following:</li><li>User name</li><li>Session Id for the session</li></ul>	Includes the following in addition to trace level 1 items:  Roles	Includes the following in addition to trace level 1 and 2 items:  Application namespace with attribute values
setAttribute	<ul> <li>Namespace name</li> <li>Name and value of the given attribute before and after the setAttribute operation</li> </ul>	Same as level 1	Same as levels 1 and 2



Table D-3 (Cont.) XSSESSION Trace Contents

Event	Trace Level 1 (Low)	Trace Level 2 (Medium)	Trace Level 3 (High)
deleteAttribute	<ul> <li>Namespace name and</li> <li>Name and value of the given attribute before and after the deleteAttribute operation</li> </ul>	Same as level 1	Same as levels 1 and 2

In addition to the preceding event, you can use the named event,  $xs\_session\_state$  to dump the current state of application sessions. Use the following SQL statement to enable tracing for the  $xs\_session\_state$  event:

```
ALTER SESSION SET EVENTS 'immediate eventdump(xs session state)';
```

The event dump contains information on all session attributes in the User Global Area (UGA) memory, such as session Id, user name, create time, last authentication time, global variable namespace, and so on. The dump does not contain information on secure items such as passwords.

# D.2.2 About Application Principals (XSPRINCIPAL) Event-Based Tracing

Use the following SQL statement to enable event-based tracing for the XSPRINCIPAL component:

```
ALTER SESSION SET EVENTS '46150 trace name context forever, level="1", level="2", level="3";
```

Here, 46150 is the Oracle Database error number associated with XSPRINCIPAL events. You can set a trace level of 1 (low), 2 (medium), or 3 (high). Table D-4 describes the trace levels.

Alternatively, you can use the following statement:

```
ALTER SESSION SET EVENTS 'TRACE [XSPRINCIPAL] disk=[low, medium, high]';
```

You can find the location of this trace file by using the following SQL statement:

```
SHOW PARAMETER USER DUMP DEST;
```

Table D-4 shows the XSPRINCIPAL trace contents for each trace level.



Table D-4 XSPRINCIPAL Trace Contents

Event	Trace Level 1 (Low)	Trace Level 2 (Medium)	Trace Level 3 (High)
Enable Role	Includes the following:  User name Session Id for the session	Includes the following in addition to trace level 1 items:  If the Enable Role operation fails, then the cause is logged. For example, the operation may fail if the role does not exist or the user has not been granted the role	Same as levels 1 and 2
Disable Role	<ul> <li>Includes the following:</li> <li>All user enabled roles in the session after the role is disabled</li> </ul>	Includes the following in addition to trace level 1 items:  If the Disable Role operation fails, then the cause is logged.	Same as levels 1 and 2
Role Graph Traverse	<ul><li>Includes the following:</li><li>User name</li><li>Session Id for the session</li></ul>	Same as level 1	Same as levels 1 and 2

### D.2.3 About Security Classes (XSSECCLASS) Event-Based Tracing

Use the following  ${\tt SQL}$  statement to enable event-based tracing for the  ${\tt XSSECCLASS}$  component:

```
ALTER SESSION SET EVENTS '46149 trace name context forever, level="1", level="2", level="3";
```

Here, 46149 is the Oracle Database error number associated with XSSECCLASS events. You can set a trace level of 1 (low), 2 (medium), or 3 (high).

Alternatively, you can use the following statement:

```
ALTER SESSION SET EVENTS 'TRACE [XSSECCLASS] disk=[low, medium, high]';
```

You can find the location of this trace file by using the following SQL statement:

```
SHOW PARAMETER USER DUMP DEST;
```

The trace information includes the following:

- Content from the Security Class document, such as parent classes, child classes, privileges, and aggregate privileges
- For security class deletions, it includes information on parent classes that require invalidation from the cache



Exception related information, such as security class validation errors

### D.2.4 About ACL (XSACL) Event-Based Tracing

Use the following SQL statement to enable event-based tracing for the XSACL component:

```
ALTER SESSION SET EVENTS '46110 trace name context forever, level="1", level="2", level="3";
```

Here, 46110 is the Oracle Database error number associated with XSACL events. You can set a trace level of 1 (low), 2 (medium), or 3 (high).

Alternatively, you can use the following statement:

```
ALTER SESSION SET EVENTS 'TRACE [XSACL] disk=[low, medium, high]';
```

You can find the location of this trace file by using the following SQL statement:

```
SHOW PARAMETER USER DUMP DEST;
```

Table D-5 shows the XSACL trace contents for each trace level.

**Table D-5 XSACL Trace Contents** 

Event	Trace Level 1 (Low)	Trace Level 2 (Medium)	Trace Level 3 (High)
Check privilege against ACLs	Includes the following:  • ACL results during cursor sharing	Includes the following in addition to trace level 1 items:  • ACL evaluation including ACL loading	Same as levels 1 and 2

### D.2.5 About Data Security (XSXDS and XSVPD) Event-Based Tracing

Use the following SQL statement to enable event-based tracing for the XSXDS component:

```
ALTER SESSION SET EVENTS '46049 trace name context forever, level="1", level="2", level="3";
```

Here, 46049 is the Oracle Database error number associated with XSXDS events. You can set a trace level of 1 (low), 2 (medium), or 3 (high). Table D-6 describes the trace levels.

Alternatively, you can use the following statement:

```
ALTER SESSION SET EVENTS 'TRACE [XSXDS] disk=[low, medium, high]';
```

You can find the location of this trace file by using the following SQL statement:

```
SHOW PARAMETER USER DUMP DEST;
```

Table D-6 shows the XSXDS trace contents for each trace level.



Table D-6 XSXDS Trace Contents

Event	Trace Level 1 (Low)	Trace Level 2 (Medium)	Trace Level 3 (High)
Data Security Document (DSD) loaded into System Global Area (SGA)	Includes the following:  Security data realm constraint rules with resolved parameter values	Includes the following in addition to trace level 1 items:  • Access Control List (ACL) identifiers	Same as levels 1 and 2

Use the following SQL statement to enable event-based tracing for the XSVPD component:

ALTER SESSION SET EVENTS '10730 trace name context forever level [1, 2, 3]';

Here, 10730 is the Oracle Database error number associated with XSVPD events. You can set a trace level of 1 (low), 2 (medium), or 3 (high). Table D-6 describes the trace levels.

Alternatively, you can use the following statement:

ALTER SESSION SET EVENTS 'TRACE [XSVPD] disk=[low, medium, high]';

Table D-6 shows the XSVPD trace contents for each trace level.

Table D-7 XSVPD Trace Contents

Event	Trace Level 1 (Low)	Trace Level 2 (Medium)	Trace Level 3 (High)
<ul> <li>Data Security         Document (DSD)             loaded into             System Global             Area (SGA)     </li> <li>All subsequent             SQL statements             issued in the             current database             session</li> </ul>	Includes the following:  VPD view of XDS enabled objects during hard-parse, soft-parse, or SQL statement parsing  Data realm constraint rules with resolved parameter values, their corresponding ACL paths and ACL identifiers	Includes the following in addition to trace level 1 items:  Current application session user name and enabled roles when the SQL statement is parsed or run	Includes the following in addition to trace level 1 items:  • Contents of all ACLs associated with data realm constraints, which are associated with the XDS enabled objects in the query

### D.3 About Exception State Dump Information

When an exception occurs, the state information for Real Application Security components is dumped into trace files. Table D-8 describes the information dumped for individual Real Application Security components:

Table D-8 Real Application Security Components and First-Failure Dump Information

Real Application Security Component	Ex	ception Related Information
XSSESSION	•	Application session state information



Table D-8 (Cont.) Real Application Security Components and First-Failure Dump Information

Real Application Security Component	Ex	ception Related Information
XSPRINCIPAL	•	Application session role lists (all roles, enabled roles, disabled roles, and database roles of the application session)
	•	Role Graph hash table of the system
	•	User hash table with direct roles granted to the users in the system
	•	Principal row cache state

### **D.4 About Session Statistics**

Real Application Security component statistics help identify performance issues in a Real Application Security system. Table D-9 describes the statistics collected for individual Real Application Security components.

Table D-9 Real Application Security Components and Performance Statistics

Real Application Security Component	Performance Statistics Collected
XSSESSION	Number of application sessions created
	<ul> <li>Number of application sessions attached and detached</li> </ul>
	<ul> <li>Number of namespaces created</li> </ul>
	<ul> <li>Number of user callbacks executed</li> </ul>
XSPRINCIPAL	Number of roles enabled/disabled
	Number of principal cache misses
	<ul> <li>Number of principal invalidations</li> </ul>
Mid-tier caches	Number of session cache synchronizations
	Number of principal cache synchronizations
	<ul> <li>Number of security class cache synchronizations</li> </ul>

### D.5 Using Middle-Tier Tracing

Middle-tier tracing uses the package oracle.security.xs. It can be done as follows:

1. Specify logging options in a property file. For example,

```
handlers= java.util.logging.ConsoleHandler
.level= SEVERE
java.util.logging.ConsoleHandler.level = FINEST
java.util.logging.ConsoleHandler.formatter =
java.util.logging.SimpleFormatter
oracle.security.xs.level = FINEST
```

2. Apply the preceding configuration during JVM start up.

```
java -Djava.util.logging.config.file=logging.properties
```

The log output will be generated to the handlers (file, console) specified in the configuration.



Real Application Security user can use mid-tier java API for authentication, authorization, session management, and so forth. In case the user needs to debug on interfacing with mid-tier API, trace can be turned on. The trace can show basic call stacks, function involved, time used, parameters passed, returning value, and so forth.



### Glossary

#### access control entry (ACE)

An entry in the access control list that grants or denies access to a given principal. One or more ACEs are listed within an access control list (ACL), in which the ordering of the ACEs is relevant.

#### access control list (ACL)

A list of access control entries that determines which principals have access to a given resource or resources. In Oracle Database Real Application Security, you use ACLs to define user privileges.

#### **ACE**

See access control entry (ACE).

#### ACL

See access control list (ACL).

#### aggregate privilege

A privilege that contains other privileges. When an aggregate privilege has been granted or denied, then all of its child privileges are granted or denied as well.

#### application role

A role that can only be granted to a application user or to another application role.

#### application session

A user session that contains information pertinent only to the application. Unlike traditional "heavyweight" database sessions, an application session does not hold its own database resources such as transactions and cursors.



#### application user

A user account that does not own a schema and can create a application session through the middle tier to the database.

#### column level security

The ability to apply specific privileges to a table column.

#### custom privilege

A privilege not predefined by Oracle Database. See also system privilege.

#### data realm

A set of rows within a database table whose access you control by associating it with an access control list (ACL). It is comprised of one or more object instances. See also dynamic data realm constraint and static data realm constraint.

#### database role

A role that can only be granted to a database user. It is also called a heavyweight role. See also application role.

#### database user

A user account that is created within the database and has a schema. It is also called a heavyweight user. See also application user.

#### dynamic ACL

An access control list that has been associated with a dynamic data realm constraint.

#### dynamic application role

A role that is enabled only under certain conditions, for example, when a user has logged on using SSL, or during a specified period.

#### dynamic data realm constraint

An data realm whose where predicate is rerun each time the user performs a query on the data realm constraint data. See also static data realm constraint.

#### function security

The mechanism by which user access to an applications functionality is controlled. For example, for Oracle Database Real Application Security, use the <code>checkPrivilege()</code> method to check the privilege on the ACL for a row to determine if a specific privilege



on one or more given ACLs is associated with that row. See About the Check Privilege API for more information.

#### globally unique identifier (GUID)

The external ID that applications can use to manage the user's session information. This identifier is not guaranteed to be unique across all tiers, but the number of unique keys that comprises it is so large that the chances of it being duplicated are small. See also unique identifier (UID).

#### **GUID**

See globally unique identifier (GUID).

#### heavyweight role

A traditional database role.

#### heavyweight user

A traditional database user account that owns a schema.

#### namespace

A container consisting of attribute-value pairs that reflects the state of the application session.

#### object instance

A single relational table row that is part of an data realm. It is identified by its primary key value.

#### password verifier

A hashed version of a clear text password, which is then encoded as a BASE64 encoded string.

#### principal

A user or collection of users alternately called a **group** or a **role**. See also application user and application role.

#### privilege

A right or permission that can be granted or denied to a principal. See also aggregate privilege, custom privilege, and system privilege.



#### security class

A named collection of privileges that can be associated with an ACL.

#### static ACL

An access control list that has been associated with a static data realm constraint.

#### static data realm constraint

An data realm whose WHERE predicate is stored in cache, so that it is not rerun each time the user performs a query on the data realm constraint data. See also dynamic data realm constraint.

#### system privilege

Predefined privilege supplied by Oracle Database. See also custom privilege.

#### unique identifier (UID)

A unique internal identifier that Oracle Database uses to track the user or role. It is used to manage the user's session information across the database enterprise. See also globally unique identifier (GUID).

#### UID

See unique identifier (UID).

#### user switch

The ability of an application user to proxy as another user. The application state (that is, namespaces and attributes) is maintained from the previous user, but the security context reflects that of the new user.



### Index

A	See access control lists
	ACLs and ACEs
access control entry (ACE)	about, <i>4-7</i>
about, <i>1-8</i>	creating, 4-8
definition, 4-8	aggregate privilege
access control lists (ACL)	about, <i>1-8</i>
about, <i>1-8</i>	benefits, 1-8
directories	definition, 4-1
trace files, using to resolve predicate	ALL grant, 4-3
errors, 5-7	ALL privilege, 4-3
dynamic data realm constraints	ALL_XDS_ACL_REFRESH view, 9-34
about, 5-7	ALL_XDS_ACL_REFSTAT view, 9-34
ACL evaluation order, 5-12	ALL_XDS_LATEST_ACL_REFSTAT view, 9-35
evaluation order, 5-12	ALL XS ACES view, 9-19
static data realm constraints	ALL_XS_ACL_PARAMETERS view, 9-27
ACL evaluation order, 5-12	ALL_XS_ACLS view, 9-17
static data realms	ALL XS APPLIED POLICIES view, 9-29
about, 5-6	ALL_XS_COLUMN_CONSTRAINTS view, 9-28
user-managed	ALL XS IMPLIED PRIVILEGES view, 9-13
example, 5-11	ALL XS INHERITED REALMS view, 9-25
ACE	ALL XS POLICIES view, 9-21
definition, 4-7	ALL XS PRIVILEGES view, 9-11
evaluation order, 4-13	ALL_XS_REALM_CONSTRAINTS view, 9-23
acl	ALL XS SECURITY CLASS DEP view, 9-16
troubleshooting, <i>D-8</i>	ALL_XS_SECURITY_CLASSES view, 9-15
ACL	anonymous user, 7-1
	application integration
adding ACE, 4-10	support for external users and roles, 7-1
binding, 4-16	application privileges
changing security class, 4-10	
constraining inheritance, 4-14	about, 1-8
create, 4-8	granting to principles, 2-17
extending inheritance, 4-13	application roles, 1-5, 2-12
identifiers	about, 1-5, 2-12
master-detail tables, retrieving ACL	creating dynamic role, 2-13, 2-14
identifiers for, 10-4	creating regular role, 2-13, 2-14
inheritance, 4-13	granting database role to an application role
constraining, 4-14	2-18
extending, 4-13	granting to another application role, 2-18
inheritance ACL	granting to existing application user, 2-18
changing parent ACL, 4-10	granting to new application user, 2-17
multilevel authentication, 4-12	using effective dates, 2-16
removing ACL, 4-10	validating, 2-15
scope	application sessions, 3-1
definition, 4-5	about, <i>3-1</i>
ACLS	advantages, 3-3



application sessions (continued)	application users (continued)
attaching, 3-5	application sessions, initializing namespaces,
cookies, setting for, 3-7	3-18 <b>–</b> 3-21
creating, 3-2, 3-3, 11-4	application sessions, initializing namespaces
creating anonymous application session, 3-4	explicitly, 3-22
database session	application sessions, switching to, 3-8
attaching to, 11-5	compared with database user, 1-6
detaching from, 11-19	creating, 2-2
destroying, 11-19	direct login users, 2-4
event handling, 3-11	creating direct login user, 2-4
global callback events, using, 3-11	definition, 1-5
namespace	general procedure, 2-2
creating, 11-9	modifying, 2-2
deleting, 11-14	validating, 2-12
namespaces	application users and roles
attribute values, getting, 11-12	troubleshooting, D-6
attribute values, setting, 11-11	applying
attributes, getting, 3-24	additional application privileges
attributes, setting, 3-23	to a column, 5-8
deleting, 3-25	assigning
roles	an application user to an anonymous
disabling for specified session, 11-16	application session, 3-7
enabling for specified session, 11-15	attaching
roles, disabling from session, 3-27	an application session, 3-5
roles, enabling for session, 3-26	and application session, 5-5
saving, 11-19	DBA_XS_AUDIT_POLICY_OPTIONS view,
	1-16
security context, setting, 11-8	
session cookie	DBA_XS_AUDIT_TRAIL view, 1-16
setting, 11-16	DBA_XS_ENB_AUDIT_POLICIES view, 1-16
session state manipulating, 3-15	in an Oracle Database Real Application
switch user, 11-8	Security environment, 1-16
troubleshooting, D-4	unified auditing, 1-16, 9-3, 9-4
users, assigning to, 3-7	authentication
users, creating namespace templates, 3-17	multilevel, 4-12
users, custom attributes, 3-25	strong, <i>4-12</i>
users, destroying, <i>3-14</i>	weak, <i>4-12</i>
users, detaching from, 3-13	
users, initializing namespaces, 3-18-3-21	С
users, initializing namespaces explicitly, 3-22	
users, switching to, 3-8	callback event handler procedure
application sessions in the database	creating, 3-9
architecture figure, 3-2	checking
application user roles	ACLs for a privilege, 4-12
application sessions, disabling from, 3-27	checking security attribute
application sessions, enabling for, 3-26	using getSecurityAttribute method
disabling for specified session, 11-16	SecurityAttribute returns value
enabling for specified session, 11-15	ENABLED, B-6
application users, 2-2	SecurityAttribute returns value NONE,
about, 1-5, 2-1	B-6
application sessions, assigning to, 3-7	SecurityAttribute returns value
application sessions, creating namespace	UNKNOWN, B-6
templates, 3-17	checking user authorization indicator
application sessions, custom attributes, 3-25	using getAuthorizationIndicator method
application sessions, destroying, 3-14	AuthorizationIndicator returns value
application sessions, detaching from, 3-13	NONE, B-7

checking user authorization indicator (continued) using getAuthorizationIndicator method (continued) AuthorizationIndicator returns value UNAUTHORIZED, B-7 AuthorizationIndicator returns value	data realms, 5-6 about, 5-1 definition, 4-15 structure, 5-5 See also dynamic data realms, static data
UNKNOWN, B-7	realms
column authorization	data security
JDBCI interface, B-6	about, <b>5-1</b>
OCI interface, <i>B-1</i>	ACLs, 4-15
COLUMN_AUTH_INDICATOR function, 10-1	automatic refreshment for static ACL, 11-50
column-level security, 5-8	troubleshooting, D-8
configuring	with Oracle Database Real Application
an application role, 2-14	Security, 1-4
application roles, 2-12	data security documents
application user switch	example, 5-2
proxying an application user, 2-10	privileges
application users, 2-1	security checks, how handled, 5-21
global callback event handlers	privileges, column-level security, 5-8
for an application session, 3-11	data security policy
constraining ACL inheritance	tables
definition, 4-14	enabling, <i>11-48</i>
cookies	removing from, 11-47
application sessions, setting for, 3-7	data security privileges
create views	alter refreshment for static ACL, 11-34, 11-51
using BEQUEATH clause, 5-23	automatic refreshment for static ACL, 11-33
creating	database role
ACLs and ACEs, 4-8	about, <i>1-5</i>
anonymous application session, 3-4	database user
application sessions, 3-3	about, <i>1-5</i>
application user accounts, 2-2	compared with application user, 1-6
application users, 2-2	DataSecurity module, 4-15
custom attributes	DBA_XDS_ACL_REFRESH view, 9-36
in application session, 3-25	DBA_XDS_ACL_REFSTAT view, 9-36
direct login user, 2-4	DBA_XDS_LATEST_ACL_REFSTAT view, 9-37
dynamic application role, 2-13, 2-14	DBA_XS_ACES view, 4-8, 4-15, 9-18
namespace templates, 3-17	DBA_XS_ACL_PARAMETERS view, 9-25
namespaces	DBA_XS_ACLS view, 4-15, 9-16
using namespace templates, 3-15	DBA_XS_ACTIVE_SESSIONS view, 9-31
regular application role, 2-13, 2-14	DBA_XS_APPLIED_POLICIES view, 9-28
security class, 4-4	DBA_XS_AUDIT_POLICY_OPTIONS view, 1-16,
simple application user account, 2-3	9-1
the other control of	DBA_XS_AUDIT_TRAIL view, 1-16, 9-1
<b>D</b>	DBA_XS_COLUMN_CONSTRAINTS view, 9-27
D	DBA_XS_DYNAMIC_ROLES view, 9-9
data realm constraints	DBA_XS_ENB_AUDIT_POLICIES view, 1-16,
affect on database tables, 5-11	9-1
membership methods, 5-5	DBA_XS_EXTERNAL_PRINICIPALS view, 9-5
membership rule (WHERE predicate)	DBA_XS_IMPLIED_PRIVILEGES view, 9-12
about, 5-5	DBA_XS_INHERITED_REALMS view, 9-24
membership rules	DBA_XS_MODIFIED_POLICIES view, 9-30
session variables, guideline for, 5-5	DBA_XS_NS_TEMPLATE_ATTRIBUTES view,
parameterized	9-33
about, 5-5	DBA_XS_NS_TEMPLATES view, 9-32
types defined by WHERE predicates, 5-5	DBA_XS_OBJECTS view, 9-4
5,511 do	DBA_XS_POLICIES view, 9-20

DBA_XS_PRINICIPALS view, 9-5	determining
DBA_XS_PRIVILEGE_GRANTS view, 9-13	invoker's rights use for nested program units
DBA_XS_PRIVILEGES view, 4-15, 9-10	using BEQUEATH clause when creating
DBA_XS_PROXY_ROLES view, 9-9	views, 5-23
DBA_XS_REALM_CONSTRAINTS view, 9-21	the invoking application user
DBA_XS_ROLE_GRANTS view, 9-10	using SQL functions, 5-25
DBA_XS_ROLES view, 9-8	direct application user accounts
DBA_XS_SECURITY_CLASS_DEP view, 4-15,	setting password verifiers, 2-5
9-15	disabling
DBA_XS_SECURITY_CLASSES view, 4-15,	application roles
9-14	for an application session, 3-27
DBA_XS_SESSION_NS_ATTRIBUTES view,	displaying secure column values
9-32	using SQL*Plus SET SECUREDCOL
DBA_XS_SESSION_ROLES view, 9-31	command, 5-22
DBA_XS_SESSIONS view, 9-30	dynamic application role, 2-13
DBA_XS_USERS view, 9-6	dynamic application roles
DBMS XS SESSIONS PL/SQL package	
	predefined, 2-15
about, 11-1	dynamic data realm constraints
ADD_GLOBAL_CALLBACK, 11-20	about, 5-7
ASSIGN_USER, 3-20	ACL evaluation order, 5-12
ATTACH_SESSION, 3-19	
constants, 11-2	E
CREATE_ATTRIBUTE, 3-25	<u>-</u>
CREATE_NAMESPACE, 3-22	enabling
CREATE_SESSION, 3-18	application roles
DELETE_GLOBAL_CALLBACK, 3-11, 11-22	for application session, 3-26
DELETE_NAMESPACE, 3-25	event handlers, 11-20
DESTROY_SESSION, 3-14	See also global callback events
DETACH_SESSION, 3-13	event-based tracing
DISABLE_ROLE, 3-27	about, <i>D-3</i>
ENABLE_GLOBAL_CALLBACK, 11-21	components, D-3
ENABLE_ROLE, 3-26	examples
GET_ATTRIBUTE, 3-24	JDBC
object types, constructor functions, 11-2	security attributes, checking, B-8
SAVE_SESSION, 3-12	user authorization, checking, B-8
security model, 11-2	OCI return codes, B-1
SET ATTRIBUTE, 3-23	Real Application Security policy on master-
SWITCH_USER, 2-10, 3-21	detail related tables, 5-13
default security class	exception dumps, D-9
definition, 4-5	exception state dumps, <i>D-2</i>
defining a basic data security policy	extending ACL inheritance
implementation tasks, 5-26	definition, 4-13
disable a data security policy for a table,	external roles, 7-1
5-32	external users, 7-1
use case, 5-26	namespaces for, 7-2
deleting	session modes, 7-1
namespaces	
·	secure mode, 7-1
in application session, 3-25	trusted mode, 7-1
destroying	external users and external roles
application session, <i>3-14</i>	createSession method, 7-2
detaching	for application integration, 7-1
application session	session APIs for, 7-2
from a traditional database session, 3-13	

F	Java environment (continued)
firewell 4.12	authorizing application users using ACLs,
firewall, 4-12	6-14
authentication, 4-14 foreign_key	changing the middle-tier cache size, 6-3
specifies foreign key of detail table, 5-13	clearing the cache, 6-4 getting the maximum cache idle time, 6-3
specifies foreign key of detail table, 3-13	getting the maximum cache size, 6-3
	removing entries from the cache, 6-4
G	removing entries from the cache, getting
getting	the high watermark for cache,
session attributes	6-4
in application session, 3-24	removing entries from the cache, getting
global callback events, 11-20	the low watermark for cache, 6-4
about, <i>3-11</i>	removing entries from the cache, setting
adding, 11-20	the watermark, 6-4
deleting, 11-22	setting the maximum cache size, 6-3
enable or disable, 11-21	setting the middle-tier cache idle time,
granting	6-3
application privileges to principles, 2-17	checking if application role is enabled, 6-8
application role	constructing an ACL identifier, 6-14
to existing application user, 2-18	creating a session namespace attribute, 6-10
to new application role, 2-18	creating a user session, 6-5
to new application user, 2-17	creating an application session, 7-2
database role	creating namespaces, 6-8
to an application role, 2-18	deleting namespaces, 6-9
	deleting session namespace attributes, 6-11
	destroying an application session, <i>6-13</i>
	detaching an application session, 6-13 disabling application roles, 6-7
in-memory tracing, D-3	enabling and disabling application roles, 6-7
inheritance	enabling application roles, 6-7
master-detail related tables, 5-12	getting a session namespace attribute, 6-10
inheritedFrom element, components, 5-13	getting data privileges associated with a
initializing	specific ACL, 6-15
namaspace when session is attached, <i>3-19</i>	getting the application user ID for the
namespace, 3-20	session, 6-12
application user is switched in application	getting the Oracle connection associated with
session, 3-21	the session, 6-12
when session is created, 3-18	getting the session cookie, 6-12
namespaces	getting the session ID for the session, 6-12
explicitly, 3-22	getting the string representation of the
• •	session, 6-12
J	implicitly creating namespaces, 6-9
J	initializing the middle tier, 6-1
Java environment	mid-tier configuration mode, 6-1
aborting a session, 7-9	privileges for the session manager, 6-1
assigning a user to a session, 7-7	roles for the session manager, 6-1
assigning or switching an application user,	using getSessionManager method, 6-1 listing session namespace attributes, 6-10
6-6	performing namespace operations as
attaching an application session, 7-4	session manager, 6-11
external role behavior, 7-6	performing namespace operations as
attachng an application session, 6-5	session user, 6-8
authenticating users using Java APIs, 6-14	resetting session namespace attributes, 6-11
	saving a session, 7-9



Java environment (continued) setting a session namespace attribute, 6-10 setting session cookie as session manager, 6-13 setting session inactivity timeout as session manager, 6-12 using namespace attributes, 6-9 using the checkAcl method, 6-15 JDBC column authorization, interface for, B-6	namespaces (continued) attribute values (continued) setting, 11-11 attributes creating, 11-10 deleting, 11-14 resetting, 11-13
M	OCI parameter handle attribute OCI_ATTR_XDS_POLICY_STATUS, B-4
	OCI_XDS_POLICY_ENABLED value,
master detail data realm foreign_key specifies foreign key of detail table, 5-13 parentObjectName element specifies name of master table, 5-13	B-4 OCI_XDS_POLICY_NONE value, B-4 OCI_XDS_POLICY_UNKNOWN value, B-4 OCI return codes
parentSchemaName element specifies name of schema containing	ORA-24530  column value is unauthorized to the user,
master table, 5-13 primary_key	<i>B-1</i> ORA-24531
specifies primary key from master table,  5-13	column value authorization is unknown,  8-1
when element	ORA-24536
specifies a predicate for detail table, 5-13 master-detail tables  ACL	column authorization unknown, <i>B-2</i> ORA_CHECK_ACL function, <i>10-4</i> , <i>D-2</i> ORA_CHECK_PRIVILEGE function, <i>10-5</i>
identifiers, retrieving, 10-4 inheritedFrom element, components, 5-13	ORA_GET_ACLIDS function  See ORA_GET_ACLIDS function
Real Application Security policies about, 5-12 creating for, 5-13	ORA_INVOKING_USER function returns name of current database user, 5-25 ORA_INVOKING_USERID function
Materialized View, 5-6	returns ID of current database user, 5-25
membership rules (WHERE predicate) in data realm constraints	ORA_INVOKING_XS_USER function returns name of current Real Application
about, 5-5 membership rules in data realm constraints session variables, guideline for, 5-5	Security application user, 5-25 ORA_INVOKING_XS_USER_GUID function returns ID of current Real Application
modifying	Security application user, 5-25 ORA-24530
application users, 2-2 multilevel authentication, 4-12 definition, 4-12	column value is unauthorized to the user  OCI return code, <i>B-1</i>
using, 4-12	ORA-24531 column value authorization is unknown
N	OCI return code, <i>B-1</i> ORA-24536
namespaces	column authorization unknown
application sessions	OCI return code, <i>B-2</i>
attributes, getting, 3-24	ORA-28113((colon)) policy predicate has error
attributes, setting, 3-23	message, 5-7 Oracle Call Interface (OCI)
creating, 11-9	column authorization, interface for, <i>B-1</i>
deleting, 3-25, 11-14 attribute values	Oracle Database Real Application Security
getting, 11-12	about data security, 1-4



Oracle Database Real Application Security (continued)	pluggable databases
access control entry (ACE), 1-8	Oracle Real Application Security support for,
access control list (ACL), 1-8	1-17
advantages of, 1-2	predefined objects
aggregate privilege, 1-8	ACLs
application privileges, 1-8	NS_UNRESTRICTED_ACL, A-4
application session concepts, 1-10	SESSIONACL, A-4
architecture, 1-2	SYSTEMACL, A-4
data security concepts, 1-3	database roles
data security policy, 1-9	PROVISIONER, A-2
flow of design and development, 1-10	XS_CACHE_ADMIN, A-3
principals	XS_NAMESPACE_ADMIN, A-2
users and roles, 1-5	XS_SESSION_ADMIN, A-2
security classes, 1-8	dynamic application roles, 2-15
security components of, 1-3	DBMS_AUTH, A-2
use case scenario example policy, 1-12	DBMS_PASSWD, A-2
component requirements, 1-14	EXTERNAL_DBMS_AUTH, A-2
description and security requirements,	MIDTIER_AUTH, A-2
1-13	XSAUTHENTICATED, A-2
implementation overview, 1-14	XSSWITCH, A-2
what is, <b>1-1</b>	namespaces
Oracle Label Security	XS\$GLOBAL VAR, A-3
context established during attach session,	XS\$SESSION, A-3
2-7	regular application roles
context established in application session,	XSBYPASS, A-1
3-5	XSCACHEADMIN, A-1
context established in named user's	XSCONNECT, A-2
application session, 3-7	XSDISPATCHER, A-1
context switches to target_user session, 3-8	XSNAMESPACEADMIN, A-1
Oracle Virtual Private Database (VPD)	XSPROVISIONER, A-1
extended for Real Application Security, 5-1	XSPUBLIC, A-1
oracle.jdbc.OracleResultSetMetaData interface	XSSESSIONADMIN, A-1
getAuthorizationIndicator method	security classes
about, B-7	DML, <i>A-3</i>
example, <i>B-8</i>	NSTEMPLATE_SC, A-3
getSecurityAttribute method	SESSION, A-3
about, B-6	SYSTEM, A-3
example, <i>B-8</i>	users
1 /	XSGUEST, A-1
D	primary_key
P	specifies primary key from master table, 5-13
parameterized ACL, 4-16	principals
parameterized data realm constraints	about, <u>1-5</u>
about, 5-5	privileges
parentObjectName element	about application, 1-8
specifies name of master table, 5-13	check, 4-12
parentSchemaName element	constrain, 4-14
specifies name of schema containing master	data security documents
table, 5-13	columns, applying additional to, 5-8
password verifiers	security checks, how handled, 5-21
direct application user accounts, 2-5	9,
PL/SQL functions	D
COLUMN_AUTH_INDICATOR, 10-1	R
XS_SYS_CONTEXT, 10-2	regular application role, 2-13
7.5_515_5511L71, 10 Z	rogulai application rolo, 2 10

roles, 1-5, 2-12	session service (continued)	
dynamic	Oracle Platform Security Service (OPSS),	
assigning to user, 11-7	8-1	
removing from user, 11-7	privilege elevation, 8-4	
See also application roles	privilege elevation API, 8-13	
	Real Application Security servlet filter, 8-4	
S	session APIs, 8-4, 8-11	
3	session filter, 8-5	
scope, ACL	session filter operation, 8-5	
definition, 4-5	supports JavaEE web application	
security class	using OPSS as application security	
about, 1-8	provider, 8-1	
adding parent security class	SessionNamespace	
inheritance, 4-5	deleteAttribute, 6-11	
configuration, 4-3	toString, 6-9	
create, 4-4	SET SECUREDCOL command	
	SQL*Plus	
definition, 4-3	•	
inheritance, 4-4	displaying secure column values, 5-22	
inheritance security class	setting	
adding privileges, 4-5	a cookie for an application session, 3-7	
deleting, 4-5	password verifiers, 2-5	
description string, 4-5	session attributes	
removing implied privileges, 4-5	in application session, 3-23	
removing parent, 4-5	SQL functions	
removing privileges, 4-5	ORA_CHECK_ACL, 10-4, D-2	
manipulating, 4-5	ORA_CHECK_PRIVILEGE, 10-5	
troubleshooting, D-7	ORA_INVOKING_USER	
session, 3-1	returns name of current datanase user,	
application, 3-1	5-25	
IDs	ORA_INVOKING_USERID	
authentication time, updating, 11-17	returns ID of current database user, 5-25	
time-out values, setting, 11-18	ORA_INVOKING_XS_USER	
statistics, <i>D-10</i>	returns name of current Real Application	
See also application sessions	Security application user, 5-25	
Session	ORA_INVOKING_XS_USER_GUID	
isRoleEnabled, 6-8	returns ID of current Real Application	
setCookie, 6-13	Security application user, 5-25	
setInactivityTimeout, 6-12	TO_ACLID, <i>10-6</i>	
session cookie	SQL operators	
application sessions	ORA_CHECK_ACL	
setting, <i>11-16</i>	checking ACLs for a privilege, 4-12	
session privilege scoping through ACL, 3-28	static data realms	
session service	about, 5-6	
application configuration of the session filter,	constraints	
8-7	ACL evaluation order, 5-12	
authorization (checkACL), 8-5	statistics in troubleshooting, D-3	
check privilege API, 8-19	switching	
deployment, 8-6	application users	
domain configuration, 8-8	in current application session, 3-8	
automatic, 8-10	SYS GET ACLIDS function	
manual, 8-9	See ORA_GET_ACLIDS function	
prerequisites, 8-8	system-constraining ACL	
namespace APIs, 8-15	about, <i>4-12</i>	
namespace operations, 8-4	definition, 4-12	
namoopaoo operations, 🗸 🛪	····	



I	buman recourses administration of ampleuse information
tables	human resources administration of employee information
dota cocurity policy	implementation overview, 1-14
data security policy	Java implementation, 6-15
enabling, 11-48	authorizing with middle-tier API, 6-16
removing from, 11-47	main method, 6-18
master-detail tables, Real Application	performing cleanup operations, 6-18
Security policies	running a query on the database, 6-17
about, 5-12	setting up connection, 6-16
creating for, 5-13	setting up session, 6-16
time-out values	user sessions, 3-1
session	See also application sessions
IDs, setting for, 11-18	USER_XDS_ACL_REFRESH view, 9-38
TO_ACLID function, 10-6	USER_XDS_ACL_REFSTAT view, 9-38
trace files	USER_XDS_LATEST_ACL_REFSTAT view,
acl, <i>D-8</i>	9-39
application roles, D-6	USER_XS_ACES view, 9-19
application sessions, D-4	USER_XS_ACL_PARAMETERS view, 9-26
application users, <i>D-6</i>	USER_XS_ACLS view, 9-17
data security, <i>D-8</i>	USER_XS_COLUMN_CONSTRAINTS view,
policy predicate errors, 5-7	9-28
Real Application Security components, D-3	USER_XS_IMPLIED_PRIVILEGES view, 9-12
security classes, D-7	USER_XS_INHERITED_REALMS view, 9-24
tracing	USER_XS_PASSWORD_LIMITS view, 9-8
event and in-memory, D-3	USER_XS_POLICIES view, 9-21
traditional security model	USER XS PRIVILEGES view, 9-11
manging application users	USER_XS_REALM_CONSTRAINTS view, 9-22
disadvantages of, 1-2	USER_XS_SECURITY_CLASS_DEP view, 9-15
troubleshooting	USER_XS_SECURITY_CLASSES view, 9-14
acl, <i>D-8</i>	USER_XS_USERS view, 9-7
application principals, <i>D-6</i>	users, 2-2
application sessions, <i>D-4</i>	See also application users
data security, D-8	using
event-based tracing	constraining application privilege, 4-14
about, <i>D-3</i>	effective dates with application roles, 2-16
components, D-3	multilevel authentication, 4-12
exception dumps, <i>D</i> -9	ORA_CHECK_ACL SQL operator, 4-12
exception state dumps, <i>D-2</i>	SQL functions
·	to determine the invoking application
in-memory tracing, <i>D-3</i> Real Application Security diagnostics, <i>D-1</i>	user, 5-25
,,	XS_DIAG.VALIDATE_PRINCIPAL function,
security classes, <i>D-7</i>	2-12, 2-15
session statistics, <i>D-10</i>	2 12, 2 13
statistics, D-3	
using the ORA_CHECK_ACL function, D-2	V
using the ORA_GET_ACLIDS function, <i>D-2</i>	141/2 272221112 1721717171
using validation APIs, <i>D-1</i>	V\$XS_SESSION_NS_ATTRIBUTES view, 9-40
	V\$XS_SESSION_ROLES view, 9-40
U	validating
	ACLs, 4-10, 11-54
use case scenario example policy	application roles, 2-15
human resources administration of employee	application users, 2-12
information, 1-12	data security policy, 5-2, 11-55
component requirements, 1-14	namespaces, 11-56
description and security requirements,	principals, 11-52
1-13	security classes, 4-5, 11-53



validating <i>(continued)</i> workspace objects, <i>11-57</i>	views (continued) USER_XS_ACES, 9-19
views, 9-1  ALL_XDS_ACL_REFRESH, 9-34  ALL_XDS_ACL_REFSTAT, 9-34  ALL_XDS_LATEST_ACL_REFSTAT, 9-35  ALL_XS_ACES, 9-19  ALL_XS_ACL_PARAMETERS, 9-27  ALL_XS_ACLS, 9-17  ALL_XS_APPLIED_POLICIES, 9-29  ALL_XS_COLUMN_CONSTRAINTS, 9-28  ALL_XS_IMPLIED_PRIVILEGES, 9-13  ALL_XS_INHERITED_REALMS, 9-25  ALL_XS_POLICIES, 9-21  ALL_XS_PRIVILEGES, 9-11  ALL_XS_REALM_CONSTRAINTS, 9-23  ALL_XS_SECURITY_CLASS_DEP, 9-16  ALL_XS_SECURITY_CLASSES, 9-15	USER_XS_ACL_PARAMETERS, 9-26 USER_XS_ACLS, 9-17 USER_XS_COLUMN_CONSTRAINTS, 9-28 USER_XS_IMPLIED_PRIVILEGES, 9-12 USER_XS_INHERITED_REALMS, 9-24 USER_XS_PASSWORD_LIMITS, 9-8 USER_XS_POLICIES, 9-21 USER_XS_PRIVILEGES, 9-11 USER_XS_REALM_CONSTRAINTS, 9-22 USER_XS_SECURITY_CLASS_DEP, 9-15 USER_XS_SECURITY_CLASSES, 9-14 USER_XS_USERS, 9-7 V\$XS_SESSION_NS_ATTRIBUTES, 9-40 V\$XS_SESSION_ROLES, 9-40
DBA_XDS_ACL_REFRESH, 9-36	
DBA_XDS_ACL_REFSTAT, 9-36 DBA_XDS_LATEST_ACL_REFSTAT, 9-37 DBA_XS_ACES, 9-18 DBA_XS_ACL_PARAMETERS, 9-25	when element specifies a predicate for detail table, 5-13
DBA_XS_ACLS, 9-16	X
DBA_XS_ACLS, 9-16  DBA_XS_ACTIVE_SESSIONS, 9-31  DBA_XS_APPLIED_POLICIES, 9-28	XS_ACL PL/SQL package
DBA_XS_COLUMN_CONSTRAINTS, 9-27 DBA_XS_DYNAMIC_ROLES, 9-9 DBA_XS_EXTERNAL_PRINCIPALS, 9-5 DBA_XS_IMPLIED_PRIVILEGES, 9-12 DBA_XS_INHERITED_REALMS, 9-24 DBA_XS_MODIFIED_POLICIES, 9-30 DBA_XS_NS_TEMPLATE_ATTRIBUTES, 9-33 DBA_XS_NS_TEMPLATES, 9-32	about, 11-23 ADD_ACL_PARAMETER, 11-29 APPEND_ACES, 4-10, 11-26 constants, 11-24 CREATE_ACL, 11-25 DELETE_ACL, 11-31 object types, constructor functions, 11-24 REMOVE_ACES, 4-10, 11-27 REMOVE_ACL_PARAMETERS, 11-30
DBA_XS_OBJECTS, 9-4 DBA_XS_POLICIES, 9-20	security model, 11-23 SET_DESCRIPTION, 11-31
DBA_XS_PRINCIPALS, 9-5 DBA_XS_PRIVILEGE_GRANTS, 9-13 DBA_XS_PRIVILEGES, 9-10	SET_PARENT_ACL, 4-10, 4-13, 4-14, 11-28 SET_SECURITY_CLASS, 4-10, 11-28 XS_ADMIN_UTIL PL/SQL package about, 11-32
DBA_XS_PROXY_ROLES, 9-9 DBA_XS_REALM_CONSTRAINTS, 9-21 DBA_XS_ROLE_GRANTS, 9-10 DBA_XS_ROLES, 9-8 DBA_XS_ROLES, 9-8	constants, 11-32 GRANT_SYSTEM_PRIVILEGE, 11-33 object types, 11-33 REVOKE_SYSTEM_PRIVILEGE, 11-34
DBA_XS_SECURITY_CLASS_DEP, 9-15 DBA_XS_SECURITY_CLASSES, 9-14 DBA_XS_SESSION_NS_ATTRIBUTES, 9-32	security model, 11-32  XS_DATA_SECURITY PL/SQL package about, 11-35  ADD_COLUMN_CONSTRAINTS Procedure,
DBA_XS_SESSION_ROLES, 9-31 DBA_XS_SESSIONS, 9-30 DBA_XS_USERS, 9-6	11-41 APPEND_REALM_CONSTRAINTS Procedure, 11-39
privileges in data security documents, 5-21 USER_XDS_ACL_REFRESH, 9-38 USER_XDS_ACL_REFSTAT, 9-38 USER_XDS_LATEST_ACL_REFSTAT, 9-39	APPLY_OBJECT_POLICY, 5-11 APPLY_OBJECT_POLICY Procedure, 11-48 CREATE_ACL_PARAMETER Procedure, 11-42

XS_DATA_SECURITY PL/SQL package (continued)	XS_NAMESPACE PL/SQL package (continued)
CREATE_POLICY Procedure, 11-38	SET_DESCRIPTION Procedure, 11-62
DELETE_ACL_PARAMETER Procedure,	SET HANDLER Procedure, 11-61
11-43	XS_PRINCIPAL PL/SQL package
DELETE_POLICY Procedure, 11-44	about, <i>11-63</i>
DISABLE_OBJECT_POLICY Procedure,	ADD_PROXY_TO_DBUSER Procedure,
11-46	11-73
ENABLE_OBJECT_POLICY	ADD_PROXY_USER, 2-10
affect on database tables, 5-11	ADD_PROXY_USER Procedure, 11-71
ENABLE_OBJECT_POLICY Procedure,	constants, 11-64
11-45	CREATE_DYNAMIC_ROLE, 2-14
object types, constructor functions, 11-36	CREATE_DYNAMIC_ROLE Procedure,
REMOVE_COLUMN_CONSTRAINTS	<u></u>
Procedure, <u>11-42</u>	CREATE_ROLE, 2-14
REMOVE_OBJECT_POLICY Procedure,	CREATE_ROLE Procedure, 11-67
11-47	CREATE_USER, 2-4, 2-16
REMOVE REALM CONSTRAINTS	CREATE_USER Procedure, 11-66
Procedure, 11-40	DELETE PRINCIPAL Procedure, 11-85
security model, 11-35	ENABLE_BY_DEFAULT Procedure, 11-76
SET_DESCRIPTION Procedure, 11-44	ENABLE_BI_DEFAULT Procedure, 11-70  ENABLE_ROLES_BY_DEFAULT Procedure,
<del>-</del>	11-77
XS_DATA_SECURITY_UTIL PL/SQL package	
about, 11-49	GRANT_ROLES, 2-18
ALTER_STATIC_ACL_REFRESH Procedure,	GRANT_ROLES Procedure, 11-69
11-51	object types, constructor functions, 11-64
constants, 11-49	REMOVE_PROXY_FROM_DBUSER
SCHEDULE_STATIC_ACL_REFRESH	Procedure, 11-74
Procedure, 11-50	REMOVE_PROXY_USERS Procedure,
security model, 11-49	11-73
XS_DIAG PL/SQL package	REVOKE_ROLES Procedure, 11-71
about, <i>11-51</i>	security model, 11-63
security model, 11-52	SET_ACL Procedure, 11-79
VALIDATE_ACL, 4-10	SET_DESCRIPTION Procedure, 11-85
VALIDATE_ACL Function, 11-54	SET_DYNAMIC_ROLE_DURATION
VALIDATE_DATA_SECURITY, 5-2	Procedure, 11-75
VALIDATE_DATA_SECURITY Function,	SET_DYNAMIC_ROLE_SCOPE Procedure,
11-55	11-76
VALIDATE_PRINCIPAL, 2-12, 2-15	SET_EFFECTIVE_DATES Procedure, 11-74
VALIDATE_PRINCIPAL Function, 11-52	SET_GUID Procedure, 11-78
VALIDATE_SECURITY_CLASS, 4-5	SET_PASSWORD, 2-4
VALIDATE_SECURITY_CLASS Function,	SET_PASSWORD Procedure, 11-82
11-53	SET_PROFILE, 2-4
VALIDATE_WORKSPACE Function, 11-57	SET PROFILE Procedure, 11-80
XS_DIAG PL/SQL PL/SQL package	SET_USER_SCHEMA Procedure, 11-78
VALIDATE_NAMESPACE_TEMPLATE	SET_USER_STATUS Procedure, 11-81
Function, <i>11-56</i>	SET VERIFIER, 2-5
XS_NAMESPACE PL/SQL package	SET_VERIFIER Procedure, 11-83
about, <u>11-57</u>	XS_SECURITY_CLASS PL/SQL package
ADD ATTRIBUTES Procedure, 11-60	about, <u>11-86</u>
constants, 11-58	ADD_IMPLIED_PRIVILEGES, 4-2, 4-5
CREATE_TEMPLATE, 3-17	ADD_IMPLIED_PRIVILEGES Procedure,
CREATE_TEMPLATE Procedure, 11-59	11-91
DELETE_TEMPLATE, 11-62	ADD PARENTS, 4-5
object types, constructor functions, <i>11-58</i>	ADD_PARENTS Procedure, 11-88
REMOVE_ATTRIBUTES Procedure, 11-61	ADD_PRIVILEGES, 4-1, 4-5
security model, 11-58	ADD_PRIVILEGES Procedure, 11-89
<del></del>	

XS_SECURITY_CLASS PL/SQL package (continued)	XS_SECURITY_CLASS PL/SQL package (continued)
CREATE_SECURITY_CLASS Procedure,	REMOVE_PRIVILEGES Procedure, 11-90
<i>11-87</i>	security model, 11-86
DELETE_SECURITY_CLASS, 4-5	SET_DESCRIPTION, 4-5
DELETE_SECURITY_CLASS Procedure,	SET_DESCRIPTION Procedure, 11-93
11-93	XS_SYS_CONTEXT function, 10-2
REMOVE_IMPLIED_PRIVILEGES, 4-5	XSSessionManager
REMOVE_IMPLIED_PRIVILEGES	clearCache, 6-4
Procedure, 11-92	createAnonymousSession, 6-5
REMOVE_PARENTS, 4-5	createSession, 6-5
REMOVE_PARENTS Procedure, 11-89	getLowWaterMark, 6-4
REMOVE PRIVILEGES, 4-5	-

