

# Oracle® Database

## Workspace Manager Developer's Guide



19c  
E94792-03  
June 2023

The Oracle logo, consisting of a solid red square with the word "ORACLE" in white, uppercase, sans-serif font centered within it.

ORACLE®

Oracle Database Workspace Manager Developer's Guide, 19c

E94792-03

Copyright © 2000, 2023, Oracle and/or its affiliates.

Primary Author: Lavanya Jayapalan

Contributors: Chuck Murray, Bill Beauregard, Ben Speckhard

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	xvi
Documentation Accessibility	xvi
Related Documents	xvi
Conventions	xvii

## Part I Conceptual and Usage Information

---

### 1 Introduction to Workspace Manager

---

1.1	Workspace Manager Overview	1-3
1.1.1	Workspace Hierarchy	1-4
1.1.2	Using Savepoints	1-5
1.1.2.1	Design Issue: Savepoint or Child Workspace?	1-6
1.1.3	Merging and Rolling Back Workspace Changes	1-7
1.1.4	Resolving Conflicts Before a Merge or Refresh Operation	1-7
1.1.5	Freezing and Unfreezing Workspaces	1-8
1.1.6	Removing Workspaces	1-9
1.1.7	Using Workspace Manager Events	1-9
1.1.8	Autocommitting of Workspace Manager Operations	1-9
1.1.9	Continually Refreshed Workspaces	1-10
1.1.10	Multiparent Workspaces	1-11
1.1.11	Infrastructure for Version-Enabling of Tables	1-12
1.1.12	Creation of Row Versions and Historical Copies	1-13
1.1.13	Workspace Manager Schema, Metadata, and Package	1-15
1.2	Session Context Information for Workspace Manager	1-15
1.3	Lock Management with Workspace Manager	1-16
1.3.1	Exclusive Locking and Row Versions	1-18
1.3.2	Locks Taken for Workspace Manager Operations	1-18
1.4	Privilege Management with Workspace Manager	1-24
1.5	System Parameters for Workspace Manager	1-25
1.6	Import and Export Considerations	1-31

1.7	Bulk Loading into Version-Enabled Tables	1-33
1.8	DDL Operations Related to Version-Enabled Tables	1-34
1.9	Constraint Support with Workspace Manager	1-36
1.9.1	Referential Integrity Support	1-36
1.9.1.1	Locking with DML Operations on Tables with Referential Integrity Constraints	1-38
1.9.2	Unique Constraints	1-39
1.9.3	SET NULL Constraints	1-40
1.10	Triggers on Version-Enabled Tables	1-40
1.11	Virtual Private Database Considerations	1-40
1.12	Support for Table Synonyms	1-40
1.13	Materialized View Support	1-41
1.14	Spatial and Graph Topology Support	1-41
1.14.1	Locking Considerations with Topologies	1-42
1.14.2	Additional Considerations with Topologies	1-42
1.15	Workspace Manager Reserved Words and Characters	1-42
1.16	DBMS_WM Subprogram Categories	1-43
1.16.1	Table Management Subprograms	1-43
1.16.2	Workspace Management Subprograms	1-44
1.16.3	Savepoint Management Subprograms	1-46
1.16.4	Privilege Management Subprograms	1-46
1.16.5	Lock Management Subprograms	1-46
1.16.6	Conflict Management Subprograms	1-47
1.16.7	Bulk Load Support Subprograms	1-47
1.17	Simplified Examples Using Workspace Manager	1-48
1.17.1	Example: Marketing Budget Options	1-48
1.17.2	Example: Warehouse Expansion Options	1-52

## 2 Workspace Manager Events

---

2.1	List of Workspace Manager Events	2-2
2.2	Event Parameters	2-2
2.3	ALLOW_CAPTURE_EVENTS System Parameter	2-3
2.4	AQ Operations and Workspace Manager Events	2-4
2.4.1	Workspace Manager Event Queue Administration	2-4
2.4.2	Privileges and Access Control for Queues	2-4
2.4.3	Rule-Based Subscription	2-5
2.4.4	Listening for Events	2-6
2.4.5	Asynchronous Notification	2-6

## 3 Workspace Manager Valid Time Support

---

3.1	Valid Time Support: Introduction and Example	3-2
3.2	WM_PERIOD Data Type	3-3
3.3	Valid Time Constants	3-4
3.4	API Features for Valid Time Support	3-4
3.5	Operators for Valid Time Support	3-5
3.5.1	WM_CONTAINS	3-6
3.5.2	WM_EQUALS	3-7
3.5.3	WM_GREATERTHAN	3-8
3.5.4	WM_INTERSECTION	3-8
3.5.5	WM_LDIFF	3-9
3.5.6	WM_LESSTHAN	3-10
3.5.7	WM_MEETS	3-11
3.5.8	WM_OVERLAPS	3-12
3.5.9	WM_RDIFF	3-12
3.6	Queries and DML Operations with Valid Time Support	3-13
3.6.1	Queries	3-13
3.6.2	Data Manipulation (DML) Operations	3-14
3.6.2.1	Update Operations	3-14
3.6.2.2	Insert Operations	3-15
3.7	Constraint Management for Valid Time Support	3-16
3.7.1	Referential Integrity Constraints	3-16
3.7.2	Unique Constraints	3-16
3.8	Locking with Valid Time Support	3-17
3.9	Static Data Dictionary Views Affected by Valid Time Support	3-17
3.9.1	xxx_CONF Views and Valid Time Support	3-17
3.9.2	xxx_DIFF Views and Valid Time Support	3-17
3.9.3	xxx_HIST Views and Valid Time Support	3-18
3.9.4	xxx_LOCK Views and Valid Time Support	3-18
3.9.5	xxx_MW Views and Valid Time Support	3-18
3.10	Adding Valid Time Support to an Existing Table	3-18
3.11	Merging and Refreshing Workspaces for Tables with Valid Time Support	3-19

## Part II Reference Information

---

### 4 DBMS\_WM Package: Reference

---

4.1	Add_Topo_Geometry_Layer	4-4
4.2	AddAsParentWorkspace	4-5
4.3	AddUserDefinedHint	4-7

4.4	AlterSavepoint	4-8
4.5	AlterVersionedTable	4-9
4.6	AlterWorkspace	4-12
4.7	BeginBulkLoading	4-13
4.8	BeginDDL	4-15
4.9	BeginResolve	4-16
4.10	ChangeWorkspaceType	4-17
4.11	CommitBulkLoading	4-18
4.12	CommitDDL	4-21
4.13	CommitResolve	4-22
4.14	CompressWorkspace	4-23
4.15	CompressWorkspaceTree	4-27
4.16	CopyForUpdate	4-31
4.17	CopyWorkspace	4-32
4.18	CreateSavepoint	4-33
4.19	CreateWorkspace	4-34
4.20	Delete_Topo_Geometry_Layer	4-36
4.21	DeleteSavepoint	4-37
4.22	DisableVersioning	4-39
4.23	EnableVersioning	4-42
4.24	Export	4-45
4.25	Export_Schemas	4-48
4.26	FindRICSet	4-51
4.27	FreezeWorkspace	4-52
4.28	GetBulkLoadVersion	4-54
4.29	GetConflictWorkspace	4-56
4.30	GetDiffVersions	4-56
4.31	GetLockMode	4-57
4.32	GetMultiWorkspaces	4-57
4.33	GetOpContext	4-58
4.34	GetOriginalDDL	4-59
4.35	GetPhysicalTableName	4-60
4.36	GetPrivs	4-61
4.37	GetSessionInfo	4-61
4.38	GetSystemParameter	4-63
4.39	GetValidFrom	4-63
4.40	GetValidTill	4-64
4.41	GetVersion	4-65
4.42	GetWMMetadataSpace	4-65
4.43	GetWorkspace	4-66
4.44	GotoDate	4-66

4.45	GotoSavepoint	4-68
4.46	GotoWorkspace	4-69
4.47	GrantGraphPriv	4-69
4.48	GrantPrivsOnPolicy	4-71
4.49	GrantSystemPriv	4-72
4.50	GrantWorkspacePriv	4-73
4.51	Import	4-75
4.52	Import_Schemas	4-78
4.53	Initialize_After_Import	4-80
4.54	IsWorkspaceOccupied	4-81
4.55	LockRows	4-82
4.56	MergeTable	4-84
4.57	MergeWorkspace	4-86
4.58	Move_Proc	4-88
4.59	PurgeTable	4-89
4.60	RecoverAllMigratingTables	4-90
4.61	RecoverFromDroppedUser	4-91
4.62	RecoverMigratingTable	4-93
4.63	RefreshTable	4-94
4.64	RefreshWorkspace	4-95
4.65	RemoveAsParentWorkspace	4-96
4.66	RemoveDeferredWorkspaces	4-98
4.67	RemoveUserDefinedHint	4-99
4.68	RemoveWorkspace	4-99
4.69	RemoveWorkspaceTree	4-100
4.70	RenameSavepoint	4-102
4.71	RenameWorkspace	4-102
4.72	ResolveConflicts	4-103
4.73	RevokeGraphPriv	4-105
4.74	RevokeSystemPriv	4-107
4.75	RevokeWorkspacePriv	4-108
4.76	RollbackBulkLoading	4-109
4.77	RollbackDDL	4-110
4.78	RollbackResolve	4-111
4.79	RollbackTable	4-112
4.80	RollbackToSP	4-113
4.81	RollbackWorkspace	4-114
4.82	SetCaptureEvent	4-115
4.83	SetCompressWorkspace	4-117
4.84	SetConflictWorkspace	4-118
4.85	SetDiffVersions	4-119

4.86	SetLockingOFF	4-120
4.87	SetLockingON	4-121
4.88	SetMultiWorkspaces	4-122
4.89	SetSystemParameter	4-123
4.90	SetTriggerEvents	4-124
4.91	SetValidTime	4-126
4.92	SetValidTimeFilterOFF	4-126
4.93	SetValidTimeFilterON	4-127
4.94	SetWMValidUpdateModeOFF	4-128
4.95	SetWMValidUpdateModeON	4-128
4.96	SetWoOverwriteOFF	4-129
4.97	SetWoOverwriteON	4-130
4.98	SetWorkspaceLockModeOFF	4-130
4.99	SetWorkspaceLockModeON	4-131
4.100	UnfreezeWorkspace	4-133
4.101	UnlockRows	4-134
4.102	UseDefaultValuesForNulls	4-136

## 5 Workspace Manager Static Data Dictionary Views

---

5.1	ALL_MP_GRAPH_WORKSPACES	5-3
5.2	ALL_MP_PARENT_WORKSPACES	5-4
5.3	ALL_REMOVED_WORKSPACES	5-4
5.4	ALL_VERSION_HVIEW	5-5
5.5	ALL_WM_CONS_COLUMNS	5-6
5.6	ALL_WM_CONSTRAINT_VIOLATIONS	5-6
5.7	ALL_WM_CONSTRAINTS	5-7
5.8	ALL_WM_IND_COLUMNS	5-7
5.9	ALL_WM_IND_EXPRESSIONS	5-8
5.10	ALL_WM_LOCKED_TABLES	5-8
5.11	ALL_WM_MODIFIED_TABLES	5-9
5.12	ALL_WM_POLICIES	5-9
5.13	ALL_WM_RIC_INFO	5-9
5.14	ALL_WM_TAB_TRIGGERS	5-10
5.15	ALL_WM_VERSIONED_TABLES	5-12
5.16	ALL_WM_VT_ERRORS	5-13
5.17	ALL_WORKSPACE_PRIVS	5-14
5.18	ALL_WORKSPACE_SAVEPOINTS	5-14
5.19	ALL_WORKSPACES	5-15
5.20	DBA_REMOVED_WORKSPACES	5-17
5.21	DBA_WM_SYS_PRIVS	5-17



5.22	DBA_WM_VERSIONED_TABLES	5-17
5.23	DBA_WM_VT_ERRORS	5-17
5.24	DBA_WORKSPACE_PRIVS	5-18
5.25	DBA_WORKSPACE_SAVEPOINTS	5-18
5.26	DBA_WORKSPACE_SESSIONS	5-18
5.27	DBA_WORKSPACES	5-18
5.28	ROLE_WM_PRIVS	5-20
5.29	USER_MP_GRAPH_WORKSPACES	5-20
5.30	USER_MP_PARENT_WORKSPACES	5-20
5.31	USER_REMOVED_WORKSPACES	5-20
5.32	USER_WM_CONS_COLUMNS	5-21
5.33	USER_WM_CONSTRAINTS	5-21
5.34	USER_WM_IND_COLUMNS	5-21
5.35	USER_WM_IND_EXPRESSIONS	5-21
5.36	USER_WM_LOCKED_TABLES	5-21
5.37	USER_WM_MODIFIED_TABLES	5-21
5.38	USER_WM_POLICIES	5-21
5.39	USER_WM_PRIVS	5-22
5.40	USER_WM_RIC_INFO	5-22
5.41	USER_WM_TAB_TRIGGERS	5-22
5.42	USER_WM_VERSIONED_TABLES	5-22
5.43	USER_WM_VT_ERRORS	5-23
5.44	USER_WORKSPACE_PRIVS	5-23
5.45	USER_WORKSPACE_SAVEPOINTS	5-23
5.46	USER_WORKSPACES	5-23
5.47	WM_COMPRESS_BATCH_SIZES	5-23
5.48	WM_COMPRESSIBLE_TABLES	5-24
5.49	WM_EVENTS_INFO	5-24
5.50	WM_INSTALLATION	5-24
5.51	xxx_CONF Views	5-25
5.52	xxx_DIFF Views	5-26
5.53	xxx_HIST Views	5-27
5.54	xxx_LOCK Views	5-28
5.55	xxx_MW Views	5-29

## Part III Supplementary Information

---

### A Installing Workspace Manager with Custom Databases

---

## B Workspace Manager Error Messages

---

Glossary

---

Index

---

## List of Figures

---

1-1	Workspace Tree	1-5
1-2	Savepoints	1-6
1-3	Multiparent Workspace in a Workspace Hierarchy	1-11

## List of Tables

---

1-1	Freeze Results of Procedures	1-9
1-2	Name Length Guidelines for Workspace Manager	1-12
1-3	Workspace Manager Lock Modes and Data Modification Ability	1-17
1-4	Operations and Incompatible Freeze Modes for Workspace Types	1-19
1-5	Workspace Manager Privileges	1-24
1-6	Workspace Manager System Parameters	1-26
1-7	Workspace Manager Reserved Words and Characters	1-43
1-8	Table Management Subprograms	1-44
1-9	Workspace Management Subprograms	1-44
1-10	Savepoint Management Subprograms	1-46
1-11	Privilege Management Subprograms	1-46
1-12	Lock Management Subprograms	1-47
1-13	Conflict Management Subprograms	1-47
1-14	Bulk Loading Support Subprograms	1-48
2-1	Workspace Manager Events	2-2
2-2	Workspace Manager Event Parameters	2-2
2-3	AQ Administrative Views for Workspace Manager	2-4
3-1	Constants for Valid Time Support	3-4
3-2	API Features for Valid Time Support	3-4
3-3	validTill Values and Intersection Result from Merge or Refresh Operation	3-19
4-1	Add_Topo_Geometry_Layer Procedure Parameters	4-5
4-2	AddAsParentWorkspace Procedure Parameters	4-6
4-3	AddUserDefinedHint Procedure Parameters	4-7
4-4	AlterSavepoint Procedure Parameters	4-8
4-5	AlterVersionedTable Procedure Parameters	4-9
4-6	AlterWorkspace Procedure Parameters	4-12
4-7	BeginBulkLoading Procedure Parameters	4-13
4-8	BeginDDL Procedure Parameters	4-15
4-9	BeginResolve Procedure Parameters	4-16
4-10	ChangeWorkspaceType Procedure Parameters	4-17
4-11	CommitBulkLoading Procedure Parameters	4-18
4-12	CommitDDL Procedure Parameters	4-21
4-13	CommitResolve Procedure Parameters	4-23
4-14	CompressWorkspace Procedure Parameters	4-24
4-15	CompressWorkspaceTree Procedure Parameters	4-28

4-16	CopyForUpdate Procedure Parameters	4-31
4-17	CopyWorkspace Procedure Parameters	4-32
4-18	CreateSavepoint Procedure Parameters	4-33
4-19	CreateWorkspace Procedure Parameters	4-34
4-20	Delete_Topo_Geometry_Layer Procedure Parameters	4-36
4-21	DeleteSavepoint Procedure Parameters	4-37
4-22	DisableVersioning Procedure Parameters	4-40
4-23	EnableVersioning Procedure Parameters	4-42
4-24	Export Procedure Parameters	4-45
4-25	Export_Schemas Procedure Parameters	4-48
4-26	Supported Cloud Regions for Oracle Workspace Manager	4-50
4-27	FindRICSet Procedure Parameters	4-51
4-28	FreezeWorkspace Procedure Parameters	4-53
4-29	GetBulkLoadVersion Function Parameters	4-55
4-30	GetOriginalDDL Procedure Parameters	4-59
4-31	GetPhysicalTableName Function Parameters	4-60
4-32	GetPrivs Function Parameters	4-61
4-33	GetSessionInfo Procedure Parameters	4-62
4-34	GetSystemParameter Procedure Parameters	4-63
4-35	GotoDate Procedure Parameters	4-66
4-36	GotoSavepoint Procedure Parameters	4-68
4-37	GotoWorkspace Procedure Parameters	4-69
4-38	GrantGraphPriv Procedure Parameters	4-70
4-39	GrantPrivsOnPolicy Procedure Parameters	4-71
4-40	GrantSystemPriv Procedure Parameters	4-72
4-41	GrantWorkspacePriv Procedure Parameters	4-73
4-42	Import Procedure Parameters	4-75
4-43	Import_Schemas Procedure Parameters	4-78
4-44	Supported Cloud Regions for Oracle Workspace Manager	4-80
4-45	Initialize_After_Import Procedure Parameters	4-81
4-46	IsWorkspaceOccupied Function Parameters	4-82
4-47	LockRows Procedure Parameters	4-83
4-48	MergeTable Procedure Parameters	4-84
4-49	MergeWorkspace Procedure Parameters	4-86
4-50	Move_Proc Procedure Parameters	4-88
4-51	PurgeTable Procedure Parameters	4-89
4-52	RecoverAllMigratingTables Procedure Parameters	4-91

4-53	RecoverFromDroppedUser Procedure Parameters	4-92
4-54	RecoverMigratingTable Procedure Parameters	4-93
4-55	RefreshTable Procedure Parameters	4-94
4-56	RefreshWorkspace Procedure Parameters	4-95
4-57	RemoveAsParentWorkspace Procedure Parameters	4-97
4-58	RemoveWorkspace Procedure Parameters	4-98
4-59	RemoveUserDefinedHint Procedure Parameters	4-99
4-60	RemoveWorkspace Procedure Parameters	4-100
4-61	RemoveWorkspaceTree Procedure Parameters	4-101
4-62	RenameSavepoint Procedure Parameters	4-102
4-63	RenameWorkspace Procedure Parameters	4-103
4-64	ResolveConflicts Procedure Parameters	4-103
4-65	RevokeGraphPriv Procedure Parameters	4-106
4-66	RevokeSystemPriv Procedure Parameters	4-107
4-67	RevokeWorkspacePriv Procedure Parameters	4-108
4-68	RollbackBulkLoading Procedure Parameters	4-109
4-69	RollbackDDL Procedure Parameters	4-110
4-70	RollbackResolve Procedure Parameters	4-111
4-71	RollbackTable Procedure Parameters	4-112
4-72	RollbackToSP Procedure Parameters	4-113
4-73	RollbackWorkspace Procedure Parameters	4-114
4-74	SetCaptureEvent Procedure Parameters	4-116
4-75	SetCompressWorkspace Procedure Parameters	4-117
4-76	SetConflictWorkspace Procedure Parameters	4-118
4-77	SetDiffVersions Procedure Parameters	4-119
4-78	SetLockingON Procedure Parameters	4-121
4-79	SetMultiWorkspaces Procedure Parameters	4-122
4-80	SetSystemParameter Procedure Parameters	4-123
4-81	SetTriggerEvents Procedure Parameters	4-125
4-82	SetValidTime Procedure Parameters	4-126
4-83	SetValidTimeFilterON Procedure Parameters	4-127
4-84	SetWorkspaceLockModeOFF Procedure Parameters	4-131
4-85	SetWorkspaceLockModeON Procedure Parameters	4-132
4-86	UnfreezeWorkspace Procedure Parameters	4-134
4-87	UnlockRows Procedure Parameters	4-134
4-88	UseDefaultValuesForNulls Procedure Parameters	4-136
5-1	Columns in the xxx_CONF Views	5-25

5-2	Columns in the xxx_DIFF Views	5-26
5-3	Columns in the xxx_HIST Views	5-28
5-4	Columns in the xxx_LOCK Views	5-29
5-5	Columns in the xxx_MW Views	5-29

# Preface

*Oracle Database Workspace Manager Developer's Guide* describes Oracle Workspace Manager, often referred to as Workspace Manager, which enables applications to create workspaces and group different versions of table row values in different workspaces.

- [Audience](#)
- [Documentation Accessibility](#)
- [Related Documents](#)
- [Conventions](#)

## Audience

*Oracle Database Workspace Manager Developer's Guide* is intended for application designers and developers. It is assumed that you have some experience programming in PL/SQL.

## Documentation Accessibility

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

### **Access to Oracle Support**

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

## Related Documents

For more information about using this product in a development environment, see the following documents:

- *Oracle Call Interface Programmer's Guide*
- *Oracle Database Concepts*
- *Oracle Database PL/SQL Language Reference*



## Conventions

The following text conventions are used in this document:

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

# Part I

## Conceptual and Usage Information

This document has three parts:

- Part I provides conceptual and usage information about Workspace Manager.
- [Reference Information](#) provides reference information about the Workspace Manager PL/SQL API (DBMS\_WM package) and static data dictionary views.
- [Supplementary Information](#) provides supplementary information (appendixes and a glossary).

Part I is organized for efficient learning about Workspace Manager. It covers basic concepts and techniques first, and proceeds to more advanced material (such as Workspace Manager events and valid time support). Part I contains the following chapters:

- [Introduction to Workspace Manager](#)  
Oracle Workspace Manager, often referred to as Workspace Manager, provides an infrastructure that enables applications to create workspaces and group different versions of table row values in different workspaces.
- [Workspace Manager Events](#)  
Certain applications may be interested in knowing what Workspace Manager operations are being performed and may want to take some actions based on that. Several types of Workspace Manager operations can be captured as events.
- [Workspace Manager Valid Time Support](#)  
This chapter describes the support for valid time, also known as effective dating, with version-enabled tables.

# 1

## Introduction to Workspace Manager

Oracle Workspace Manager, often referred to as Workspace Manager, provides an infrastructure that enables applications to create workspaces and group different versions of table row values in different workspaces.

Users are permitted to create new versions of data to update, while maintaining a copy of the old data. The ongoing results of the activity are stored persistently, assuring concurrency and consistency.

Applications that can benefit from Workspace Manager typically do one or more of the following operations:

- Manage a collection of updates and insertions as a unit before incorporating them into production data

You can review changes and roll back undesirable ones before making the changes public. Until you make the changes public, they are invisible to other users of the database, who will access only the regular production data. You can organize the changes in a simple set of workspaces or in a complex workspace hierarchy. A typical example might be a life sciences application in which Workspace Manager supports the discovery and quality assurance (QA) processes by managing a collection of updates before they are merged with the production data.

- Support a collaborative development effort

A team can share access to a collection of updates and insertions for a collaborative project. Workspace privileges control access to a workspace and its operations, and you can restrict workspace access to single-writer, read-only, or no access. Workspace locks prevent update conflicts between projects in separate workspaces. A typical example might be an application to design an engineering project, in which multiple subprojects are concurrently developed in separate workspaces.

- Use a common data set to create multiple scenarios for *what-if* analyses or multiple editions of data for publication

You can organize changes in workspaces to view them in the context of the whole database, but without requiring that you actually copy data between tables. Different users can make simultaneous changes to the same row, and you can detect and resolve conflicts. A typical example might be a telecommunications application in which you create multiple cell phone coverage scenarios to find the optimal design.

- Keep a history of changes to data

You can navigate workspaces and row versions to view the database as of a particular milestone or point in time. You can roll back changes to a row or table in a workspace to a milestone. A typical example might be a land information management application where Workspace Manager supports regulatory requirements by maintaining a history of all changes to land parcels.

Workspace Manager is also useful in managing *long-transaction* scenarios, where complex, long-duration database transactions can take days to complete, and multiple users must access the same database.

This chapter explains concepts and operations that you must understand to use Workspace Manager. For complete examples of Workspace Manager, see [Simplified Examples Using Workspace Manager](#). However, you may want to read the rest of this chapter first, to understand the concepts that the examples illustrate.

 **Note:**

Workspace Manager is installed by default in the Oracle seed database and any database created using the Database Configuration Assistant (DBCA). To use Workspace Manager in any other Oracle database, you must first perform the installation procedure described in [Installing Workspace Manager with Custom Databases](#).

- [Workspace Manager Overview](#)  
Workspace Manager enables you to **version-enable** one or more user tables in the database. When a table is version-enabled, all rows in the table can support multiple versions of the data.
- [Session Context Information for Workspace Manager](#)  
Users perform Workspace Manager operations within a standard Oracle session.
- [Lock Management with Workspace Manager](#)
- [Privilege Management with Workspace Manager](#)  
Workspace Manager provides a set of privileges that are separate from standard Oracle database privileges.
- [System Parameters for Workspace Manager](#)  
Workspace Manager provides a set of system parameters that allow a user with the WM\_ADMIN system privilege to enforce global Workspace Manager-specific settings for the database.
- [Import and Export Considerations](#)  
Workspace Manager supports the import and export of version-enabled tables in any of the following ways: a full database import and export, an import and export that only includes the schemas required by Workspace Manager, or a workspace-level import and export through Workspace Manager procedures.
- [Bulk Loading into Version-Enabled Tables](#)  
You can use SQL\*Loader to perform bulk loading into version-enabled tables, but you must also call some special Workspace Manager procedures, and some restrictions apply.
- [DDL Operations Related to Version-Enabled Tables](#)  
To perform DDL (data definition language) operations on a version-enabled table, you must use special Workspace Manager procedures before and after the DDL operations, and you must specify the name of a special table created by Workspace Manager.
- [Constraint Support with Workspace Manager](#)  
This section describes Workspace Manager considerations relating to the use of database constraints.
- [Triggers on Version-Enabled Tables](#)  
Version-enabled tables can have triggers defined; however, the following considerations and restrictions apply.

- [Virtual Private Database Considerations](#)  
You can use Workspace Manager in conjunction with the Oracle Virtual Private Database (VPD) technology.
- [Support for Table Synonyms](#)  
For any Workspace Manager procedure or function input parameter that calls for a table name, you can instead specify a synonym.
- [Materialized View Support](#)  
This section describes considerations for using Workspace Manager with materialized views.
- [Spatial and Graph Topology Support](#)  
This section describes special considerations and techniques for using Workspace Manager with tables in Oracle Spatial and Graph topologies,
- [Workspace Manager Reserved Words and Characters](#)  
Because Workspace Manager creates internal objects using its own naming conventions, you must avoid some words and characters in the names for certain kinds objects.
- [DBMS\\_WM Subprogram Categories](#)  
The Workspace Manager application programming interface (API) consists of PL/SQL subprograms (procedures and functions) in a single PL/SQL package named `DBMS_WM`.
- [Simplified Examples Using Workspace Manager](#)  
This topic presents two simplified examples of using Workspace Manager to try out some scenarios and select one of them.

## 1.1 Workspace Manager Overview

Workspace Manager enables you to **version-enable** one or more user tables in the database. When a table is version-enabled, all rows in the table can support multiple versions of the data.

The versioning infrastructure is not visible to the users of the database, and application SQL statements for selecting, inserting, modifying, and deleting data continue to work in the usual way with version-enabled tables, although you cannot update a primary key column value in a version-enabled table. (Workspace Manager implements these capabilities by maintaining system views and creating `INSTEAD OF` triggers, as explained in [Infrastructure for Version-Enabling of Tables](#); however, application developers and users do not need to see or interact with the views and triggers.)

After a table is version-enabled, users in a workspace automatically see the correct version of the record in which they are interested. If you no longer need a table to be version-enabled, you can disable versioning for the table.

A **workspace** is a virtual environment that one or more users can share to make changes to the data in the database. A workspace logically groups collections of new row versions from one or more version-enabled tables, and isolates these versions until they are explicitly merged with production data or discarded, thus providing maximum concurrency. Users can perform a variety of operations involving workspaces: go to, create, refresh, merge, roll back, remove, compress, alter, and other operations.

Users in a workspace always see a transactionally consistent view of the entire database; that is, they see changes made in their current workspace plus the rest of the data in the database as it existed either when the workspace was created or when the workspace was most recently refreshed with changes from the parent workspace. (Workspace hierarchy and parent and child workspaces are explained in [Workspace Hierarchy](#).)

Workspace Manager automatically detects **conflicts**, which are differences in data values resulting from changes to the same row in a workspace and its parent workspace. You must resolve conflicts before merging changes from a workspace into its parent workspace. You can use workspace locks to avoid conflicts.

**Savepoints** are points in the workspace to which row changes in version-enabled tables can be rolled back, and to which users can go to see the database as it existed at that point. Savepoints are usually created in response to a business-related milestone, such as the completion of a design phase or the end of a billing period. (For more information about savepoints, see [Using Savepoints](#).)

The **history option** enables you to timestamp changes made to all rows in a version-enabled table and to save a copy of either all changes or only the most recent changes to each row. If you keep all changes (specifying the "without overwrite" history option) when version-enabling a table, you keep a persistent history of all changes made to all row versions, and enable users to go to any point in time to view the database as it existed from the perspective of that workspace.

Workspace Manager provides a comprehensive PL/SQL API that you can add to new and existing applications to manage workspaces, savepoints, history information, privileges, access modes, and Workspace Manager locks, and to detect and resolve conflicts. You can also perform many of these operations using the Oracle Enterprise Manager graphical user interface.

Another database object created by Workspace Manager is a database-wide system table that maps row versions to workspaces. This table is not visible to users.

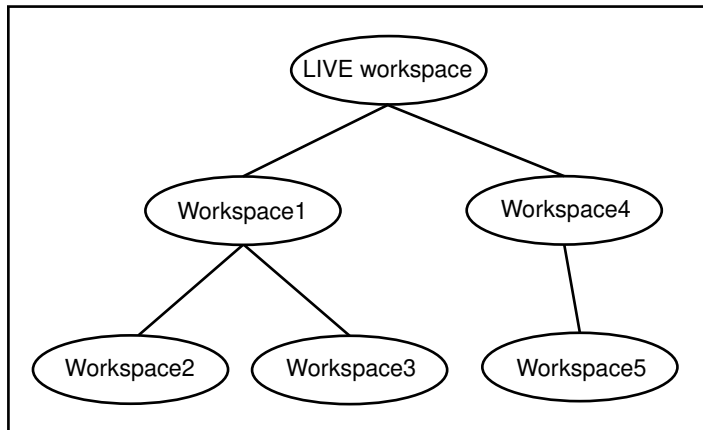
- [Workspace Hierarchy](#)
- [Using Savepoints](#)
- [Merging and Rolling Back Workspace Changes](#)
- [Resolving Conflicts Before a Merge or Refresh Operation](#)
- [Freezing and Unfreezing Workspaces](#)
- [Removing Workspaces](#)
- [Using Workspace Manager Events](#)
- [Autocommitting of Workspace Manager Operations](#)
- [Continually Refreshed Workspaces](#)
- [Multiparent Workspaces](#)
- [Infrastructure for Version-Enabling of Tables](#)
- [Creation of Row Versions and Historical Copies](#)
- [Workspace Manager Schema, Metadata, and Package](#)

### 1.1.1 Workspace Hierarchy

There can be a hierarchy of workspaces in the database. For example, a workspace can be a parent to one or more workspaces (child workspaces). By default, when a workspace is created, it is created from the topmost, or `LIVE`, database workspace. (Workspace names are case-sensitive, and the workspace name of the live database is spelled `LIVE`. The length of a workspace name must not exceed 30 characters, and the depth of the workspace hierarchy must not exceed 30 levels.) Users are included in a workspace by a [GotoWorkspace](#) operation.

Figure 1-1 shows a hierarchy of workspaces. Workspace1 and Workspace4 were formed off the LIVE database workspace; Workspace2 and Workspace3 were formed off Workspace1, and Workspace5 was formed off Workspace4. After Workspace1 was created, a user executed a [GotoWorkspace](#) operation specifying Workspace1, and then executed [CreateWorkspace](#) operations to create Workspace2 and Workspace3. A comparable sequence was followed with Workspace4 and Workspace5.

**Figure 1-1 Workspace Tree**



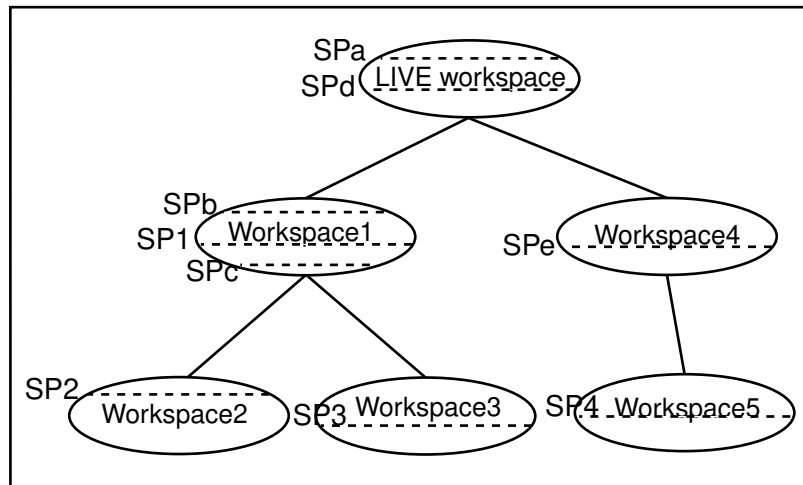
See also [Design Issue: Savepoint or Child Workspace?](#) for a discussion of design issues in deciding whether to create a child workspace or a savepoint for certain needs

## 1.1.2 Using Savepoints

A savepoint is a point in the workspace to which data changes can be rolled back. Workspace Manager accomplishes the rollback by deleting the row versions that contain the unwanted changes.

An **explicit savepoint** is a savepoint that you create and name. You can later roll back changes in version-enabled tables to the savepoint, or you can go to the savepoint to view the state of the entire database (including versioned rows) at the time the savepoint was created. In [Figure 1-2](#), SP1, SP2, SP3, and SP4 are explicit savepoints that were created in the workspaces indicated. (Savepoints are indicated by dashed lines in [Figure 1-2](#).)

Figure 1-2 Savepoints



In addition, **implicit savepoints** are created automatically whenever a new workspace is created. An implicit savepoint is needed so that the users in the child workspace get a view of the database that is frozen at the time of the workspace creation. Thus, in [Figure 1-2](#) two implicit savepoints (SPa and SPd) are created in the `LIVE` workspace corresponding to `Workspace1` and `Workspace4` creation; two implicit savepoints (SPb and SPc) are created in `Workspace1` corresponding to `Workspace2` and `Workspace3` creation; and one implicit savepoint (SPe) is created in `Workspace4` corresponding to `Workspace5` creation.

Workspace Manager uses the name `LATEST` to designate a logical savepoint that refers to the latest version in the workspace. `LATEST` is often the default when a savepoint is an optional parameter for a `DBMS_WM` subprogram (procedure or function).

A **removable savepoint** is a savepoint that can be deleted by the [CompressWorkspace](#), [CompressWorkspaceTree](#), and [DeleteSavepoint](#) procedures. A savepoint is removable if either of the following applies:

- It is an explicit savepoint.
- It is an implicit savepoint that does not have any child dependencies.
- [Design Issue: Savepoint or Child Workspace?](#)

### 1.1.2.1 Design Issue: Savepoint or Child Workspace?

A Workspace Manager design issue that you may face is whether to create a savepoint or a child workspace to "save" a project at a given point. Both a savepoint and a child workspace allow you to group a set of changes, compare changes in different row versions, and roll back a set of changes. However, creating a savepoint enables you to continue to make changes in the same workspace, and it allows other users in the workspace immediate access to the changes. (Changes in another workspace are not visible to users until the current workspace is refreshed or merged.) Creating a savepoint also makes it convenient to archive a set of changes, to which you can later roll back.

On the other hand, creating a child workspace is convenient for providing an isolated environment in which a complex set of changes can be made, completely removed



from the parent workspace (for example, the production data). If you want to set up an independent environment for a scenario, and if regular users in the parent workspace do not need access to this scenario's data, you probably want to create a child workspace instead of simply creating a savepoint in the parent workspace.

### 1.1.3 Merging and Rolling Back Workspace Changes

Workspaces can be merged or rolled back.

Merging a workspace involves applying changes made in a child workspace to its parent workspace, after which the child workspace is removed. To merge a workspace, use the [MergeWorkspace](#) procedure.

Rolling back a workspace involves deleting either all data changes (row versions) made in the workspace or all changes made after an explicit savepoint.

- To roll back all changes made in the workspace, use the [RollbackWorkspace](#) procedure.
- To roll back changes made in the workspace after a savepoint, use the [RollbackToSP](#) procedure.

#### Note:

You cannot roll back to a savepoint if any implicit savepoints were created since the specified savepoint, unless you first merge or remove the descendent workspaces that caused the implicit savepoints to be created. For example, referring to the figure in [Using Savepoints](#), the user in `Workspace1` cannot roll back to savepoint `SP1` until `Workspace3` (which caused implicit savepoint `SPc` to be created) is merged or removed.

A workspace cannot be rolled back when it has open database transactions. Rollback of a workspace leaves behind the workspace structure for future use; only the data in the workspace is deleted. (To completely remove a workspace, use the [RemoveWorkspace](#) procedure, as described in [Removing Workspaces](#).)

### 1.1.4 Resolving Conflicts Before a Merge or Refresh Operation

When a child workspace is merged, the row changes in the child workspace are incorporated in its parent workspace; and when a child workspace is refreshed, row changes in the parent workspace are incorporated in the child workspace. When a row is changed in both the child and parent workspace, a data conflict is created. Conflicts are automatically detected when a merge or refresh operation is requested, and they are presented to the user in conflict (`xxx_CONF`) views. There is one conflict view for each table, as described in [xxx\\_CONF Views](#). This view lists the column values of the rows in the two workspaces involved in the conflict.

You must resolve conflicts manually by setting the workspace conflict context for the session and then using the [ResolveConflicts](#) procedure. For each conflict you can choose to keep the row from the child workspace, the row from the parent workspace, or the common base row (that is, no change: keep the original data values for the row). You must resolve the conflicts before you can perform a merge ([MergeWorkspace](#)) or refresh ([RefreshWorkspace](#)) operation.

The **base row** is the currently visible row at the time of the first update or delete operation in the child workspace. In the case of an insert operation, the base row does not exist, except if the inserted row was previously deleted in an ancestor version of a parent workspace, the base row is that deleted row. The **parent row** is the currently visible row in the parent workspace at the time of the conflict resolution, and the **child row** is the currently visible row in the child workspace at the time of the conflict resolution.

Absence of data is not a conflict. In these cases, you can use the `xxx_DIFF` view (described in [xxx\\_DIFF Views](#)) to detect a change in one workspace when there is no row or no change to the row in the other workspace.

The general process for resolving conflicts is as follows:

1. Use the [GotoWorkspace](#) or [SetConflictWorkspace](#) procedure to set the workspace conflict context for the session.
2. Examine the `xxx_CONF` views (described in [xxx\\_CONF Views](#)) to see what conflicts exist.
3. Execute the [BeginResolve](#) procedure.
4. Execute the [ResolveConflicts](#) procedure as often as needed: once for each affected combination of table and workspace.
5. After resolving all conflicts, execute one of the following procedures:
  - [CommitResolve](#) if you want to make permanent all changes from the preceding step. (However, the changes are not made permanent in the database until you perform a standard database commit operation and execute the [MergeWorkspace](#) or [RefreshWorkspace](#) procedure, as described in the next step.)
  - [RollbackResolve](#) to discard all changes from the preceding step. (If you discard all changes, do not perform the next step.)
6. Perform a standard database commit operation and execute the [MergeWorkspace](#) or [RefreshWorkspace](#) procedure.

## 1.1.5 Freezing and Unfreezing Workspaces

You can control read and write access to a workspace by freezing and unfreezing the workspace. If a workspace is frozen, the ability of users to access the workspace and to make changes to rows in version-enabled tables is restricted. You can freeze a workspace in any of the following modes: no access, read-only, and one writer only (`1WRITER`).

To make a workspace frozen, use the [FreezeWorkspace](#) procedure. To make a frozen workspace not frozen, use the [UnfreezeWorkspace](#) procedure.

In addition, some procedures automatically freeze one or more workspaces. [Table 1-1](#) lists these procedures, the workspaces affected, and the mode in which the workspaces are frozen. (For explanations of the mode values, see the [FreezeWorkspace](#) procedure description in [DBMS\\_WM Package: Reference](#).)

**Table 1-1 Freeze Results of Procedures**

Procedure	Workspace and Mode
<a href="#">BeginResolve</a>	Specified workspace: 1WRITER
<a href="#">MergeWorkspace</a>	Specified workspace: NO_ACCESS Parent workspace: READ_ONLY
<a href="#">CompressWorkspace</a>	Specified workspace: NO_ACCESS (Also, checks to ensure that there are no sessions on savepoints other than LATEST.)
<a href="#">CompressWorkspaceTree</a>	Specified workspace: NO_ACCESS (Also, checks to ensure that there are no sessions on savepoints other than LATEST.)
<a href="#">CreateSavepoint</a>	Specified workspace: READ_ONLY
<a href="#">DeleteSavepoint</a>	Specified workspace: NO_ACCESS
<a href="#">CreateWorkspace</a>	Specified workspace: READ_ONLY
<a href="#">RemoveWorkspace</a>	Specified workspace: NO_ACCESS
<a href="#">RefreshWorkspace</a>	Specified workspace: READ_ONLY Parent workspace: READ_ONLY
<a href="#">RollbackResolve</a>	Specified workspace: 1WRITER
<a href="#">RollbackWorkspace</a>	Specified workspace: NO_ACCESS

## 1.1.6 Removing Workspaces

A workspace can be removed with the [RemoveWorkspace](#) procedure. [RemoveWorkspace](#) rolls back the data in a workspace and then deletes the workspace structure. An entire tree of workspaces can be removed with the [RemoveWorkspaceTree](#) procedure. This will remove the workspace and all its descendant workspaces. A workspace cannot be removed when it has users in it.

## 1.1.7 Using Workspace Manager Events

Several types of Workspace Manager operations can be captured as events, and can be communicated to applications through the Oracle Advanced Queuing (AQ) framework. Messaging features provided by AQ, such as asynchronous notification, persistence, propagation, access control, history, and rule-based subscription, can be used for Workspace Manager events.

Support for Workspace Manager events includes the `ALLOW_CAPTURE_EVENTS` Workspace Manager system parameter, the [SetCaptureEvent](#) procedure, and the `WM_EVENTS_INFO` metadata view.

[Workspace Manager Events](#) describes Workspace Manager events and explains how to use them in applications.

## 1.1.8 Autocommitting of Workspace Manager Operations

Many Workspace Manager operations are by default executed as autonomous database transactions that will be committed when they finish. That is, each such transaction is an independent transaction that is called from within the current database transaction, leaves the context of the calling transaction, performs the Workspace Manager operation and then

automatically commits it, and then returns to the calling transaction's context and continues with that transaction. Workspace Manager (DBMS\_WM) subprograms that operate in this way have an optional `auto_commit` parameter, which has a default value of `TRUE`.

For example, the [CompressWorkspace](#) procedure by default starts an autonomous transaction, compresses the workspace, commits the compression operation, and returns to the calling transaction's context, where the current database transaction continues.

However, if you want such subprograms not to start an autonomous transaction, but instead to execute in the context of the calling transaction, you can specify the `auto_commit` parameter with a value of `FALSE`. In this case, the Workspace Manager operation is executed as part of the current database transaction; and if there is no current open transaction, the Workspace Manager operation starts a new transaction. In either case, the Workspace Manager operation does not take effect until that transaction ends with a commit operation. For example, if you call the [CompressWorkspace](#) procedure with the `auto_commit` parameter specified as `FALSE`, the workspace is not compressed until the transaction is committed; and if the transaction is rolled back, the workspace is not compressed.

Note that if you specify `FALSE` for the `auto_commit` parameter, you must remember to commit or roll back the transaction explicitly.

If the `auto_commit` parameter value is `TRUE` and any open transactions exist, the following considerations apply:

- For the [CompressWorkspace](#) and [CompressWorkspaceTree](#) procedures, an exception is raised if there is any open transaction.
- For all other Workspace Manager procedures, an exception is raised if there is an open transaction in a parent or child workspace of any table that needs to be modified.

## 1.1.9 Continually Refreshed Workspaces

A **continually refreshed workspace** is a workspace that is automatically refreshed from its parent workspace whenever data changes are committed in the parent workspace or are merged into the parent workspace from another child workspace. You do not need to call the [RefreshWorkspace](#) procedure for a continually refreshed workspace.

Any workspace in a branch of the workspace tree can be continually refreshed. A child workspace can be a continually refreshed workspace, regardless of whether its parent workspace is continually refreshed. However, if a parent workspace is a continually refreshed workspace, its child workspaces must also be continually refreshed.

To create a continually refreshed workspace, specify `TRUE` for the `isrefreshed` parameter in the call to the [CreateWorkspace](#) procedure. See the Usage Notes for the [CreateWorkspace](#) procedure for rules that apply to the creation of a continually refreshed workspace.

To change a workspace that is not continually refreshed to be continually refreshed, use the [ChangeWorkspaceType](#) procedure.

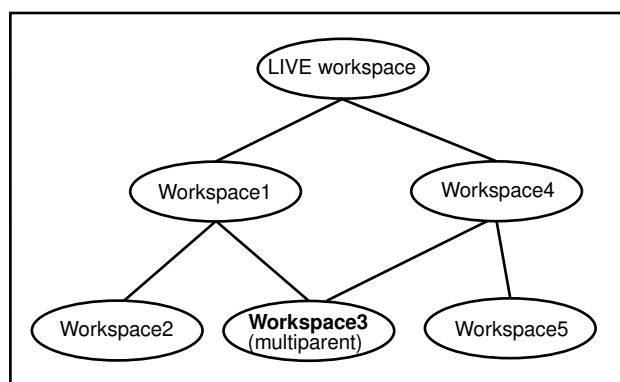
If a workspace is not continually refreshed, you must call the [RefreshWorkspace](#) procedure whenever you want to ensure that data changes in its parent workspace are visible in the workspace.

## 1.1.10 Multiparent Workspaces

A **multiparent workspace** is a child workspace that has two or more parent workspaces. A workspace is initially created with a single parent workspace. However, if the need arises, you can add other workspaces as parent workspaces to an existing workspace, thus making it a multiparent workspace. The multiparent workspace can see data from all of its parent workspaces and their ancestor workspaces, and it can be merged with and refreshed from its parent workspaces.

Figure 1-3 shows the same hierarchy of workspaces in Figure 1-1, except that `Workspace3` is now a multiparent workspace with two parent workspaces: `Workspace1` and `Workspace4`.

**Figure 1-3 Multiparent Workspace in a Workspace Hierarchy**



A multiparent workspace is also called a **multiparent leaf workspace**. Thus, in Figure 1-3, `Workspace3` is a multiparent leaf workspace. The nearest common ancestor of all parent workspaces of a multiparent leaf workspace is called the **multiparent root workspace**. In Figure 1-3, the `LIVE workspace` is the multiparent root workspace of `Workspace3`. All of the workspaces in the directed acyclic graph (DAG) formed as a result of adding parent workspaces as parents of a leaf workspace are called **multiparent graph workspaces**. In Figure 1-3, `Workspace1`, `Workspace4`, and `Workspace3` are the multiparent graph workspaces.

Multiparent workspaces are allowed only if the `ALLOW_MULTI_PARENT_WORKSPACE` Workspace Manager system parameter is set to `ON`. In addition, for a continually refreshed workspace to be a multiparent workspace, the `CR_WORKSPACE_MODE` Workspace Manager system parameter must be set to `PESSIMISTIC_LOCKING`; and for a workspace that is not continually refreshed to be a multiparent workspace, the `NONCR_WORKSPACE_MODE` Workspace Manager system parameter must be set to `PESSIMISTIC_LOCKING`. For information about Workspace Manager system parameters, see [System Parameters for Workspace Manager](#).

To create a multiparent workspace, use the [AddAsParentWorkspace](#) procedure. To remove a workspace as a parent of a multiparent workspace, use the [RemoveAsParentWorkspace](#) procedure. To grant and revoke privileges on multiparent graph workspaces, use the [GrantGraphPriv](#) and [RevokeGraphPriv](#) procedures, respectively. These procedures are described in [DBMS\\_WM Package: Reference](#).

Workspace Manager provides the following static data dictionary views (described in [Workspace Manager Static Data Dictionary Views](#)) to store information about multiparent workspaces:

- [USER\\_MP\\_GRAPH\\_WORKSPACES](#) and [ALL\\_MP\\_GRAPH\\_WORKSPACES](#) contain information about multiparent graph workspaces.
- [USER\\_MP\\_PARENT\\_WORKSPACES](#) and [ALL\\_MP\\_PARENT\\_WORKSPACES](#) contain information about parent workspaces of multiparent leaf workspaces.

### 1.1.11 Infrastructure for Version-Enabling of Tables

When you version-enable a table using the [EnableVersioning](#) procedure, Workspace Manager automatically performs operations and creates data structures that are invisible to non-DBA users, but that permit Workspace Manager to function. Some of the information maintained by Workspace Manager is stored in the static data dictionary views described in [Workspace Manager Static Data Dictionary Views](#), and some is stored in system data structures not accessible by users.

When a table is version-enabled, Workspace Manager renames the table to `<table-name>_LT`, and it adds several columns to this table to store versioning metadata. The added columns have names starting with `WM_`: `WM_VERSION`, `WM_CREATETIME`, `WM_RETIRETIME`, `WM_NEXTVER`, `WM_DELSTATUS`, and `WM_LTLOCK`.

Note that users and applications should not specify the `<table-name>_LT` table in SQL statements; they should continue to specify the original table name (`<table-name>`). (If you ever need to find the name of the `<table_name>_LT` table associated with a version-enabled table, or if you want to find out if a table is version-enabled by checking for the existence of a `<table_name>_LT` table, use the [GetPhysicalTableName](#) function.)

Workspace Manager also creates a view on the original table (`<table-name>`), as well as `INSTEAD OF` triggers on the view for insert, update, and delete operations. When an application executes a statement to insert, update, or delete data in a version-enabled table, the appropriate `INSTEAD OF` trigger performs the actual operation. When the view is accessed, it uses the workspace metadata to show only the row versions relevant to the current workspace of the user.

Because Workspace Manager uses the original object name when it creates infrastructure objects, the effective maximum length of the name for some kinds of objects is shorter than the maximum permitted by Oracle Database. The database can be defined to support either 30 or 128 character identifiers based on the database's compatible setting. If the compatibility setting is  $\geq 12.2$  then the identifier length (`<DB-identifier-length>`) is 128; otherwise, it is 30. The specific object length restrictions are expressed as an offset of (a value to be subtracted from) that `<DB-identifier-length>` value.

[Table 1-2](#) provides guidelines for the maximum name length for version-enabled tables and related objects. (See also the information in [Workspace Manager Reserved Words and Characters](#) about reserved words and characters for certain names.)

**Table 1-2 Name Length Guidelines for Workspace Manager**

Type of Object	Maximum Name Length in Characters
Table	<code>&lt;DB-identifier-length&gt; - 5</code>
Column	<code>&lt;DB-identifier-length&gt; - 2</code>
Index	<code>&lt;DB-identifier-length&gt;</code> ( <code>&lt;DB-identifier-length&gt; - 4</code> if the index is created or altered between calls to the <a href="#">BeginDDL</a> and <a href="#">CommitDDL</a> procedures)



**Table 1-2 (Cont.) Name Length Guidelines for Workspace Manager**

Type of Object	Maximum Name Length in Characters
Trigger	<DB-identifier-length>
Constraint	<DB-identifier-length> (<DB-identifier-length> - 4 if the constraint is created or altered between calls to the <a href="#">BeginDDL</a> and <a href="#">CommitDDL</a> procedures)

Workspace Manager does not support SQL MERGE statement (any use), or the RETURNING clause with INSERT or UPDATE statements, on version-enabled tables. The RETURNING clause restriction is caused by the fact that Workspace Manager creates views with `INSTEAD OF` triggers on version-enabled tables, and Oracle Database does not support the RETURNING clause on views that have `INSTEAD OF` triggers defined on them.

## 1.1.12 Creation of Row Versions and Historical Copies

This section describes the process by which Workspace Manager creates new row versions and maintains historical copies of old versions.

A new row version is created in a version-enabled table when you do either of the following:

- Create an explicit savepoint in the current workspace, and update the row.
- Create a child workspace (using the [CreateWorkspace](#) procedure) under the current workspace, thus creating an implicit savepoint in the current workspace; execute the [GotoWorkspace](#) procedure to go to the child workspace; and update the row.

Any subsequent update of the current row overwrites the current row version, unless history is enabled on the table or another savepoint is created.

- If history is enabled *without* overwrite (`VIEW_WO_OVERWRITE`), each update of the current row version creates a full copy of the current row version with the changes and with a timestamp based on the transaction time. This copy becomes the new current row version.
- If history is enabled *with* overwrite (`VIEW_W_OVERWRITE`), each update of the current row version overwrites the current row version and updates the transaction timestamp.
- If a savepoint is created in the workspace, the next change to the row creates a new version, and the history cycle begins again.

Row versions created in a workspace are not visible from the parent workspace until you execute the [MergeWorkspace](#) or [MergeTable](#) procedure.

When you execute the [MergeWorkspace](#) procedure for a child workspace, only the current row version in the child workspace is merged into the parent workspace. If you specify the `remove_workspace` parameter as `TRUE`, any intermediate row versions in the child workspace are deleted when the child workspace is removed. To retain all intermediate versions created in the child workspace, the `remove_workspace` parameter value must be `FALSE` (the default).

When you execute the [CompressWorkspace](#) procedure on a child workspace to eliminate intermediate savepoints, you can also remove the associated historical copies of that row version. If you do not remove these copies, they are associated with the next version.

Intermediate row versions can only be selected for read-only access. To select an intermediate row version for read-only access, go to the workspace ([GotoWorkspace](#) procedure) if you are not already in it, and execute either the [GotoDate](#) procedure to set the

session context to a historical time or the [GotoSavepoint](#) procedure to set the session context to a specific savepoint. Subsequent SELECT statements will select, for read-only access, the latest row version as of the specified date or savepoint.

[Example 1-1](#) shows when row versions are created for a version-enabled table after the creation of an implicit savepoint and an explicit savepoint. For the example, assume that the EMP table contains columns named EMPNO, SALARY, and COMMISSION.

If [Example 1-1](#) is executed for a table that has the `VIEW_WO_OVERWRITE` history option, then the `xxx_HIST` views (see [xxx\\_HIST Views](#)) will contain a single row for each DML operation. For example:

```
select salary, commission, wm_workspace, wm_optype from EMP_HIST where empno = 100;
```

SALARY	COMMISSION	WM_WORKSPACE	WM_OPTYPE
0	0	LIVE	I
10000	0	LIVE	U
10000	1000	LIVE	U
20000	1000	Workspace 1	U
20000	2000	Workspace 1	U
30000	3000	LIVE	U
40000	3000	LIVE	U
40000	4000	LIVE	U
40000	4000	LIVE	D

However, if [Example 1-1](#) is executed for a table that has the `VIEW_W_OVERWRITE` or `NONE` history option, then the `xxx_HIST` views (see [xxx\\_HIST Views](#)) will contain only a single row for each savepoint, because the row version is overridden by subsequent DML operations within the same savepoint. For example:

```
select salary, commission, wm_workspace, wm_optype from EMP_HIST where empno = 100;
```

SALARY	COMMISSION	WM_WORKSPACE	WM_OPTYPE
10000	1000	LIVE	U
20000	2000	Workspace 1	U
30000	3000	LIVE	U
40000	4000	LIVE	D

### Example 1-1 Row Versions Created After Implicit and Explicit Savepoints

```
execute dbms_wm.GotoWorkspace('LIVE') ;
insert into EMP(empno, salary, commission) values (100, 0, 0) ;
update EMP set salary = 10000 ;
update EMP set commission = 1000 ;
commit ;

-- CreateWorkspace creates an implicit savepoint in 'LIVE'; DML operations
-- create new row versions for both workspaces.
execute dbms_wm.CreateWorkspace('WorkSpace 1') ;
```



```
execute dbms_wm.GotoWorkspace('WorkSpace 1') ;
update EMP set salary = 20000 ;
update EMP set commission = 2000 ;
commit ;

execute dbms_wm.GotoWorkspace('LIVE') ;
update EMP set salary = 30000, commission = 3000 ;
commit ;

-- CreateSavepoint creates an explicit savepoint in 'LIVE'; DML operations
-- create new row versions for only the LIVE workspace.
execute dbms_wm.CreateSavepoint('LIVE', 'SP1') ;
update EMP set salary = 40000 ;
update EMP set commission = 4000 ;
delete EMP where empno = 100 ;
commit ;
```

### 1.1.13 Workspace Manager Schema, Metadata, and Package

Workspace Manager creates a user named `WMSYS`. The `WMSYS` schema is used to store all the metadata information for Workspace Manager. A PL/SQL package with the public synonym `DBMS_WM` contains the Workspace Manager subprograms (procedures and functions).

The following privileges are granted to the `PUBLIC` user group:

- `SELECT` privilege on Workspace Manager static data dictionary views (described in [Workspace Manager Static Data Dictionary Views](#) )
- `EXECUTE` privilege on the `DBMS_WM` package (described in [DBMS\\_WM Package: Reference](#) )

## 1.2 Session Context Information for Workspace Manager

Users perform Workspace Manager operations within a standard Oracle session.

A session is a specific connection of a user to an Oracle instance through a user process; a session lasts from the time the user connects until the time the user disconnects or exits the database application. When you perform Workspace Manager operations, information relating to the session context is automatically recorded.

The session context information includes the workspace name and a context value, and it determines what data the session can see in the workspace and what workspaces the session can enter. The context value is one of the following:

- `LATEST`: The session is currently set to the `LATEST` savepoint (explained in [Using Savepoints](#)), and it can see changes as they are made in the workspace. The context is automatically set to `LATEST` when the session enters the workspace (using the [GotoWorkspace](#) procedure).
- A savepoint name: The session is currently set to a savepoint in the workspace. The session cannot see changes as they are made in the latest version of the workspace, but instead sees a static view of the data as of the savepoint creation time. The session context is set to the savepoint name after a call to the [GotoDate](#) procedure.
- An instant (a point in time): The session is currently set to a specific point in time. The session cannot see changes as they are made in the latest version of the workspace, but instead sees a static view of the data as of the specific time. The session context is set to an instant after a call to the [GotoDate](#) procedure. (The exact time point depends on the

history option for tracking modifications, as set by the [EnableVersioning](#) procedure or modified by the [SetWoOverwriteOFF](#) or [SetWoOverwriteON](#) procedure.)

You can retrieve information about the session context by using the [GetSessionInfo](#) procedure. Retrieving this information can be useful if you need to know where a session is (workspace and context) -- for example, after you performed a combination of [GotoWorkspace](#), [GotoSavepoint](#), and [GotoDate](#) operations.

## 1.3 Lock Management with Workspace Manager

In addition to locks provided by regular Oracle database transactions, Workspace Manager provides two types of version locks. These locks are primarily intended to eliminate row conflicts between a parent workspace and a child workspace. You can enable locking for the workspace, the session, or specified rows, or some combination:

- Lock at the workspace level ([SetWorkspaceLockModeON](#) procedure) if the data changes are in one or a few workspaces, or if you want all data changes in the workspace to be locked.
- Lock at the session level ([SetLockingON](#) procedure) if the data changes are being made in many workspaces. When locking is enabled for a session, Workspace Manager locks rows in all workspaces in which the session participates.
- Lock specific rows ([LockRows](#) procedure) either to lock the rows before they are updated or to automatically lock rows after they are inserted (or updated if they satisfy the `WHERE` clause after the update).

Workspace or session locks persist for the duration of the workspace or session, respectively, or until the workspace is merged or rolled back.

Like database locks, Workspace Manager locks can be exclusive or shared:

- Exclusive locks - The locks are very similar to database transaction locks in that once an exclusive lock is placed on a record, no other user in the database can change the record except for the session (user) that locked it. When exclusive locking is enabled for a user, any row that the user changes is locked exclusively. In addition, the parent row to that row is also locked exclusively. Thus, exclusive locking can be used to eliminate data conflicts between a child and its parent workspace. (However, see [Exclusive Locking and Row Versions](#) for information about row versions with exclusive locking, and how the timing of an exclusive lock with respect to an update operation can affect whether other users can update a row.)
- Shared locks - Once a shared lock is placed on a row, only users in the workspace in which it is locked are allowed to modify it. Shared locks are also placed on the parent version of the row, thus protecting the row from conflicts. The benefit of shared locks over exclusive locks is that all users in the workspace where the row is locked can access the row for changes. An ideal use for this kind of lock is on a row that needs to have no conflicts with its parent, but that needs to be changed by a collection of users participating in a group project. Note that shared locking must be individually enabled for each session in the workspace.

Workspace-exclusive locks and version-exclusive locks are forms of exclusive locking that control which users can and cannot change data values, but (unlike exclusive locking) they do not prevent conflicts from occurring. **Workspace-exclusive locks** lock rows such that only the user that set the lock can change the values in the current workspace; however, other users in other workspaces can change the values.

**Version-exclusive locks** lock rows such that only the user that set the lock can

change the values (and that user can be in any workspace); no other users (in any workspace) can change the values.

The following table indicates, for a row locked by a specific user in a specific workspace, which users in which workspaces can and cannot modify the row. For example, the first two entries in this table mean that when a shared (S) lock is placed on a row, any user in the workspace in which the row was locked can modify the row, but any user in a workspace different from the workspace in which the row was locked cannot modify the row.

**Table 1-3 Workspace Manager Lock Modes and Data Modification Ability**

Lock Mode	User	Workspace of User	Can Modify?
Shared (S)	Any user	Workspace in which row was locked	Yes
Shared (S)	Any user	Different from workspace in which row was locked	No
Exclusive (E)	User that locked the row	Workspace in which row was locked	Yes
Exclusive (E)	User that locked the row	Different from workspace in which row was locked	No
Exclusive (E)	Different user from the one that locked the row	Any workspace	No
Workspace Exclusive (WE)	User that locked the row	Any workspace	Yes
Workspace Exclusive (WE)	Different user from the one that locked the row	Different from workspace in which row was locked	Yes
Workspace Exclusive (WE)	Different user from the one that locked the row	Workspace in which row was locked	No
Version Exclusive (VE)	User that locked the row	Any workspace	Yes
Version Exclusive (VE)	Different user from the one that locked the row	Any workspace	No

Locking a row does not affect workspace merge, refresh, and rollback operations, but it affects what can be done with the row after these operations. You can control these workspace operations by using workspace privileges, calling the [FreezeWorkspace](#) procedure, and checking the workspace `xxx_LOCK` view or views (described in [xxx\\_LOCK Views](#)) before performing the operations.

The `xxx_LOCK` static data dictionary views (described in [xxx\\_LOCK Views](#)) contain information about locks in each version-enabled table.

For information about Workspace Manager locking with DML operations on tables with referential integrity constraints, see [Locking with DML Operations on Tables with Referential Integrity Constraints](#).

- [Exclusive Locking and Row Versions](#)
- [Locks Taken for Workspace Manager Operations](#)

## 1.3.1 Exclusive Locking and Row Versions

The timing of an exclusive lock with respect to an update operation in a child workspace can affect which version, if any, of the row can be updated in a parent workspace. For example, when a table is version-enabled in the `LIVE` workspace, each original row is assigned version 0. Assume that a workspace named `W1` is created as a child of the `LIVE` workspace. When workspace `W1` is created, the following things happen:

- Version 1 is assigned to the `LIVE` workspace (but no additional row is created).
- Version 2 is assigned to workspace `W1` (but no additional row is created). Queries in workspace `W1` still return version 0 of the row that is in the `LIVE` workspace.

Using this example, if a user in workspace `W1` places an exclusive lock on a row before it updates the row, only that user in workspace `W1` can update the row. Specifically:

- Version 0 of the row is locked, preventing any update of the row from any workspace until the row is unlocked.
- The lock can be placed from workspace `W1` (or a descendent workspace of `W1`) because version 0 is the current physical row for the workspace.
- When a user in workspace `W1` updates the row, a new row (version 2) is created that is visible only from workspace `W1` and any of its child workspaces.

However, if the row is not locked in the `LIVE` workspace and if a user in workspace `W1` updates the row and then places an exclusive lock on the row, a user in the `LIVE` workspace can update the row. Specifically:

- A new row (version 2) is created that is visible only from workspace `W1` and any of its child workspaces.
- Version 2 of the row is locked. No user in workspace `W1` other than the user that placed the lock, or no user in any child workspace of `W1`, can update the row or create a new version of the row.
- Version 0 of the row in the `LIVE` workspace is not locked. If a user in the `LIVE` workspace or a sibling workspace of `W1` updates the row, a new version (version 1) of the row is created. (Version 0 is not locked because it is no longer the current version of the row for users in workspace `W1`; rather, version 2 is the current version of the row in that workspace.)

In other words, an exclusive lock after an update does not lock previous versions of the row in workspaces above the locking workspace in the workspace tree or in other branches of the workspace tree.

## 1.3.2 Locks Taken for Workspace Manager Operations

Workspace Manager automatically takes locks when performing certain operations related to `DBMS_WM` subprograms. Workspace Manager checks:

- If the required lock can be successfully requested
- If the workspace is frozen in a mode that is incompatible with the requested operation

If the required lock cannot be requested or if the workspace is frozen in an incompatible mode, an error is generated.

Table 1-4 lists Workspace Manager operations and the incompatible freeze modes for certain types of workspaces (parent workspace, current workspace, intermediate multiparent workspace) to be locked for the operation. In Table 1-4:

- N/A means not applicable; that is, no freeze modes are incompatible.
- Exclusive + means that in addition to an exclusive lock, no sessions can be in the workspace.
- Sub eXclusive means the SX\_MODE constant as used in the DBMS\_LOCK procedure, described in *Oracle Database PL/SQL Packages and Types Reference*. SX\_MODE can be used on an aggregate object to indicate that exclusive locks are being acquired on sub-parts of the object.
- Shared Sub eXclusive means the SSX\_MODE constant as used in the DBMS\_LOCK procedure, described in *Oracle Database PL/SQL Packages and Types Reference*. SSX\_MODE can be used to indicate that the entire aggregate object has a shared lock, but some of the sub-parts may also have exclusive locks.
- Sub Shared means the SS\_MODE constant as used in the DBMS\_LOCK procedure, described in *Oracle Database PL/SQL Packages and Types Reference*. SS\_MODE can be used on an aggregate object to indicate that share locks are being acquired on subparts of the object.
- Shared(1) means shared, except Shared Sub eXclusive when ROW\_LEVEL\_LOCKING=OFF.

For [CompressWorkspace](#) and [CompressWorkspaceTree](#), when the current version within any affected workspace is contained in a compressible range with at least one other version, an attempt is made to acquire a Shared Sub eXclusive lock on that workspace. If it fails to be acquired, no error is raised, but the current version is not compressed.

When the [MergeTable](#) and [MergeWorkspace](#) procedures are executed with the `create_savepoint` parameter set to `true`, the parent workspace lock taken is an SSX (Shared Sub eXclusive) lock.

**Table 1-4 Operations and Incompatible Freeze Modes for Workspace Types**

Operation	Parent Workspace	Incompatible Freeze Modes	Current Workspace	Incompatible Freeze Modes	Intermediate Multiparent Workspace	Incompatible Freeze Modes
Any DML	None	N/A	Shared	DEFERRED_REMOVAL, NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_SESSION, WM_ONLY	None	N/A
<a href="#">BeginResolve</a>	None	N/A	Shared Sub eXclusive	DEFERRED_REMOVAL	None	N/A

**Table 1-4 (Cont.) Operations and Incompatible Freeze Modes for Workspace Types**

Operation	Parent Workspace	Incompatible Freeze Modes	Current Workspace	Incompatible Freeze Modes	Intermediate Multiparent Workspace	Incompatible Freeze Modes
<a href="#">ChangeWorkspaceType</a>	Shared	N/A	Shared SubeXclusive	DEFERRED_R EMOVAL, NO_ACCESS, ONEWRITER, ONEWRITER_SESSION, READ_ONLY	Exclusive	NO_ACCESS, ONEWRITER, ONEWRITER_SESSION, READ_ONLY
<a href="#">CommitResolve</a>	None	N/A	Shared SubeXclusive	DEFERRED_R EMOVAL	None	N/A
<a href="#">CompressWorkspace</a>	None	N/A	Shared	DEFERRED_R EMOVAL, NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_SESSION	None	N/A
<a href="#">CompressWorkspaceTree</a>	None	N/A	Shared	DEFERRED_R EMOVAL, NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_SESSION	None	N/A
<a href="#">CopyWorkspace</a>	None	N/A	Shared	DEFERRED_R EMOVAL, NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_SESSION	None	N/A
<a href="#">CreateSavepoint</a>	None	N/A	Shared (Shared SubeXclusive when new version must be created)	DEFERRED_R EMOVAL, NO_ACCESS	None	N/A
<a href="#">CreateWorkspace</a>	None	N/A	Shared (Shared SubeXclusive when new version must be created)	DEFERRED_R EMOVAL, NO_ACCESS	None	N/A

**Table 1-4 (Cont.) Operations and Incompatible Freeze Modes for Workspace Types**

Operation	Parent Workspace	Incompatible Freeze Modes	Current Workspace	Incompatible Freeze Modes	Intermediate Multiparent Workspace	Incompatible Freeze Modes
DeleteSavepoint	None	N/A	Exclusive +	DEFERRED_R EMOVAL, NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_ SESSION	None	N/A
Export	None	N/A	Shared	DEFERRED_R EMOVAL, NO_ACCESS	None	N/A
FreezeWorkspace	None	N/A	Shared Sub eXclusive	DEFERRED_R EMOVAL	None	N/A
GotoWorkspace	None	N/A	Sub Shared	DEFERRED_R EMOVAL, NO_ACCESS <sup>1</sup>	None	N/A
Import	None	N/A	Shared Sub eXclusive	DEFERRED_R EMOVAL, NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_ SESSION	None	N/A
LockRows	None	N/A	Shared	DEFERRED_R EMOVAL, NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_ SESSION	None	N/A
MergeTable (remove_data=>false)	Shared	NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_ SESSION	Sub eXclusive	DEFERRED_R EMOVAL, NO_ACCESS	Sub eXclusive	NO_ACCESS
MergeTable (remove_data=>true)	Shared	NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_ SESSION	Sub eXclusive	DEFERRED_R EMOVAL, NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_ SESSION	Sub eXclusive	NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_ SESSION

**Table 1-4 (Cont.) Operations and Incompatible Freeze Modes for Workspace Types**

Operation	Parent Workspace	Incompatible Freeze Modes	Current Workspace	Incompatible Freeze Modes	Intermediate Multiparent Workspace	Incompatible Freeze Modes
<a href="#">MergeWorkspace</a> (remove_workspace=>false)	Shared(1)	NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_SESSION	Shared SubeXclusive	DEFERRED_R EMOVAL, NO_ACCESS	Exclusive	NO_ACCESS
<a href="#">MergeWorkspace</a> (remove_workspace=>>true)	Shared(1)	NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_SESSION	Exclusive +	DEFERRED_R EMOVAL, NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_SESSION	Exclusive +	NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_SESSION
<a href="#">PurgeTable</a>	None	N/A	None; table locked in Exclusive mode	None	None	N/A
<a href="#">RefreshTable</a>	Shared(1)	NO_ACCESS	Shared	DEFERRED_R EMOVAL, NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_SESSION	Shared	NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_SESSION
<a href="#">RefreshWorkspace</a>	Shared(1)	NO_ACCESS	Shared	DEFERRED_R EMOVAL, NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_SESSION	Exclusive	NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_SESSION
<a href="#">RemoveDeferredWorkspaces</a>	None	N/A	Exclusive +	N/A	None	N/A
<a href="#">RemoveWorkspace</a>	Shared(1)	N/A	Exclusive +	NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_SESSION	Shared	N/A
<a href="#">RenameSavepoint</a>	None	N/A	Exclusive +	DEFERRED_R EMOVAL, NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_SESSION	None	N/A



**Table 1-4 (Cont.) Operations and Incompatible Freeze Modes for Workspace Types**

Operation	Parent Workspace	Incompatible Freeze Modes	Current Workspace	Incompatible Freeze Modes	Intermediate Multiparent Workspace	Incompatible Freeze Modes
<a href="#">RenameWorkspace</a>	None	N/A	Exclusive +	DEFERRED_R EMOVAL  NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_ SESSION	None	N/A
<a href="#">RollbackResolve</a>	None	N/A	Shared Sub eXclusive	DEFERRED_R EMOVAL	None	N/A
<a href="#">RollbackTable</a>	None	N/A	Shared Sub eXclusive	DEFERRED_R EMOVAL,  NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_ SESSION	None	N/A
<a href="#">RollbackWorkspace</a>	None	N/A	Shared Sub eXclusive	DEFERRED_R EMOVAL,  NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_ SESSION	None	N/A
<a href="#">SetConflictWorkspace</a>	None	NO_ACCESS	Sub Shared	DEFERRED_R EMOVAL, NO_ACCESS	None	N/A
<a href="#">SetDiffVersions</a>	None	N/A	Sub Shared	DEFERRED_R EMOVAL, NO_ACCESS	None	N/A
<a href="#">SetMultiWorkspaces</a>	None	N/A	Sub Shared	DEFERRED_R EMOVAL, NO_ACCESS	None	N/A
<a href="#">SetWorkspaceLockModeOFF</a>	None	N/A	Shared Sub eXclusive	DEFERRED_R EMOVAL, NO_ACCESS	None	N/A
<a href="#">SetWorkspaceLockModeON</a>	None	N/A	Shared Sub eXclusive	DEFERRED_R EMOVAL, NO_ACCESS	None	N/A
<a href="#">UnfreezeWorkspace</a>	None	N/A	Shared Sub eXclusive	DEFERRED_R EMOVAL	None	N/A

**Table 1-4 (Cont.) Operations and Incompatible Freeze Modes for Workspace Types**

Operation	Parent Workspace	Incompatible Freeze Modes	Current Workspace	Incompatible Freeze Modes	Intermediate Multiparent Workspace	Incompatible Freeze Modes
UnlockRows	None	N/A	Shared	DEFERRED_REMOVAL, NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_SESSION	None	N/A

<sup>1</sup> The LIVE workspace cannot be frozen in NO\_ACCESS mode.

## 1.4 Privilege Management with Workspace Manager

Workspace Manager provides a set of privileges that are separate from standard Oracle database privileges.

Workspace Manager **workspace-level privileges** (with names in the form `xxx_WORKSPACE`) allow the user to affect a specified workspace, and **system-level privileges** (with names in the form `xxx_ANY_WORKSPACE`) allow the user to affect any workspace.

[Table 1-5](#) lists the Workspace Manager privileges.

**Table 1-5 Workspace Manager Privileges**

Privilege	Description
ACCESS_WORKSPACE	Allows the user to go to a specified workspace. ACCESS_WORKSPACE or ACCESS_ANY_WORKSPACE privilege is needed for all other privileges.
ACCESS_ANY_WORKSPACE	Allows the user to go to any workspace. ACCESS_WORKSPACE or ACCESS_ANY_WORKSPACE privilege is needed for all other privileges.
CREATE_WORKSPACE	Allows the user to create a child workspace in a specified workspace.
CREATE_ANY_WORKSPACE	Allows the user to create a child workspace in any workspace.
FREEZE_WORKSPACE	Allows the user to freeze and unfreeze a specified workspace.
FREEZE_ANY_WORKSPACE	Allows the user to freeze and unfreeze any workspace.
GRANTPRIV_WORKSPACE	Allows the user to grant privileges on the workspace to other users.
GRANTPRIV_ANY_WORKSPACE	Allows the user to grant privileges on any workspace to other users.
MERGE_WORKSPACE	Allows the user to merge the changes in a specified workspace to its parent workspace.

**Table 1-5 (Cont.) Workspace Manager Privileges**

Privilege	Description
MERGE_ANY_WORKSPACE	Allows the user to merge the changes in any workspace to its parent workspace.
REMOVE_WORKSPACE	Allows the user to remove a specified workspace.
REMOVE_ANY_WORKSPACE	Allows the user to remove any workspace.
ROLLBACK_WORKSPACE	Allows the user to roll back the changes in a specified workspace.
ROLLBACK_ANY_WORKSPACE	Allows the user to roll back the changes in any workspace.
WM_ADMIN	Provides the user with all Workspace Manager-related privileges with the grant option.

Each privilege can be granted with or without the grant option. The **grant option** allows the user to which the privilege is granted to grant the privilege to other users.

The `WM_ADMIN` system privilege has all Workspace Manager privileges with the grant option. By default, the `WM_ADMIN` system privilege is granted to `WM_ADMIN_ROLE`. This role is in turn granted to the database administrator (DBA role). Thus, after you decide which users should be granted which privileges, either have the DBA grant the privileges, or have the DBA grant the `WM_ADMIN_ROLE` role to one or more selected users and have these users grant the privileges.

The [GrantWorkspacePriv](#) and [GrantSystemPriv](#) procedures are used to grant workspace-level privileges and system-level privileges, respectively.

The [RevokeWorkspacePriv](#) and [RevokeSystemPriv](#) procedures are used to revoke workspace-level privileges and system-level privileges, respectively. These procedures require that the user have sufficient privilege to revoke the specified privilege from the specified user. The user that granted a privilege can revoke it.

## 1.5 System Parameters for Workspace Manager

Workspace Manager provides a set of system parameters that allow a user with the `WM_ADMIN` system privilege to enforce global Workspace Manager-specific settings for the database.

The `WM_ADMIN` system privilege is described in [Privilege Management with Workspace Manager](#). These Workspace Manager system parameters are not Oracle initialization parameters. The only way to set Workspace Manager system parameters is to use the [SetSystemParameter](#) procedure, described in [DBMS\\_WM Package: Reference](#).

To set a system parameter, use the [SetSystemParameter](#) procedure. To get the current setting for a system parameter, use the [GetSystemParameter](#) procedure. Both procedures are described in [DBMS\\_WM Package: Reference](#).

The following table lists the Workspace Manager system parameters.

**Table 1-6 Workspace Manager System Parameters**

Parameter Name	Values
ADD_UNIQUE_COLUMN_TO_HISTORY_VIEW	<p>ON adds columns to the xxx_HIST views (described in <a href="#">xxx_HIST Views</a>) of version-enabled tables. When a table is version-enabled with valid time support (described in <a href="#">Workspace Manager Valid Time Support</a>), WM_ROWID (rowid) and WM_FLAG (integer) are added. When a table is version-enabled without valid time support, only the WM_ROWID column is added. These columns can be used to uniquely identify each row in the xxx_HIST views. For a table with valid time support, you must use the WM_ROWID column and the WM_FLAG column in any WHERE clauses.</p> <p>OFF (the default) does not add any columns to the xxx_HIST views.</p>
ALLOW_CAPTURE_EVENTS	<p>ON allows Workspace Manager events (described in <a href="#">Workspace Manager Events</a>) to be captured. Setting this parameter to ON causes some additional internal Workspace Manager processing operations; therefore, for performance reasons you should not set the value to ON unless you plan to capture events.</p> <p>OFF (the default) does not allow Workspace Manager events to be captured.</p>
ALLOW_MULTI_PARENT_WORKSPACES	<p>ON allows multiparent workspaces (described in <a href="#">Multiparent Workspaces</a>) to be created. Setting this parameter to ON causes some additional internal Workspace Manager processing operations; therefore, for performance reasons you should not set the value to ON unless you plan to use multiparent workspaces.</p> <p>OFF (the default) does not allow multiparent workspaces to be created.</p>
ALLOW_NESTED_TABLE_COLUMNS	<p>ON allows tables containing a nested table column to be version-enabled. Setting this parameter to ON causes some additional internal Workspace Manager processing operations; therefore, for performance reasons you should not set the value to ON unless you plan to version-enable any tables with nested table columns.</p> <p>OFF (the default) does not allow tables containing a nested table column to be version-enabled.</p>
COMPRESS_PARENT_AFTER_REMOVE	<p>ON (the default) implicitly calls the <a href="#">CompressWorkspace</a> procedure to compress the parent workspace after a <a href="#">RemoveWorkspace</a> operation, or after a <a href="#">MergeWorkspace</a> operation in which the <code>remove_workspace</code> parameter was set to true.</p> <p>OFF does not compress the parent workspace in these cases.</p>

Table 1-6 (Cont.) Workspace Manager System Parameters

Parameter Name	Values
CR_WORKSPACE_MODE	<p>OPTIMISTIC_LOCKING allows a record to be edited in two or more continually refreshed workspaces. If differences occur between parent and child workspaces, the record is considered to be in conflict, and the conflict must be resolved before the child workspace can be merged or refreshed.</p> <p>PESSIMISTIC_LOCKING does not allow a record to be edited in two or more continually refreshed workspaces. This setting ensures that there are no conflicts between parent and child workspaces.</p> <p>OPTIMISTIC_LOCKING is the default for new installations, but PESSIMISTIC_LOCKING is the default for upgrades from a version before release 9.2.0.2.</p>
CREATEWORKSPACE_SHARED_LOCK	<p>Applies to the creation of workspaces that are not continually refreshed.</p> <p>ON causes the <a href="#">CreateWorkspace</a> procedure to acquire only a Shared lock on the parent workspace when creating the workspace, if possible. This allows multiple <a href="#">CreateWorkspace</a> procedures from the same parent workspace to be executed simultaneously, as well to run at the same time as other procedures that have a compatible lock on the same workspace. This is implemented by creating the workspace not from the latest version, but from the most recent non-latest savepoint. Any modified rows in the latest version of the workspace will not be seen by the workspace until a refresh operation occurs.</p> <p>OFF (the default) causes the <a href="#">CreateWorkspace</a> procedure to acquire a Shared Sub eXclusive (SSX) lock on the parent workspace when creating a workspace. This prevents multiple <a href="#">CreateWorkspace</a> procedures from the same parent workspace being run simultaneously, but creates the workspace from the latest version of the workspace, and so all data from the parent workspace is immediately seen.</p>
DEFAULT_WORKSPACE	<p>The name of an existing workspace, identifying the default workspace that users are placed into when first connecting to the database. This is the workspace that will be used for all queries and DML operations until the <a href="#">GotoWorkspace</a> procedure is explicitly executed. The default is the LIVE workspace. Any workspace specified must have the ACCESS_WORKSPACE privilege granted to PUBLIC.</p>

**Table 1-6 (Cont.) Workspace Manager System Parameters**

Parameter Name	Values
FIRE_TRIGGERS_FOR_NONDML_EVENTS	<p>ON (the default) causes user-defined triggers on version-enabled tables to be fired when a workspace non-DML operation (such as <a href="#">MergeWorkspace</a> or <a href="#">MergeTable</a>) is executed, unless later overridden for specific triggers by the <a href="#">SetTriggerEvents</a> procedure.</p> <p>OFF causes user-defined triggers on version-enabled tables not to be fired when a workspace non-DML operation (such as <a href="#">MergeWorkspace</a> or <a href="#">MergeTable</a>) is executed, unless later overridden for specific triggers by the <a href="#">SetTriggerEvents</a> procedure.</p>
KEEP_REMOVED_WORKSPACES_INFO	<p>ON keeps information about any workspaces that are removed. This information will be available in the <code>DBA_REMOVED_WORKSPACES</code>, <code>ALL_REMOVED_WORKSPACES</code>, and <code>USER_REMOVED_WORKSPACES</code> views. ON also adds two new columns to <code>xxx_HIST</code> views (described in <a href="#">xxx_HIST Views</a>): <code>WM_CREATEWORKSPACEID</code> and <code>WM_RETIREWORKSPACEID</code> can be used to determine the workspace from which a row was merged and the workspace that merged a row that caused the row to be retired.</p> <p>OFF (the default) does not keep this information about any workspaces that are removed.</p>
NONCR_WORKSPACE_MODE	<p>OPTIMISTIC_LOCKING (the default for both new installations and upgrades) allows a record to be edited in two or more workspaces that are not continually refreshed. If differences occur between parent and child workspaces, the record is considered to be in conflict, and the conflict must be resolved before the child workspace can be merged or refreshed.</p> <p>PESSIMISTIC_LOCKING does not allow a record to be edited in two or more workspaces that are not continually refreshed. This setting ensures that there are no conflicts between parent and child workspaces.</p>
NUMBER_OF_COMPRESS_BATCHES	<p>A number from 1 to 1000, identifying the number of batches to be used when the <code>batch_size</code> parameter value is <code>PRIMARY_KEY_RANGE</code> and general statistics, but not histogram statistics, are available for a primary key column of type <code>NUMBER</code>, <code>INTEGER</code>, <code>DATE</code>, or <code>TIMESTAMP</code>. (See the reference information for any <code>DBMS_WM</code> subprogram that has a <code>batch_size</code> parameter.)</p>

Table 1-6 (Cont.) Workspace Manager System Parameters

Parameter Name	Values
REMOVEWORKSPACE_DEFERRED	<p>Specifies the default value for the <code>defer_option</code> parameter of the <a href="#">RemoveWorkspace</a> and <a href="#">RemoveWorkspaceTree</a> procedures.</p> <p>OFF (the default) causes the <a href="#">RemoveWorkspace</a> and <a href="#">RemoveWorkspaceTree</a> operations to completely remove any rows or locks associated with the removed workspace from any version-enabled table that was modified in the workspace.</p> <p>FAST removes the workspace in its entirety, so that it is no longer available for use. However, any rows associated with the workspace that are stored in a version-enabled table are not deleted. Keeping the rows means locks that were associated with the workspace might not be released. These locks, along with the versioned rows, will remain until the <a href="#">RemoveDeferredWorkspaces</a> procedure is executed.</p> <p>REMOVE_LOCKS, like FAST, removes the workspace in its entirety, and any rows associated with the workspace that are stored in a version-enabled table are not deleted. Unlike with FAST, though, any locks that could prevent other users in different workspaces from modifying or obtaining a lock on a row <b>are</b> released. The rows that are contained within version-enabled tables that are part of the workspace will not be removed until the <a href="#">RemoveDeferredWorkspaces</a> procedure is executed.</p>
ROW_LEVEL_LOCKING	<p>ON takes row-level locks on any rows that need to be merged or refreshed during <a href="#">MergeTable</a>, <a href="#">MergeWorkspace</a>, or <a href="#">RefreshWorkspace</a> operations. This allows these procedures to run in parallel for workspaces that share the same parent. It allows multiple workspaces or session threads to issue merge or refresh requests concurrently against the same table; it does not parallelize a single merge or refresh operation that includes a list of tables. (However, parallel merge and refresh operations are supported only for tables without valid time support, and only for workspaces that are not continually refreshed.) The ON setting does not take effect for continually refreshed workspaces or if any of the tables that need to be merged or refreshed have valid time support enabled.</p> <p>OFF (the default) takes workspace-level locks for all workspace operations. This setting prevents <a href="#">MergeTable</a>, <a href="#">MergeWorkspace</a>, and <a href="#">RefreshWorkspace</a> operations from running in parallel while operating on the same parent workspace.</p>

Table 1-6 (Cont.) Workspace Manager System Parameters

Parameter Name	Values
TARGET_PGA_MEMORY	A number representing the maximum amount of memory, specified in bytes, that should be used for selecting rows into memory during any <a href="#">MergeTable</a> , <a href="#">MergeWorkspace</a> , or <a href="#">RefreshWorkspace</a> operation. The default is 8388608 (8 megabytes). Workspace Manager uses this value to determine the optimal number of rows to fetch at any one time. This value does not affect the amount of memory used by other database processes, but only internal workspace operations.
UNDO_SPACE	A string containing <code>UNLIMITED</code> (for no specified limit) or a number representing the maximum number of bytes for undo space available for Workspace Manager operations. Example: '1048576' for 1 megabyte. Workspace manager tries to minimize the amount of undo space used in a single transaction so as not to exceed the <code>UNDO_SPACE</code> value.  You can override the value of the <code>UNDO_SPACE</code> system parameter by specifying the <code>undo_space</code> parameter in the call to the <a href="#">EnableVersioning</a> procedure.
USE_SCALAR_TYPES_FOR_VALIDTIME	<code>ON</code> causes Workspace Manager to use two columns, named <code>WM_VALIDFROM</code> and <code>WM_VALIDTILL</code> , of type <code>TIMESTAMP WITH TIME ZONE</code> , instead of a single column named <code>WM_VALID</code> (of type <code>WM_PERIOD</code> ) to indicate the valid time range in views created on a version-enabled table that has valid time support. (The <code>WM_PERIOD</code> type is described in <a href="#">WM_PERIOD Data Type</a> .)  <code>OFF</code> (the default) causes Workspace Manager to a single column named <code>WM_VALID</code> (of type <code>WM_PERIOD</code> ) to indicate the valid time range in views created on a version-enabled table that has valid time support.  This parameter affects only tables that are subsequently version-enabled; it does not affect the views on existing version-enabled tables. To change the views on an existing version-enabled table, use the <a href="#">AlterVersionedTable</a> procedure and specify the <code>alter_option</code> parameter value <code>USE_SCALAR_TYPES_FOR_VALIDTIME</code> or <code>USE_WM_PERIOD_FOR_VALIDTIME</code> .



**Table 1-6 (Cont.) Workspace Manager System Parameters**

Parameter Name	Values
USE_TIMESTAMP_TYPE_FOR_HISTORY	<p>ON (the default) causes Workspace Manager, if the Oracle database release is 9.0.1 or later, to use the <code>TIMESTAMP WITH TIME ZONE</code> type for <code>WM_CREATETIME</code> and <code>WM_RETIRETIME</code> columns.</p> <p>OFF causes Workspace Manager to use the <code>DATE</code> type for <code>WM_CREATETIME</code> and <code>WM_RETIRETIME</code> columns.</p>

## 1.6 Import and Export Considerations

Workspace Manager supports the import and export of version-enabled tables in any of the following ways: a full database import and export, an import and export that only includes the schemas required by Workspace Manager, or a workspace-level import and export through Workspace Manager procedures.

No other export modes, such as single schema, table, or partition level, are currently supported.

Full database import and export operations can be performed on version-enabled databases using the Oracle utilities; however, the following considerations and restrictions apply:

- A database with version-enabled tables can be exported to another Oracle database only if the other database has Workspace Manager installed and does not currently have any version-enabled tables or workspaces (that is, other than the `LIVE` workspace).
- For an import operation using the Oracle Data Pump Import utility, if the dump file includes the `WMSYS` schema, you must specify `table_exists_action=truncate`. If the dump file does not include the `WMSYS` schema, you can specify `table_exists_action=append` if the version-enabled tables being imported do not yet exist or are empty. (In general, dump files generated by Oracle Database release 10.2 or later will not include the `WMSYS` schema, while dump files generated by earlier releases will include the `WMSYS` schema.)

The dump files must be from compatible versions of Workspace Manager. In general, any dump file created with `VERSION=12` is capable of being supported.

- If you are using Data Pump Import, the dump file must have been created using Data Pump Export.
- The `REMAP_SCHEMA` capability in Data Pump Import utility is not supported with version-enabled databases.
- Workspace Manager no longer supports using the original Import and Export utilities for this mode.

Do not use the `SYS` schema when performing Workspace Manager import or export operations.

You can perform a limited (as opposed to full) export and import that includes all schemas related to version-enabled tables and workspaces, as well as any Workspace Manager metadata, but excludes all other schemas, as follows:

1. Call the [Export\\_Schemas](#) procedure to generate a dump file with the necessary objects and data.
2. Call the [Import\\_Schemas](#) procedure. (As with a full database import, Workspace Manager must already be installed and there must be no existing version-enabled tables or workspaces other than the `LIVE` workspace.)

For workspace-level export operations, each version-enabled table can be exported at the workspace level. Follow these steps to export a version-enabled table from one database into another database:

1. Call the [Export](#) procedure to store all of the data that needs to be exported into a staging table (for example, `t1`). The data that is exported can either be all of the data as seen from a particular workspace, savepoint, or instant, or only the data that was modified in the particular workspace. See the information about the [Export](#) procedure in [DBMS\\_WM Package: Reference](#) for more details.

 **Note:**

Tables with valid time support (that is, with a column named `WM_VALID` of type `WM_PERIOD`) are not supported with the [Export](#) procedure. (Valid time support is explained in [Workspace Manager Valid Time Support](#).)

To export multiple workspaces for a version-enabled table, call the [Export](#) procedure again, specifying the new workspace that needs to be exported as well as the original staging table. If you intend to import the data into a non-versioned table, specify the `versioned_db` parameter as `FALSE`.

2. Export the staging table (for example, `t1`), using the Oracle Data Pump Export utility or the original Export utility.
3. Import the staging table (for example, `t1`), using the Oracle Data Pump Import utility or the original Import utility, into the destination database.
4. If you are importing into a version-enabled table, call the [Import](#) procedure to move the data from the staging table to the version-enabled table, specifying the workspace where the data resided on the source database and the workspace into which the data should be stored.

The structure of the staging table must match that of the version-enabled table. By default, all enabled constraints must be validated before the import procedure successfully completes.

 **Note:**

For exporting or importing version-enabled **topologies**, see also the Usage Notes for the relevant `DBMS_WM` procedures, including [Export\\_Schemas](#) and [Initialize\\_After\\_Import](#).

## 1.7 Bulk Loading into Version-Enabled Tables

You can use SQL\*Loader to perform bulk loading into version-enabled tables, but you must also call some special Workspace Manager procedures, and some restrictions apply.

You can perform both direct-path and conventional-path bulk loading of data into either the latest version of any workspace or into the root version (version number 0, which is in the `LIVE` workspace). The root version is the ancestor of all other versions, so data in the root version is visible from all other workspaces (unless non-`LIVE` workspaces have updated the data).

Follow these general steps for bulk loading into a version-enabled table:

1. Call the [BeginBulkLoading](#) procedure to prepare the table for bulk loading. When data is being bulk loaded into a version-enabled table, DML and workspace operations on the table are not allowed, although workspace operations that do not involve this table are allowed. The [BeginBulkLoading](#) procedure prevents invalid operations from being performed on this table.
2. Use SQL\*Loader to perform the bulk loading. Only one line needs to be changed in the control file, to specify the `<table_name>_LT` name. For example, assume that the existing control file has the following line:

```
Load data into table departments (name, loc)
```

The line in the control file for bulk loading into the version-enabled table should be changed to:

```
Load data into table departments_LT (name, loc)
```

### Note:

In versions of Workspace Manager before 12.1, it was necessary to include the `wm_version` column. This column is now automatically populated by Workspace Manager, and an error will be generated if you populate it explicitly. This ensures that all the bulk-loaded rows will be tagged with the appropriate version, and that the other Workspace Manager-specific columns for these rows will have null values.

If the table was version-enabled with the history option, create and retire times can be bulk loaded into the `wm_createtime` and `wm_retiretime` columns of `<table_name>_LT`.

3. Complete the bulk loading process by calling either the [CommitBulkLoading](#) procedure to commit the bulk loading changes or the [RollbackBulkLoading](#) procedure to roll back the bulk loading changes.

If you commit the bulk loading changes, Workspace Manager ensures that the data is updated in the required workspace and version. By default, the bulk-loaded data is checked for each unique or referential constraint defined on the table, and any bulk-loaded rows that are in violation of any constraints are moved to a discards table specified as a parameter to the [CommitBulkLoading](#) procedure. If you specified to check for duplicates (that is, records in the data to be bulk loaded that have the same values in the primary key columns), for any duplicate records only the record with the lowest ROWID value is loaded into the table, and the rest are moved to the discards table.

The following restrictions apply to bulk loading with version-enabled tables in the current release:

- Bulk loading into a table with a self-referential integrity constraint is not allowed.
- Bulk loading into a workspace, other than `LIVE`, that has continually refreshed child workspaces is not allowed.
- Only the owner of a table or a user with the `WM_ADMIN` system privilege can bulk load into a version-enabled table.
- The user that is bulk loading the version-enabled table must have the `INSERT` privilege for `<table_name>_LT`.
- User-defined triggers on version-enabled tables are not executed during bulk loading.
- Session locking mode is not enforced for the bulk-loaded rows. Use the [LockRows](#) procedure to lock these rows.

## 1.8 DDL Operations Related to Version-Enabled Tables

To perform DDL (data definition language) operations on a version-enabled table, you must use special Workspace Manager procedures before and after the DDL operations, and you must specify the name of a special table created by Workspace Manager.

You cannot perform DDL operations in the usual manner on the table or any index or trigger that refers to the table. For example, to add a column to a table named `EMPLOYEES` that has been version-enabled, you cannot simply enter a statement in the form `ALTER TABLE EMPLOYEES ADD (column-name data-type)`.

The reason for these requirements is to ensure that Workspace Manager versioning metadata is updated to reflect the DDL changes. Therefore, DDL operations affecting a version-enabled table must be preceded by a call to the [BeginDDL](#) procedure, and must be concluded by a call to either the [CommitDDL](#) or [RollbackDDL](#) procedure. The [BeginDDL](#) procedure creates an empty temporary table with a name in the form `<table-name>_LTS` (the *S* standing for *skeleton*). The actual DDL statement must specify the name of the temporary `<table-name>_LTS` table, and must not specify the `<table-name>` or `<table-name>_LT` name. The [CommitDDL](#) and [RollbackDDL](#) procedures delete the temporary `<table-name>_LTS` table.



### Note:

An exception to this procedure is adding valid time support to an existing version-enabled table. To add valid time support, use the [AlterVersionedTable](#) procedure, as explained in [Adding Valid Time Support to an Existing Table](#).

The following DDL operations related to version-enabled tables are supported:

- **Table-related:** Modifying the following table properties: logging, pctfree, pctused, initrans, next, minextents, maxextents, pctincrease, freelists, and `buffer_pool`; adding and removing supplemental logging on the table; modifying the compression options on the table

- **Column-related:** ADD, DROP, MODIFY (but for MODIFY only the following operations: changing the default value of a column; changing the data type of a column that contains only null values or for which there are no existing data rows; changing the length of a column of type VARCHAR2, VARCHAR, CHAR, NCHAR, NVCHAR, or NVCHAR2; changing the scale or precision of a column of type NUMBER); renaming a column

Note that any new length, scale, or precision for a column should be adequate for any existing data in the column.

- **Index-related:** CREATE INDEX, DROP INDEX, ALTER INDEX (but for ALTER INDEX only the following options: logging, pctfree, initrans, initialextent, minextents, nextextent, maxextents, pctincrease, freelists, freelist groups, and buffer\_pool)

If the name of the index on a version-enabled table is longer than 26 characters, you must use the [AlterVersionedTable](#) procedure if you want to rename the index; you cannot use the ALTER INDEX statement with the RENAME clause. If the name of the index on a version-enabled table is 26 or fewer characters long, you can do either of the following to rename the index: use the [AlterVersionedTable](#) procedure, or use the ALTER INDEX statement with the RENAME clause between calls to the [BeginDDL](#) and [CommitDDL](#) procedures. See the Usage Notes for [AlterVersionedTable](#) for more information.

- **Trigger-related:** CREATE TRIGGER, DROP TRIGGER, ALTER TRIGGER ENABLE/DISABLE
- **Referential integrity constraint-related:** add, drop, enable, or disable a referential integrity constraint. For information about Workspace Manager referential integrity support, see [Referential Integrity Support](#).
- **Unique constraint-related:** add, drop, enable, or disable a unique constraint. For information about Workspace Manager unique constraint support, see [Unique Constraints](#).
- **Privilege-related:** grant table-level privileges to users and revoke these privileges from users.

You can create the following types of indexes on version-enabled tables: normal, bitmap, function-based normal, function-based bitmap, invisible, reverse, and domain. You cannot create or drop a partitioned or join index on a version-enabled table. (You can, however, version-enable a table that has a partitioned or join index.) You can use the `compress` and `prefix_length` parameters in index DDL operations.

Invisible columns are not supported in any DDL operations.

If an identity column is added to the table during a DDL session, there is no support for the `LIMIT VALUE` keyword. If that keyword is specified, the start value is reset based on the start value of the skeleton `_LTS` table.

If you try to perform an unsupported DDL operation, the change will not be made, and an exception might be raised by the [CommitDDL](#) procedure.

If the DDL operation involving a version-enabled table is on a domain index (for example, creating an R-tree index on the table), you must have the `CREATE TABLE` privilege.

If you need to perform DDL operations on a version-enabled table in an Oracle Label Security (OLS) environment, you can use the `apply_table_policy`, `remove_table_policy`, `enable_table_policy`, and `disable_table_policy` procedures of the `SA_POLICY_ADMIN` package on the skeleton (`_LTS`) table, and the changes will be transferred to the version-enabled table.

The following example shows the statements needed to add a column named `COMMENTS` to the `COLA_MARKETING_BUDGET` table by using the special table named

COLA\_MARKETING\_BUDGET\_LTS. It also includes a DESCRIBE statement to show the addition of the column.

### Example 1-2 DDL Operation on a Version-Enabled Table

```
EXECUTE DBMS_WM.BeginDDL('COLA_MARKETING_BUDGET');
ALTER TABLE cola_marketing_budget_lts ADD (comments VARCHAR2(100));
DESCRIBE cola_marketing_budget_lts;
```

Name	Null?	Type
PRODUCT_ID	NOT NULL	NUMBER
PRODUCT_NAME		VARCHAR2(32)
MANAGER		VARCHAR2(32)
BUDGET		NUMBER
COMMENTS		VARCHAR2(100)

```
EXECUTE DBMS_WM.CommitDDL('COLA_MARKETING_BUDGET');
```

In the preceding example, the ALTER TABLE statement specifies the COLA\_MARKETING\_BUDGET\_LTS table, which is created by the BeginDDL procedure. The CommitDDL procedure applies the change to the COLA\_MARKETING\_BUDGET table and deletes the COLA\_MARKETING\_BUDGET\_LTS table.

## 1.9 Constraint Support with Workspace Manager

This section describes Workspace Manager considerations relating to the use of database constraints.

- [Referential Integrity Support](#)
- [Unique Constraints](#)
- [SET NULL Constraints](#)

### 1.9.1 Referential Integrity Support

Version-enabled tables can have referential integrity constraints, including constraints with the CASCADE and RESTRICT options; however, the following considerations and restrictions apply:

- If the parent table in a referential integrity relationship is version-enabled, the child table must be version-enabled also. (The child table is the one on which the constraint is defined.) For example, consider the following EMPLOYEE and DEPARTMENT table definitions, with a foreign key constraint added after the creation (that is, the dept\_id value in each EMPLOYEE row must match an existing dept\_id value in a DEPARTMENT row).

```
CREATE TABLE employee (
  employee_id NUMBER,
  last_name VARCHAR2(32),
  first_name VARCHAR2(32),
  dept_id NUMBER);
CREATE TABLE department (
  dept_id NUMBER,
  name VARCHAR2(32);
ALTER TABLE employee ADD CONSTRAINT emp_forkey_deptid
```

```
FOREIGN KEY (dept_id) REFERENCES department (dept_id)
ON DELETE CASCADE;
```

In this example, `DEPARTMENT` is considered the parent and `EMPLOYEE` is considered the child in the referential integrity relationship; and if `DEPARTMENT` is version-enabled, `EMPLOYEE` must be version-enabled also. In this relationship definition, when a `DEPARTMENT` row is deleted, all its child rows in the `EMPLOYEE` table are deleted (cascading delete operation).

- A child table in a referential integrity relationship is allowed to be version-enabled without the parent table being version-enabled.
- The foreign key in a child table must refer to the primary key in the parent table.
- Primary key values in the parent table cannot be updated. For example, if `DEPARTMENT` is the parent table and `EMPLOYEE` is the child table, you cannot change the department ID of a department.
- Multilevel referential integrity constraints are permitted on version-enabled tables. For example, the table `EMPLOYEE(emp_id, dept_id)` could have the constraint that the department ID must exist in the table `DEPARTMENT(dept_id, dept_name, loc_id)`; and the table `DEPARTMENT(dept_id, dept_name, loc_id)` could have the constraint that the location ID must exist in the table `LOCATION(loc_id, loc_name)`. However, all tables that are involved in multilevel referential integrity constraints must be version-enabled and version-disabled together, unless all the referential integrity constraints involved have the `Restrict` rule. If all the constraints involved have the `Restrict` rule, you can version-enable the tables either all together or one at a time with child tables preceding their parent tables. The table names must be passed as a comma-delimited list to the [EnableVersioning](#) and [DisableVersioning](#) procedures.

Workspace Manager uses the static data dictionary views [ALL\\_WM\\_RIC\\_INFO](#) and [USER\\_WM\\_RIC\\_INFO](#) (described in [Workspace Manager Static Data Dictionary Views](#)) to hold information pertinent to referential integrity support.

If you need to add, drop, enable, or disable a referential integrity constraint that involves two tables, it is more convenient if you perform the operation before version-enabling the tables. However, you can add, drop, enable, or disable a referential integrity constraint that involves a version-enabled table if you follow these steps:

1. If the parent table has been version-enabled, begin a DDL session specifying the parent table.
2. Begin a DDL session specifying the child table.
3. Alter the `<table-name>_LTS` table for the child table to add the foreign key constraint. If a version-enabled table is the referenced table, specify `<table-name>_LTS` for the parent table. (See [DDL Operations Related to Version-Enabled Tables](#) for information about `<table-name>_LTS` tables and performing DDL operations on version-enabled tables.)
4. Commit the DDL changes specifying the child table.
5. If the parent table has been version-enabled, commit the DDL changes specifying the parent table.

**Example 1-3** adds a foreign key constraint. Assume that the `EMPLOYEE` and `DEPARTMENT` tables are version-enabled and are defined as follows:

```
EMPLOYEE(emp_id number primary key, dept_id number)
DEPARTMENT(dept_id number primary key, dept_name varchar2(30))
```



**Example 1-3 Adding a Referential Integrity Constraint**

```
-- Begin a DDL session on the parent table.
DBMS_WM.BeginDDL('DEPARTMENT');

-- Begin a DDL session on the child table.
DBMS_WM.BeginDDL('EMPLOYEE');

-- Add the constraint between EMPLOYEE_LTS and DAPATMENT_LTS.
ALTER TABLE employee_lts ADD CONSTRAINT employee_fk FOREIGN KEY (dept_id)
    REFERENCES department_lts(dept_id);

-- Commit DDL on the child table (transfers the constraint on employee_lts
-- to employee and drops employee_lts).
EXECUTE DBMS_WM.CommitDDL('EMPLOYEE');

-- Commit DDL on the parent table (drops the department_lts table).
EXECUTE DBMS_WM.CommitDDL('DEPARTMENT');
```

If you are in a DDL session (that is, if you have called the [BeginDDL](#) procedure), you cannot add, drop, enable, or disable a referential integrity constraint that involves two tables if one table is version-enabled and the other is not version-enabled. Both tables must be version-enabled.

- [Locking with DML Operations on Tables with Referential Integrity Constraints](#)

### 1.9.1.1 Locking with DML Operations on Tables with Referential Integrity Constraints

When data manipulation language (DML) operations are performed on version-enabled tables that have referential integrity constraints, Workspace Manager acquired shared locks so that the following conditions are enforced:

- During an insert operation or an update operation affecting the foreign key column on the child table, delete operations cannot be performed on the parent table. For example, if `DEPARTMENT` is the parent table and `EMPLOYEE` is the child table, during the time that a new employee is being added or an existing employee is being assigned to a different department, no departments can be deleted.
- During a delete operation on the parent table, insert operations or updates operation affecting the foreign key column cannot be performed on the child table. For example, during the time that a department is being deleted, new employees cannot be added and existing employees cannot be assigned to different departments.

**Note:**

For general information about locking performed by Workspace Manager, including explanations of shared and exclusive locks, see [Lock Management with Workspace Manager](#).

Multiple sessions can simultaneously perform either of the following, but not both of the following, DML operations simultaneously:



- Insert operations or update operations affecting the foreign key column on the child table
- Delete operations on the parent table

Multiple sessions can simultaneously perform any of the following Workspace Manager operations simultaneously:

- Use the [MergeTable](#) procedure to apply changes to a child table or parent table in different workspaces.
- Use the [MergeTable](#) procedure to apply changes to a child table in one workspace, and insert or update the child table in another workspace.
- Use the [MergeTable](#) procedure to apply changes to a parent table in one workspace, and delete from the parent table in another workspace.

One session will be blocked until the other session finishes in the following situations:

- A session tries to merge changes to a child table in one workspace, and another session tries to merge changes to the parent table in another workspace.
- A session tries to merge changes to a child table in one workspace, and another session tries to delete from the parent table.
- A session tries to merge changes to a parent table in one workspace, and another session tries to insert into a child table or change a value in the foreign key of a child table.

## 1.9.2 Unique Constraints

Tables with unique constraints defined on them can be version-enabled. The following are supported:

- `UNIQUE` constraint on a single column or multiple columns
- Unique index on a single column or multiple columns
- Functional unique index on the table

The treatment of null values is the same for version-enabled tables as for tables that are not version-enabled.

Workspace Manager uses the following static data dictionary views (described in [Workspace Manager Static Data Dictionary Views](#)) to hold information pertinent to support for unique constraints:

- [ALL\\_WM\\_CONSTRAINTS](#) and [USER\\_WM\\_CONSTRAINTS](#) contain information about columns in unique constraints on version-enabled tables.
- [ALL\\_WM\\_CONS\\_COLUMNS](#) and [USER\\_WM\\_CONS\\_COLUMNS](#) contain information about constraints on version-enabled tables.
- [ALL\\_WM\\_IND\\_COLUMNS](#) and [USER\\_WM\\_IND\\_COLUMNS](#) contain information about indexes used for enforcing unique constraints on version-enabled tables.
- [ALL\\_WM\\_IND\\_EXPRESSIONS](#) and [USER\\_WM\\_IND\\_EXPRESSIONS](#) contain information about functional expressions on functional unique indexes on version-enabled tables.

## 1.9.3 SET NULL Constraints

SET NULL constraints are not supported by Workspace Manager. If a table has any SET NULL constraints, they are converted to the RESTRICT option when the table is version-enabled.

For example, the constraint ON DELETE SET NULL is converted to ON DELETE RESTRICT.

## 1.10 Triggers on Version-Enabled Tables

Version-enabled tables can have triggers defined; however, the following considerations and restrictions apply.

- Only per-row triggers are supported. Per-statement triggers are not supported.
- The only call-out supported is to PL/SQL procedures. That is, the `action_type` must be PL/SQL.

Any triggers that are not supported for version-enabled tables are deactivated when versioning is enabled, and are activated when versioning is disabled.

You can selectively enable specific user-defined triggers for certain kinds of events by using the [SetTriggerEvents](#) procedure.

## 1.11 Virtual Private Database Considerations

You can use Workspace Manager in conjunction with the Oracle Virtual Private Database (VPD) technology.

Virtual private databases are described in *Oracle Database Security Guide*. However, the following considerations apply Workspace Manager in a VPD:

- Row-level security policies are not enforced during workspace operations, such as [MergeWorkspace](#). A call to [MergeWorkspace](#) will merge all the changes made in a workspace, not just the changes that the current user can see. You can use Workspace Manager privileges (such as `MERGE_WORKSPACE`) to control workspace operations.
- Row-level security policies cannot be defined on a version-enabled table by defining them only on the specified table (`<table_name>`). Instead, you must define row-level security policies on all of the following that exist: `<table_name>`, `<table_name>_LOCK`, `<table_name>_CONF`, `<table_name>_DIFF`, and `<table_name>_HIST`. Do not use the Workspace Manager DDL framework described in [DDL Operations Related to Version-Enabled Tables](#) (that is, do not use the [BeginDDL](#) and [CommitDDL](#) procedures) when defining row-level security policies.

## 1.12 Support for Table Synonyms

For any Workspace Manager procedure or function input parameter that calls for a table name, you can instead specify a synonym.

When Workspace Manager looks for a table, it searches in the following sequence and uses the first match for the specified name:

1. A table in the specified schema (or local schema if no schema is specified)
2. A private synonym in the specified schema (or local schema if no schema is specified)
3. A public synonym

## 1.13 Materialized View Support

This section describes considerations for using Workspace Manager with materialized views.

You can create a materialized view on a version-enabled table only if you specify the complete refresh method (`REFRESH COMPLETE`) when you create the materialized view. You cannot specify any of the following clauses in the `CREATE MATERIALIZED VIEW` statement:

- `FAST` (incremental refresh)
- `ON COMMIT`
- `FOR UPDATE`

You cannot version-enable a materialized view or the base table of a materialized view.

When the materialized view is created, its content is based on the workspace in which the session is at that time. When the materialized view is refreshed, its content is based on the workspace in which the session is when the `DBMS_MVIEW.REFRESH` operation is performed. When the materialized view is created or refreshed, it shows the same data in all workspaces.

## 1.14 Spatial and Graph Topology Support

This section describes special considerations and techniques for using Workspace Manager with tables in Oracle Spatial and Graph topologies,

Topologies are documented in *Oracle Spatial and Graph Topology Data Model and Network Data Model Graph Developer's Guide*.

A topology consists of feature tables, as well as tables with names in the form `<topology-name>_NODE$`, `<topology-name>_EDGE$`, `<topology-name>_FACE$`, `<topology-name>_RELATION$`, and `<topology-name>_HISTORY$`. If you want to version-enable any topology tables, you must version-enable all tables associated with the topology. To do so, you must specify the topology name as the `table_name` parameter to the [EnableVersioning](#) procedure, and you must specify the `isTopology` parameter as `TRUE`. For example:

```
EXECUTE DBMS_WM.EnableVersioning(table_name => 'xyz_topo', isTopology => TRUE);
```

The preceding example version-enables the `xyz_topo` topology; that is, it version-enables all feature tables associated with the `xyz_topo` topology, as well as the `XYZ_TOPO_NODE$`, `XYZ_TOPO_FACE$`, `XYZ_TOPO_EDGE$`, `XYZ_TOPO_RELATION$`, and `XYZ_TOPO_HISTORY$` tables.

A version-enabled topology must have at least one feature table.

To disable versioning on any topology tables, you must disable versioning on all tables associated with the topology by specifying the topology name as the `table_name` parameter to the [DisableVersioning](#) procedure and the `isTopology` parameter as `TRUE`.

However, exceptions apply to the preceding guidelines about version-enabling and version-disabling topology tables in the following cases:

- If a feature table of a topology is the child table of a referential integrity constraint with `CASCADE` option with a table that is not in the topology
- If a feature table of a topology is the parent table of a referential integrity constraint with a table that is not in the topology

In these cases, you must version-enable or version-disable the feature table separately. That is, first call the [EnableVersioning](#) or [DisableVersioning](#) procedure on the feature table (along with any tables required by the referential integrity constraint), and then invoke the [EnableVersioning](#) or [DisableVersioning](#) procedure specifying the topology name.

- [Locking Considerations with Topologies](#)
- [Additional Considerations with Topologies](#)

### 1.14.1 Locking Considerations with Topologies

To lock or unlock rows in tables associated with a topology, you must specify the topology name as the `table_name` parameter to the [LockRows](#) or [UnlockRows](#) procedure, and you must identify the window containing the rows by using the `Xmin`, `Ymin`, `Xmax`, and `Ymax` parameters. You must also not specify the `where_clause` parameter. For example:

```
EXECUTE DBMS_WM.LockRows (workspace => 'ws1', table_name => 'xyz_topo', Xmin => 0.1, Ymin => 0.1, Xmax => 0.5, Ymax => 0.5 );
```

The preceding example puts version locks on all the rows of the specified topology contained in the specified window. To edit the elements of a topology in a workspace (including the `LIVE` workspace), follow these steps:

1. Invoke the [LockRows](#) procedure to put version locks on all the elements of the topology contained in a window of interest.
2. Invoke the Oracle Spatial and Graph Topology Java client `loadWindow` method for the same window of interest.

### 1.14.2 Additional Considerations with Topologies

The following additional considerations apply to using Workspace Manager with Spatial and Graph topologies:

- You must invoke the `SDO_TOPO.INITIALIZE_METADATA` procedure at least once on a topology before you version-enable the tables associated with the topology. (You can also invoke the `SDO_TOPO.INITIALIZE_METADATA` procedure as needed after version-enabling a topology.)
- Do not use the [MergeTable](#), [RefreshTable](#), or [RollbackTable](#) procedure on a version-enabled table associated with a topology. Instead, use the [MergeWorkspace](#), [RefreshWorkspace](#), or [RollbackWorkspace](#) procedure to merge, refresh, or roll back tables associated with a topology.

## 1.15 Workspace Manager Reserved Words and Characters

Because Workspace Manager creates internal objects using its own naming conventions, you must avoid some words and characters in the names for certain kinds of objects.

Table 1-7 lists kinds of objects and restrictions that apply to their names. (See also the name length guidelines in Table 1-2 in [Infrastructure for Version-Enabling of Tables](#).)

**Table 1-7 Workspace Manager Reserved Words and Characters**

Object	Name Cannot Be Any of the Following
Workspace	BASE, LIVE, or a string containing any of the following characters: / (slash), * (asterisk), , (comma), \$ (dollar sign), # (pound sign)
Column in a version-enabled table	A string starting with WM\$ or WM_
Index on a version-enabled table	A string of the form <table_name>_PKI\$ or <table_name>_TI\$ based on the name of the table on which the index exists

## 1.16 DBMS\_WM Subprogram Categories

The Workspace Manager application programming interface (API) consists of PL/SQL subprograms (procedures and functions) in a single PL/SQL package named `DBMS_WM`.

The subprograms can be logically grouped into the categories described in this section.

### Note:

Most Workspace Manager subprograms are procedures, but a few are functions. (A function returns a value; a procedure does not return a value.)

Most functions have names starting with *Get* (such as [GetConflictWorkspace](#) and [CreateSavepoint](#)).

Reference information for all subprograms is in [DBMS\\_WM Package: Reference](#).

- [Table Management Subprograms](#)
- [Workspace Management Subprograms](#)
- [Savepoint Management Subprograms](#)
- [Privilege Management Subprograms](#)
- [Lock Management Subprograms](#)
- [Conflict Management Subprograms](#)
- [Bulk Load Support Subprograms](#)

### 1.16.1 Table Management Subprograms

Table management subprograms enable and disable workspace management on a table, and perform other table-related operations.

Table 1-8 shows the subprograms available for table management.

**Table 1-8 Table Management Subprograms**

Procedure	Description
<a href="#">EnableVersioning</a>	Version-enables a table, creating the necessary structures to enable the table to support multiple versions of rows.
<a href="#">DisableVersioning</a>	Deletes all support structures that were created to enable the table to support versioned rows.
<a href="#">SetWoOverwriteOFF</a>	Disables the <code>VIEW_WO_OVERWRITE</code> history option that was enabled by the <a href="#">EnableVersioning</a> or <a href="#">SetWoOverwriteON</a> procedure, changing the option to <code>VIEW_W_OVERWRITE</code> (with overwrite).
<a href="#">SetWoOverwriteON</a>	Enables the <code>VIEW_WO_OVERWRITE</code> history option that was disabled by the <a href="#">SetWoOverwriteOFF</a> procedure.
<a href="#">BeginDDL</a>	Starts a DDL (data definition language) session for a specified table.
<a href="#">CommitDDL</a>	Commits DDL changes made during a DDL session for a specified table, and ends the DDL session.
<a href="#">RollbackDDL</a>	Rolls back (cancels) DDL changes made during a DDL session for a specified table, and ends the DDL session.
<a href="#">RecoverAllMigratingTables</a>	Attempts to complete the migration process on a table that was left in an inconsistent state after the Workspace Manager migration procedure failed.
<a href="#">RecoverAllMigratingTables</a>	Attempts to complete the migration process on all tables that were left in an inconsistent state after the Workspace Manager migration procedure failed.
<a href="#">CopyForUpdate</a>	Allows LOB columns (BLOB, CLOB, or NCLOB) in version-enabled tables to be modified.
<a href="#">Export</a>	Exports data from a version-enabled table (all rows, or as limited by any combination of several parameters) to a staging table.
<a href="#">Import</a>	Imports data from a staging table (all rows, or as limited by any combination of several parameters) into a version-enabled table in a specified workspace.

## 1.16.2 Workspace Management Subprograms

Workspace management subprograms perform operations on workspaces.

[Table 1-9](#) shows the subprograms available for workspace management.

**Table 1-9 Workspace Management Subprograms**

Procedure	Description
<a href="#">CreateWorkspace</a>	Creates a new workspace in the database.
<a href="#">GotoWorkspace</a>	Moves the current session to the specified workspace.
<a href="#">SetDiffVersions</a>	Finds differences in values in version-enabled tables for two savepoints and their common ancestor (base). It creates rows in the differences views describing these differences.

**Table 1-9 (Cont.) Workspace Management Subprograms**

Procedure	Description
<a href="#">GetDiffVersions</a>	Returns the names of the (workspace, savepoint) pairs on which the session has performed the <a href="#">SetDiffVersions</a> operation.
<a href="#">MergeTable</a>	Applies changes to a table (all rows or as specified in the <code>WHERE</code> clause) in a workspace to its parent workspace.
<a href="#">MergeWorkspace</a>	Applies all changes in a workspace to its parent workspace, and optionally removes the workspace.
<a href="#">RollbackWorkspace</a>	Discards all data changes made in the workspace to version-enabled tables.
<a href="#">RollbackTable</a>	Discards all changes made in the workspace to a specified table (all rows or as specified in the <code>WHERE</code> clause).
<a href="#">RollbackToSP</a>	Discards all data changes made in the workspace to version-enabled tables since the specified savepoint.
<a href="#">RefreshTable</a>	Applies to a workspace all changes made to a table (all rows or as specified in the <code>WHERE</code> clause) in its parent workspace.
<a href="#">RefreshWorkspace</a>	Applies to a workspace all changes made in its parent workspace.
<a href="#">AlterWorkspace</a>	Modifies the description of a workspace.
<a href="#">ChangeWorkspaceType</a>	Changes a workspace that is not continually refreshed to be continually refreshed.
<a href="#">RemoveWorkspace</a>	Discards all row versions associated with a workspace and deletes the workspace.
<a href="#">RemoveWorkspaceTree</a>	Discards all row versions associated with a workspace and its descendant workspaces, and deletes the affected workspaces.
<a href="#">FreezeWorkspace</a>	Restricts access to a workspace and the ability of users to make changes in the workspace.
<a href="#">UnfreezeWorkspace</a>	Enables access and changes to a workspace, reversing the effect of the <a href="#">FreezeWorkspace</a> procedure.
<a href="#">CompressWorkspace</a>	Deletes removable savepoints in a workspace, and minimizes the Workspace Manager metadata structures for the workspace.
<a href="#">CompressWorkspaceTree</a>	Deletes removable savepoints in a workspace and all its descendant workspaces. It also minimizes the Workspace Manager metadata structures for the affected workspaces, and eliminates any redundant data that might arise from the deletion of the savepoints.
<a href="#">IsWorkspaceOccupied</a>	Checks whether or not a workspace has any active sessions.
<a href="#">CreateSavepoint</a>	Returns the current workspace for the session.
<a href="#">SetMultiWorkspaces</a>	Makes the specified workspace or workspaces visible in the multiworkspace views for version-enabled tables.
<a href="#">GetMultiWorkspaces</a>	Returns the names of workspaces visible in the multiworkspace views for version-enabled tables.
<a href="#">GetOpContext</a>	Returns the context of the current operation for the current session.
<a href="#">AddAsParentWorkspace</a>	Adds a workspace as a parent workspace to a child workspace in a multiparent workspace environment.

**Table 1-9 (Cont.) Workspace Management Subprograms**

Procedure	Description
<a href="#">RemoveAsParentWorkspace</a>	Removes a workspace as a parent workspace in a multiparent workspace environment.

### 1.16.3 Savepoint Management Subprograms

Savepoint management subprograms perform operations related to savepoints.

[Table 1-10](#) shows the subprograms available for savepoint management.

**Table 1-10 Savepoint Management Subprograms**

Procedure	Description
<a href="#">CreateSavepoint</a>	Creates a savepoint for the current version.
<a href="#">GotoSavepoint</a>	Goes to the specified savepoint in the current workspace.
<a href="#">GotoDate</a>	Goes to a point at or near the specified date and time in the current workspace.
<a href="#">GetSessionInfo</a>	Retrieves information about the current workspace and session context; useful for finding the session's current savepoint or instant in time.
<a href="#">AlterSavepoint</a>	Modifies the description of a savepoint.
<a href="#">DeleteSavepoint</a>	Deletes a savepoint and associated rows in version-enabled tables.

### 1.16.4 Privilege Management Subprograms

Privilege management subprograms grant and revoke Workspace Manager privileges.

[Table 1-11](#) shows the subprograms available for privilege management.

**Table 1-11 Privilege Management Subprograms**

Procedure	Description
<a href="#">GrantWorkspacePriv</a>	Grants workspace-level privileges to users, roles, or PUBLIC.
<a href="#">RevokeWorkspacePriv</a>	Revokes workspace-level privileges from users and roles.
<a href="#">GrantSystemPriv</a>	Grants privileges on all workspaces to users, roles, or PUBLIC.
<a href="#">RevokeSystemPriv</a>	Revokes system-level privileges from users and roles.
<a href="#">GetPrivs</a>	Returns a comma-delimited list of all privileges that the current user has for the specified workspace.

### 1.16.5 Lock Management Subprograms

Lock management subprograms control Workspace Manager locking.

[Table 1-12](#) shows the subprograms available for lock management.



**Table 1-12 Lock Management Subprograms**

Procedure	Description
<a href="#">SetLockingON</a>	Enables Workspace Manager locking for the current session.
<a href="#">SetLockingOFF</a>	Disables Workspace Manager locking for the current session.
<a href="#">SetWorkspaceLockModeON</a>	Enables Workspace Manager locking for the specified workspace.
<a href="#">SetWorkspaceLockModeOFF</a>	Disables Workspace Manager locking for the specified workspace.
<a href="#">GetLockMode</a>	Returns the locking mode for the current session, which determines whether or not access is enabled to versioned rows and corresponding rows in the previous version.
<a href="#">LockRows</a>	Controls access to versioned rows in a specified table and to corresponding rows in the parent workspace.
<a href="#">UnlockRows</a>	Enables access to versioned rows in a specified table and to corresponding rows in the parent workspace.

## 1.16.6 Conflict Management Subprograms

Conflict management subprograms detect and resolve conflicts between workspaces.

[Table 1-13](#) shows the subprograms available for conflict management.

**Table 1-13 Conflict Management Subprograms**

Procedure	Description
<a href="#">SetConflictWorkspace</a>	Determines whether or not conflicts exist between a workspace and its parent workspace.
<a href="#">GetConflictWorkspace</a>	Returns the name of the workspace on which the session has performed the <a href="#">SetConflictWorkspace</a> procedure.
<a href="#">BeginResolve</a>	Starts a conflict resolution session.
<a href="#">ResolveConflicts</a>	Resolves conflicts between workspaces.
<a href="#">CommitResolve</a>	Ends a conflict resolution session and saves (makes permanent) any changes in the workspace since the <a href="#">BeginResolve</a> procedure was executed.
<a href="#">RollbackResolve</a>	Quits a conflict resolution session and discards all changes in the workspace since the <a href="#">BeginResolve</a> procedure was executed.

## 1.16.7 Bulk Load Support Subprograms

Bulk load support subprograms enable SQL\*Loader to be used for bulk loading data into version-enabled tables, as explained in [Bulk Loading into Version-Enabled Tables](#).

[Table 1-14](#) shows the subprograms available for bulk loading support.

**Table 1-14 Bulk Loading Support Subprograms**

Procedure	Description
<a href="#">GetBulkLoadVersion</a>	Returns a version number to be specified when you call the <a href="#">BeginBulkLoading</a> procedure.
<a href="#">BeginBulkLoading</a>	Starts the bulk loading process for a version-enabled table.
<a href="#">CommitBulkLoading</a>	Ends the bulk loading process for a version-enabled table by committing the bulk load changes.
<a href="#">RollbackBulkLoading</a>	Rolls back changes made to a version-enabled table during a bulk load operation.

## 1.17 Simplified Examples Using Workspace Manager

This topic presents two simplified examples of using Workspace Manager to try out some scenarios and select one of them.

Each example uses workspaces and one or more savepoints. One example (in [Example: Warehouse Expansion Options](#)) uses the `OE.WAREHOUSES` table in the Oracle sample schemas.

The examples refer to concepts that were explained in this chapter, and they use procedures documented in [DBMS\\_WM Package: Reference](#).

- [Example: Marketing Budget Options](#)
- [Example: Warehouse Expansion Options](#)

### 1.17.1 Example: Marketing Budget Options

In [Example 1-4](#), a soft drink (cola) manufacturer has four products, each with a marketing manager and a marketing budget. Because of an exceptional opportunity for growth in the market for one product (`cola_b`), the company wants to do *what-if* analyses involving different managers and budget amounts.

#### Example 1-4 Marketing Budget Options

```
-----
-- INITIAL SET-UP
-----
-- Create the user for schema objects.
CREATE USER wm_developer IDENTIFIED BY password;

-- Grant regular privileges.
GRANT create session,
      unlimited tablespace,
      create table
TO wm_developer;

-- Grant WM-specific privileges (with grant_option = YES).
EXECUTE DBMS_WM.GrantSystemPriv ('ACCESS_ANY_WORKSPACE, MERGE_ANY_WORKSPACE,
      CREATE_ANY_WORKSPACE, REMOVE_ANY_WORKSPACE, ROLLBACK_ANY_WORKSPACE',
      'wm_developer', 'YES');
-----
-- CREATE AND POPULATE DATA TABLE --
```

```
-----  
CONNECT wm_developer  
-- Enter password when prompted.  
  
-- Cleanup: remove B_focus_2 workspace if it exists from previous run.  
EXECUTE DBMS_WM.RemoveWorkspace ('B_focus_2');  
  
-- Create a table for the annual marketing budget for  
-- several cola (soft drink) products.  
-- Each row will contain budget data for a specific  
-- product. Note: This table does not reflect recommended  
-- database design. (For example, a manager ID should  
-- be used, not a name.) It is deliberately oversimplified  
-- for purposes of illustration.  
  
CREATE TABLE cola_marketing_budget (  
  product_id NUMBER PRIMARY KEY,  
  product_name VARCHAR2(32),  
  manager VARCHAR2(32), -- Here a name, just for simplicity  
  budget NUMBER -- Budget in millions of dollars. Example: 3 = $3,000,000.  
);  
  
-- Version-enable the table. Specify hist option of VIEW_WO_OVERWRITE so that  
-- the COLA_MARKETING_BUDGET_HIST view contains complete history information  
-- about data changes.  
EXECUTE DBMS_WM.EnableVersioning ('cola_marketing_budget', 'VIEW_WO_OVERWRITE');  
  
INSERT INTO cola_marketing_budget VALUES (  
  1,  
  'cola_a',  
  'Alvarez',  
  2.0  
);  
INSERT INTO cola_marketing_budget VALUES (  
  2,  
  'cola_b',  
  'Baker',  
  1.5  
);  
INSERT INTO cola_marketing_budget VALUES (  
  3,  
  'cola_c',  
  'Chen',  
  1.5  
);  
INSERT INTO cola_marketing_budget VALUES (  
  4,  
  'cola_d',  
  'Davis',  
  3.5  
);  
COMMIT;  
  
-- Relevant data values now in LIVE workspace:  
-- 1, cola_a, Alvarez, 2.0  
-- 2, cola_b, Baker, 1.5  
-- 3, cola_c, Chen, 1.5  
-- 4, cola_d, Davis, 3.5  
  
-----  
-- CREATE WORKSPACES --
```

```

-----
-- Create workspaces for the following scenario: a major marketing focus
-- for the cola_b product. Managers and budget amounts for each
-- product can change, but the total marketing budget cannot grow.
--
-- One scenario (B_focus_1) features a manager with more expensive
-- plans (which means more money taken from other products' budgets).
-- The other scenario (B_focus_2) features a manager with less expensive
-- plans (which means less money taken from other products' budgets).
--
-- Two workspaces (B_focus_1 and B_focus_2) are created as child workspaces
-- of the LIVE database workspace.

EXECUTE DBMS_WM.CreateWorkspace ('B_focus_1');
EXECUTE DBMS_WM.CreateWorkspace ('B_focus_2');

-----
-- WORK IN FIRST WORKSPACE --
-----

-- Enter the B_focus_1 workspace and change the cola_b manager to Beasley and
-- raise the cola_b budget amount by 1.5 to bring it to 3.0. Reduce all other
-- products' budget amounts by 0.5 to stay within the overall budget.

EXECUTE DBMS_WM.GotoWorkspace ('B_focus_1');
UPDATE cola_marketing_budget
   SET manager = 'Beasley' WHERE product_name = 'cola_b';
UPDATE cola_marketing_budget
   SET budget = 3 WHERE product_name = 'cola_b';
UPDATE cola_marketing_budget
   SET budget = 1.5 WHERE product_name = 'cola_a';
UPDATE cola_marketing_budget
   SET budget = 1 WHERE product_name = 'cola_c';
UPDATE cola_marketing_budget
   SET budget = 3 WHERE product_name = 'cola_d';
COMMIT;

-- Relevant data values now in B_focus_1 workspace::
-- 1, cola_a, Alvarez, 1.5
-- 2, cola_b, Beasley, 3.0
-- 3, cola_c, Chen, 1.0
-- 4, cola_d, Davis, 3.0

-- Freeze this workspace to prevent any changes until workspace is unfrozen.
-- However, first go to the LIVE workspace, because a workspace cannot be frozen
-- if any users (including you) are in it.
EXECUTE DBMS_WM.GotoWorkspace ('LIVE');
EXECUTE DBMS_WM.FreezeWorkspace ('B_focus_1');

-----
-- CREATE ANOTHER SCENARIO IN SECOND WORKSPACE --
-----

-- Enter the B_focus_2 workspace and change the cola_b manager to Burton and
-- raise the cola_b budget amount by 0.5 to bring it to 2.0. Reduce only the
-- cola_d amount by 0.5 to stay within the overall budget.

EXECUTE DBMS_WM.GotoWorkspace ('B_focus_2');
UPDATE cola_marketing_budget
   SET manager = 'Burton' WHERE product_name = 'cola_b';
UPDATE cola_marketing_budget
   SET budget = 2 WHERE product_name = 'cola_b';
UPDATE cola_marketing_budget

```

```
    SET budget = 3 WHERE product_name = 'cola_d';
COMMIT;

-- Relevant data values now in B_focus_2 workspace::
-- 1, cola_a, Alvarez, 2.0 (no change from LIVE)
-- 2, cola_b, Burton, 2.0
-- 3, cola_c, Chen, 1.5 (no change from LIVE)
-- 4, cola_d, Davis, 3.0 (same manager, new budget)

-- Create a savepoint (B_focus_2_SP1), then change scenario to
-- raise cola_b budget and reduce cola_d budget by 0.5 each.

EXECUTE DBMS_WM.CreateSavepoint ('B_focus_2', 'B_focus_2_SP1');
UPDATE cola_marketing_budget
    SET budget = 2.5 WHERE product_name = 'cola_b';
UPDATE cola_marketing_budget
    SET budget = 2.5 WHERE product_name = 'cola_d';
COMMIT;

-- Relevant data values now in B_focus_2 workspace:
-- 1, cola_a, Alvarez, 2.0 (no change from LIVE)
-- 2, cola_b, Burton, 2.5
-- 3, cola_c, Chen, 1.5 (no change from LIVE)
-- 4, cola_d, Davis, 2.5 (same manager, new budget)

-- Discard this scenario; roll back to row values at the time savepoint
-- B_focus_2_SP1 was created. First, though, get out of the workspace
-- so it can be rolled back (no users in it).

EXECUTE DBMS_WM.GotoWorkspace ('LIVE');
EXECUTE DBMS_WM.RollbackToSP ('B_focus_2', 'B_focus_2_SP1');

-- Go back to the B_focus_2 workspace and display current values
-- (should include budget of 2 for cola_b and 3 for cola_d).
EXECUTE DBMS_WM.GotoWorkspace ('B_focus_2');
SELECT * FROM cola_marketing_budget;

-----
-- SELECT SCENARIO AND UPDATE DATABASE --
-----

-- Assume that you have decided to adopt the scenario of the second
-- workspace (B_focus_2) using that workspace's current values.

-- First go to the LIVE workspace, because a workspace cannot be removed
-- or merged if any users (including you) are in it.
EXECUTE DBMS_WM.GotoWorkspace ('LIVE');

-- Unfreeze the first workspace and remove it to discard any changes there.
EXECUTE DBMS_WM.UnfreezeWorkspace ('B_focus_1');
EXECUTE DBMS_WM.RemoveWorkspace ('B_focus_1');

-- Apply changes in the second workspace to the LIVE database workspace.
-- Note that the workspace is not removed by default after MergeWorkspace.
EXECUTE DBMS_WM.MergeWorkspace ('B_focus_2');

-- Display the current data values as seen by the LIVE workspace.
SELECT * FROM cola_marketing_budget;

-----
-- DISABLE VERSIONING --
-----
```

```
-- Disable versioning on the table because you are finished testing scenarios.
-- Set force parameter to TRUE if you want to force the disabling even
-- if changes were made in a non-LIVE workspace.

EXECUTE DBMS_WM.DisableVersioning ('cola_marketing_budget', TRUE);
```

## 1.17.2 Example: Warehouse Expansion Options

In [Example 1-5](#), a company that uses the Oracle sample schemas decided that it needs additional warehouse space. It wants to consider two scenarios: a single large warehouse in Town A, and two smaller warehouses in Town B and Town C that together offer more total storage capacity. There are potential advantages and disadvantages to each scenario, and financial and legal issues to be resolved with each. Later, the company decides that it might need even more warehouse space under each scenario, so it wants to consider the same additional warehouse in each scenario.

[Example 1-5](#) creates a workspace for each scenario; and within each workspace it creates a savepoint before adding an extra new warehouse to the table, because the company might decide not to use the extra warehouse. The warehouse rows are stored on the `OE.WAREHOUSES` table, which is part of the Oracle sample schemas.

### Example 1-5 Warehouse Expansion Options

```
-----
-- INITIAL SET-UP
-----

-- Clean up from any previous running of this procedure.
DROP USER wm_developer CASCADE;

-- Create the user for schema objects.
CREATE USER wm_developer IDENTIFIED BY password;

-- Grant regular privileges.
GRANT create session,
      unlimited tablespace,
      create table
TO wm_developer;

-- Grant privileges on tables to be modified.
GRANT select, insert, delete, update ON oe.warehouses TO wm_developer;
GRANT select, insert, delete, update ON hr.locations TO wm_developer;

-- Grant WM-specific privileges (with grant_option = YES).
EXECUTE DBMS_WM.GrantSystemPriv ('ACCESS_ANY_WORKSPACE, MERGE_ANY_WORKSPACE,
      CREATE_ANY_WORKSPACE, REMOVE_ANY_WORKSPACE, ROLLBACK_ANY_WORKSPACE',
      'wm_developer', 'YES');

-- WM_ADMIN_ROLE grants the ability to version-enable a table in another schema.
GRANT wm_admin_role TO wm_developer;

-- Create rows for new locations, since a valid location ID is needed for each
-- proposed new warehouse.
INSERT INTO hr.locations VALUES
      (4000, '123 Any Street', '01234', 'Town A', 'MA', 'US');
INSERT INTO hr.locations VALUES
      (4100, '456 Some Street', '01235', 'Town B', 'MA', 'US');
INSERT INTO hr.locations VALUES
      (4200, '789 Other Street', '01236', 'Town C', 'MA', 'US');
INSERT INTO hr.locations VALUES
```

```
(4300, '1 Yetanother Street', '01237', 'Town D', 'MA', 'US');

-----
-- CREATE AND VERSION-ENABLE THE DATA TABLE --
-----

CONNECT wm_developer
-- Enter password when prompted.
set echo on
set serveroutput on

-- Version-enable the OE.WAREHOUSES table. Specify hist option of
-- VIEW_WO_OVERWRITE so that the WAREHOUSES_HIST view contains
-- complete history information about data changes. However, because
-- OE.WAREHOUSES is the parent table in a referential integrity constraint
-- with OE.INVENTORIES, you must also version-enable that table.

EXECUTE DBMS_WM.EnableVersioning ('OE.WAREHOUSES, OE.INVENTORIES', hist =>
'VIEW_WO_OVERWRITE');

-----
-- CREATE AND USE WORKSPACES --
-----

-- The company has decided that it needs additional warehouse space.
-- It wants to consider two scenarios: a single large warehouse in Town A,
-- and two smaller warehouses in Town B and Town C that together offer more
-- total storage capacity. There are potential advantages and disadvantages
-- to each scenario, and financial and legal issues to be resolved with each.
--
-- Later, the company decides that it might need even more warehouse
-- space under each scenario, so it wants to consider the same additional
-- warehouse in each scenario.

-- Create a workspace for each scenario, with both created as child
-- workspaces of the LIVE database workspace.
-- In workspace large_warehouse, add one row for the single large warehouse.
-- In workspace smaller_warehouses, add two rows, one for each warehouse.
--
-- Also, within each workspace create a savepoint before adding the
-- extra warehouse, because the company might decide it does not
-- need the warehouse.

EXECUTE DBMS_WM.CreateWorkspace (workspace => 'large_warehouse');
EXECUTE DBMS_WM.CreateWorkspace (workspace => 'smaller_warehouses');

-- Set up the first scenario: Go to the large_warehouse workspace and first add
-- one row for a warehouse.

EXECUTE DBMS_WM.GotoWorkspace (workspace => 'large_warehouse');

INSERT INTO oe.warehouses VALUES (10, NULL, 'Town A', 4000,
SDO_GEOMETRY(2001, 8307,
SDO_POINT_TYPE(-71.00703, 42.27099, NULL), NULL, NULL));

UPDATE oe.warehouses SET warehouse_spec = sys.xmltype.createxml(
'<?xml version="1.0"?>
<Warehouse>
<Building>Owned</Building>
<Area>100000</Area>
<Docks>2</Docks>
<DockType>Side load</DockType>
<WaterAccess>Y</WaterAccess>
```

```
<RailAccess>Y</RailAccess>
<Parking>Lot</Parking>
<VClearance>15 ft</VClearance>
</Warehouse>'
) WHERE warehouse_id = 10;

COMMIT;

-- Create a savepoint so that you can, if necessary, roll back to the point
-- before the extra warehouse was added.
EXECUTE DBMS_WM.CreateSavepoint ('large_warehouse', 'large_warehouse_add_wh');

-- Add another warehouse for this scenario.
INSERT INTO oe.warehouses VALUES (11, NULL, 'Town D', 4300,
    SDO_GEOMETRY(2001, 8307,
    SDO_POINT_TYPE(-71.00707, 42.35226, NULL), NULL, NULL));

UPDATE oe.warehouses SET warehouse_spec = sys.xmltype.createxml(
'<?xml version="1.0"?>
<Warehouse>
<Building>Leased</Building>
<Area>55000</Area>
<Docks>1</Docks>
<DockType>Rear load</DockType>
<WaterAccess>N</WaterAccess>
<RailAccess>N</RailAccess>
<Parking>Street</Parking>
<VClearance>10 ft</VClearance>
</Warehouse>'
) WHERE warehouse_id = 11;

COMMIT;

-- Freeze this workspace to prevent any changes until the workspace is unfrozen.
-- However, first go to the LIVE workspace, because a workspace cannot be frozen
-- if any users (including you) are in it.
EXECUTE DBMS_WM.GotoWorkspace ('LIVE');
EXECUTE DBMS_WM.FreezeWorkspace ('large_warehouse');

-- Set up the second scenario: Go to the smaller_warehouses workspace and first
-- add two rows for the smaller warehouses.

EXECUTE DBMS_WM.GotoWorkspace ('smaller_warehouses');

INSERT INTO oe.warehouses VALUES (10, NULL, 'Town B', 4100,
    SDO_GEOMETRY(2001, 8307,
    SDO_POINT_TYPE(-71.02439, 42.28628, NULL), NULL, NULL));

INSERT INTO oe.warehouses VALUES (11, NULL, 'Town C', 4200,
    SDO_GEOMETRY(2001, 8307,
    SDO_POINT_TYPE(-70.97980, 42.37961, NULL), NULL, NULL));

UPDATE oe.warehouses SET warehouse_spec = sys.xmltype.createxml(
'<?xml version="1.0"?>
<Warehouse>
<Building>Owned</Building>
<Area>60000</Area>
<Docks>1</Docks>
<DockType>Side load</DockType>
<WaterAccess>Y</WaterAccess>
<RailAccess>Y</RailAccess>
```



```

<Parking>Lot</Parking>
<VClearance>15 ft</VClearance>
</Warehouse>'
) WHERE warehouse_id = 10;

UPDATE oe.warehouses SET warehouse_spec = sys.xmltype.createxml(
'<?xml version="1.0"?>
<Warehouse>
<Building>Leased</Building>
<Area>550000</Area>
<Docks>1</Docks>
<DockType>Rear load</DockType>
<WaterAccess>N</WaterAccess>
<RailAccess>Y</RailAccess>
<Parking>Street</Parking>
<VClearance>12 ft</VClearance>
</Warehouse>'
) WHERE warehouse_id = 11;

COMMIT;

-- Create a savepoint so that you can, if necessary, roll back to the point
-- before the extra warehouse was added.
EXECUTE DBMS_WM.CreateSavepoint ('smaller_warehouses', 'smaller_warehouses_add_wh');

-- Add the extra warehouse for this scenario.
INSERT INTO oe.warehouses VALUES (12, NULL, 'Town D', 4300,
SDO_GEOMETRY(2001, 8307,
SDO_POINT_TYPE(-71.00707, 42.35226, NULL), NULL, NULL));

UPDATE oe.warehouses SET warehouse_spec = sys.xmltype.createxml(
'<?xml version="1.0"?>
<Warehouse>
<Building>Leased</Building>
<Area>55000</Area>
<Docks>1</Docks>
<DockType>Rear load</DockType>
<WaterAccess>N</WaterAccess>
<RailAccess>N</RailAccess>
<Parking>Street</Parking>
<VClearance>10 ft</VClearance>
</Warehouse>'
) WHERE warehouse_id = 12;

COMMIT;

-----
-- SELECT A SCENARIO, AND APPLY IT --
-----

-- Later, the company makes its decisions:
-- 1. Add two smaller warehouses.
-- 2. Do not add the extra warehouse (that is, no third new warehouse).
-- Consequently, you need to discard the first scenario (large_warehouse
-- workspace) completely, discard the warehouse addition in the second
-- scenario (roll back to smaller_warehouses_add_wh savepoint), and
-- apply the second scenario.

-- First go to the LIVE workspace, because a workspace cannot be removed
-- or merged if any users (including you) are in it.
EXECUTE DBMS_WM.GotoWorkspace ('LIVE');

```

```

-- Unfreeze the first workspace and remove it to discard any changes there.
EXECUTE DBMS_WM.UnfreezeWorkspace ('large_warehouse');
EXECUTE DBMS_WM.RemoveWorkspace ('large_warehouse');

-- Rollback the workspace for the second scenario to the savepoint created
-- before the extra warehouse was added.
EXECUTE DBMS_WM.RollbackToSP ('smaller_warehouses', 'smaller_warehouses_add_wh');

-- Apply changes in the smaller_warehouses workspace to the LIVE database
-- workspace; use the remove_workspace parameter to remove the
-- smaller_warehouses workspace after the merge.
EXECUTE DBMS_WM.MergeWorkspace ('smaller_warehouses', remove_workspace => TRUE);

-- The OE.WAREHOUSES table now has the desired data (two additional warehouses
-- from the smaller_warehouses scenario). Display the IDs and names just to be
-- sure.
SELECT warehouse_id, warehouse_name FROM oe.warehouses
       ORDER BY warehouse_id;

-- Disable versioning on the table because you are finished testing scenarios.
-- Set the force parameter to TRUE to force disabling even though changes
-- were made in a non-LIVE workspace. You must also version-disable
-- the other tables previously version-enabled (along with OE.WAREHOUSES).

EXECUTE DBMS_WM.DisableVersioning ('OE.WAREHOUSES, OE.INVENTORIES', force =>
TRUE);

-- Clean up by deleting the rows that were added to the OE.WAREHOUSES table.
DELETE FROM oe.warehouses WHERE warehouse_id >= 10;

-- Clean up by deleting the locations that were added.
DELETE FROM hr.locations WHERE location_id >= 4000;

```

The **SELECT** statement near the end of [Example 1-5](#) displays the IDs and names of warehouses in the **OE.WAREHOUSES** table, including the newly added warehouses in Town B and Town C, as shown in the following example:

```

SELECT warehouse_id, warehouse_name FROM oe.warehouses
       ORDER BY warehouse_id;

```

```

WAREHOUSE_ID WAREHOUSE_NAME
-----
1 Southlake, Texas
2 San Francisco
3 New Jersey
4 Seattle, Washington
5 Toronto
6 Sydney
7 Mexico City
8 Beijing
9 Bombay
10 Town B
11 Town C

```

# 2

## Workspace Manager Events

Certain applications may be interested in knowing what Workspace Manager operations are being performed and may want to take some actions based on that. Several types of Workspace Manager operations can be captured as events.

Workspace Manager provides a framework for communicating these events asynchronously to the interested applications. The applications can then take some actions based on the event. Some scenarios in which events can be used include the following:

- An application wants to be notified whenever a workspace is merged to LIVE so that it can refresh its data.
- Workspace data needs to be archived whenever a new savepoint is created.

The Workspace Manager event framework is built on the Oracle Advanced Queuing (AQ) capability. Messaging features provided by AQ, such as asynchronous notification, persistence, propagation, access control, history, and rule-based subscription, can be used for Workspace Manager events.

Workspace Manager creates a multiconsumer queue where events are enqueued. The relevant information about the event, such as the type of event, the user and workspace that triggered the event, and the name of the versioned table, is initialized in the event payload and enqueued. Applications can subscribe to these events, optionally specifying a rule for their subscriptions. Only the events that satisfy the rule will be applicable to the subscriber. Subscribers can get event notification in variety of ways, such as listening for the events in the queue, registering a callback for notification, or explicitly dequeuing events from the queue.

Because events are communicated asynchronously to the other applications, the performance of the workspace operation generating the event is not affected.

### Note:

To use Workspace Manager events in an application, you must understand the relevant AQ concepts and techniques described in *Oracle Database Advanced Queuing User's Guide*.

This chapter contains the following major sections:

- [List of Workspace Manager Events](#)  
The following table lists the Workspace Manager events and when each occurs.
- [Event Parameters](#)  
When an event occurs, information is stored in parameters that are bundled into an object type called `WMSYS.WM$EVENT_TYPE` and enqueued into the event queue. A subscriber can dequeue the event object on receiving notification.

- [ALLOW\\_CAPTURE\\_EVENTS System Parameter](#)  
The Workspace Manager system parameter `ALLOW_CAPTURE_EVENTS` can be used to capture events.
- [AQ Operations and Workspace Manager Events](#)  
This section describes Advanced Queuing objects and techniques relevant to developers of applications that work with captured Workspace Manager events.

## 2.1 List of Workspace Manager Events

The following table lists the Workspace Manager events and when each occurs.

**Table 2-1 Workspace Manager Events**

Event	Occurs
TABLE_MERGE_W_REMOVE_DATA	When <a href="#">MergeTable</a> is invoked with <code>remove_data</code> set to <code>TRUE</code> .
TABLE_MERGE_WO_REMOVE_DATA	When <a href="#">MergeTable</a> is invoked with <code>remove_data</code> set to <code>FALSE</code> .
TABLE_REFRESH	When <a href="#">RefreshTable</a> is invoked.
TABLE_ROLLBACK	When <a href="#">RollbackTable</a> is invoked.
WORKSPACE_COMPRESS	When <a href="#">CompressWorkspace</a> or <a href="#">CompressWorkspaceTree</a> is invoked.
WORKSPACE_CREATE	When <a href="#">CreateWorkspace</a> is invoked.
WORKSPACE_MERGE_W_REMOVE	When <a href="#">MergeWorkspace</a> is invoked with <code>remove_workspace</code> set to <code>TRUE</code> .
WORKSPACE_MERGE_WO_REMOVE	When <a href="#">MergeWorkspace</a> is invoked with <code>remove_workspace</code> set to <code>FALSE</code> .
WORKSPACE_REFRESH	When <a href="#">RefreshWorkspace</a> is invoked.
WORKSPACE_REMOVE	When <a href="#">RemoveWorkspace</a> or <a href="#">RemoveWorkspaceTree</a> is invoked.
WORKSPACE_ROLLBACK	When <a href="#">RollbackWorkspace</a> is invoked.
WORKSPACE_VERSION	When a new version is created in the workspace as a result of the creation of an explicit or implicit savepoint. (Savepoints are described in <a href="#">Using Savepoints</a> .)

## 2.2 Event Parameters

When an event occurs, information is stored in parameters that are bundled into an object type called `WMSYS.WM$EVENT_TYPE` and enqueued into the event queue. A subscriber can dequeue the event object on receiving notification.

The following table describes the Workspace Manager event parameters.

**Table 2-2 Workspace Manager Event Parameters**

Event Parameter	Data Type	Description
<code>event_name</code>	<code>VARCHAR2(128)</code>	Name indicating the type of event.

**Table 2-2 (Cont.) Workspace Manager Event Parameters**

Event Parameter	Data Type	Description
workspace_name	VARCHAR2 (128)	Workspace that caused the event to occur.
parent_workspace_name	VARCHAR2 (128)	Parent workspace of the workspace that caused the event to occur.
user_name	VARCHAR2 (128)	User that caused the event to occur.
table_name	VARCHAR2 (128)	Version-enabled table on which the event occurred. If this parameter does not apply to an event, it is null.
aux_params	WMSYS.WM\$NV_PAIR_NT_TYPE (which is table of WMSYS.WM\$NV_PAIR_TYP E)	<p>A nested table of (name,value) pairs that can contain additional information about the event.</p> <p>For TABLE_xxx events, it has one row containing the WHERE clause string used for the operation.</p> <p>For WORKSPACE_VERSION events, it has one row containing the savepoint name associated with the newly created version.</p>

## 2.3 ALLOW\_CAPTURE\_EVENTS System Parameter

The Workspace Manager system parameter `ALLOW_CAPTURE_EVENTS` can be used to capture events.

Before you can capture any Workspace Manager events, you must use the [SetSystemParameter](#) procedure to set the Workspace Manager system parameter `ALLOW_CAPTURE_EVENTS` to the value `ON`. This does not, however, cause any events to be captured; to capture events, you must use the [SetCaptureEvent](#) procedure.

You can later disallow the capture of Workspace Manager events by using the [SetSystemParameter](#) procedure to set `ALLOW_CAPTURE_EVENTS` to the value `OFF`, but you must first ensure that no events are currently being captured. [Example 2-1](#) shows the sequence of procedure calls for enabling and disabling the capture of all events, and starting and stopping the capture all events.

### Example 2-1 Capturing Workspace Manager Events

```
-- Allow Workspace Manager events to be captured. (Required for SetCaptureEvent)
EXECUTE DBMS_WM.SetSystemParameter ('ALLOW_CAPTURE_EVENTS', 'ON');
-- Start capturing all Workspace Manager events.
EXECUTE DBMS_WM.SetCaptureEvent ('ALL_EVENTS','ON');
.
.
.
-- Stop capturing all Workspace Manager events.
EXECUTE DBMS_WM.SetCaptureEvent ('ALL_EVENTS','OFF');
-- Disallow capture of Workspace Manager events.
EXECUTE DBMS_WM.SetSystemParameter ('ALLOW_CAPTURE_EVENTS', 'OFF');
```

## 2.4 AQ Operations and Workspace Manager Events

This section describes Advanced Queuing objects and techniques relevant to developers of applications that work with captured Workspace Manager events.

- [Workspace Manager Event Queue Administration](#)
- [Privileges and Access Control for Queues](#)
- [Rule-Based Subscription](#)
- [Listening for Events](#)
- [Asynchronous Notification](#)

### 2.4.1 Workspace Manager Event Queue Administration

Workspace Manager creates a multiconsumer queue named `WMSYS.WM$EVENT_QUEUE` based on a queue table named `WMSYS.WM$EVENT_QUEUE_TABLE`. The queue payload type is `WMSYS.WM$EVENT_TYPE`, which is an object type.

AQ creates some views for the queue that can be used for administrative purposes. [Table 2-3](#) describes the views of interest to developers of Workspace Manager applications.

**Table 2-3 AQ Administrative Views for Workspace Manager**

View Name	Description
<code>WMSYS.AQ\$WM\$EVENT_QUEUE_TABLE</code>	Describes the queue table in which events are stored. This view can be used for querying the events. The <code>AQ_ADMINISTRATOR_ROLE</code> role and the <code>WM_ADMIN</code> system privilege are granted <code>select</code> privileges on this view.
<code>WMSYS.AQ\$WM\$EVENT_QUEUE_TABLE_S</code>	Displays all the subscribers for the event queue; also displays the transformation for the subscriber if it was created with one. The <code>AQ_ADMINISTRATOR_ROLE</code> role and the <code>WM_ADMIN</code> system privilege are granted <code>select</code> privileges on this view.
<code>WMSYS.AQ\$WM\$EVENT_QUEUE_TABLE_R</code>	Displays only the rule-based subscribers for all queues in a given queue table, as well as the text of the rule defined by each subscriber. Also displays the transformation for the subscriber if one was specified. The <code>AQ_ADMINISTRATOR_ROLE</code> role and the <code>WM_ADMIN</code> system privilege are granted <code>select</code> privileges on this view.

### 2.4.2 Privileges and Access Control for Queues

The database administrator has several options for granting privileges and access to queues. Some possible scenarios include:

- Grant the system privileges `ENQUEUE ANY QUEUE` and `DEQUEUE ANY QUEUE` directly to a database user by using the `DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE` procedure, and optionally later revoke privileges by using the `DBMS_AQADM.REVOKE_SYSTEM_PRIVILEGE` procedure.

- Grant the queue privileges `ENQUEUE` and `DEQUEUE` to the event queue `WMSYS.WM$EVENT_QUEUE` to a database user by using the `DBMS_AQADM.GRANT_QUEUE_PRIVILEGE` procedure, and optionally later revoke privileges by using the `DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE` procedure.
- Grant the role `AQ_ADMINISTRATOR_ROLE` to a database user to give that user administrative privileges on any queue.

[Example 2-2](#) shows privileges being granted for a user to subscribe to the event queue and dequeue events.

### Example 2-2 Granting Privileges for Queue Access

```
-- Do the following while connected as SYSDBA.
-- These privileges are required for the user to execute AQ packages.
grant execute on DBMS_AQ to SCOTT ;
grant execute on DBMS_AQADM to SCOTT ;

-- Grant privilege to SCOTT for subscribing to the event queue.
exec DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE('MANAGE_ANY','SCOTT') ;

-- Grant privilege to SCOTT to dequeue events. (As an alternative, you could use
-- DBMS_AQADM.GRANT_QUEUE_PRIVILEGE to grant the DEQUEUE privilege on
-- WMSYS.WM$EVENT_QUEUE.)
exec DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE('DEQUEUE_ANY','SCOTT') ;
```

## 2.4.3 Rule-Based Subscription

An event can be delivered to multiple recipients based on event parameters. You can define a rule-based subscription for the event queue as the mechanism for specifying interest in receiving events. Subscriber rules are then used to evaluate recipients for event delivery. A null rule indicates that the subscriber wishes to receive all events.

[Example 2-3](#) creates a rule-based subscription for user `SCOTT` to deliver `WORKSPACE_MERGE_WO_REMOVE` events when the parent workspace is the `LIVE` workspace.

### Example 2-3 Rule-Based Subscription for Workspace Manager Events

```
rem =====
rem Create queue subscribers
rem Register for MergeWorkspace event when
rem a workspace is merged to LIVE
rem =====

connect scott
-- Enter password when prompted.

DECLARE
    subscriber sys.aq$_agent;
BEGIN
    subscriber := sys.aq$_agent('MERGE_LISTENER', NULL, NULL);
    dbms_aqadm.add_subscriber(
        queue_name => 'WMSYS.WM$EVENT_QUEUE',
        subscriber => subscriber,
        rule => 'tab.user_data.event_name = ''WORKSPACE_MERGE_WO_REMOVE''
              and tab.user_data.parent_workspace_name = ''LIVE''');
END;
/
```

## 2.4.4 Listening for Events

The listen call is a blocking call that can be used to wait for events on a queue or a list of subscriptions. If the listen returns successfully, a dequeue must be used to retrieve the event.

[Example 2-4](#) listens for events on an event queue.

### Example 2-4 Listening for a Workspace Manager Event

```
rem =====
rem The following example shows how an application can listen for
rem an event. Explicit dequeue must be performed to get the actual
rem event parameters. The user SCOTT must have sufficient privileges
rem as described in the "Access Control" section.
rem =====

connect scott
-- Enter password when prompted.

set serveroutput on

DECLARE
  qlist dbms_aq.aq$agent_list_t;
  agent_w_msg sys.aq$agent;
  listen_timeout exception;
  pragma exception_init(listen_timeout, -25254);
BEGIN
  qlist(0) := sys.aq$agent('MERGE_LISTENER', 'WMSYS.WM$EVENT_QUEUE', NULL);

  dbms_output.put_line ('Listening on event queue.');
```

```
  BEGIN

    DBMS_AQ.LISTEN(
      agent_list => qlist,
      wait => 30,
      agent => agent_w_msg);

    dbms_output.put_line(agent_w_msg.name) ;

    /* The event can be dequeued here to get the event data */

  EXCEPTION
    when listen_timeout THEN
      null;
  END;
```

```
END;
/
```

## 2.4.5 Asynchronous Notification

Asynchronous notification allows clients to receive notification of an event of interest. The client can use it to monitor multiple subscriptions. The client does not have to be connected to the database to receive notifications regarding its subscriptions.



If an application registers for asynchronous notification of Workspace Manager events using callbacks, the minimum values for the following `init.ora` parameters should be:

- `aq_tm_processes = 1`
- `job_queue_processes = 2`

[Example 2-5](#) registers for a callback to receive asynchronous notification of events.

### Example 2-5 Receiving Asynchronous Notification of Events

```
rem =====
rem Example of how to register for a callback to the event
rem queue on behalf of a subscriber. Subscriber has already
rem been defined in previous section. The callback is
rem invoked by the AQ framework whenever an event satisfying the
rem rule for the subscriber occurs. The minimum values for
rem the following init.ora parameters should be set as follows.
rem  aq_tm_processes = 1
rem  job_queue_processes = 2
rem The user SCOTT must have sufficient privileges.
rem =====

CONNECT scott
-- Enter password when prompted.

CREATE TABLE merge_log
(
  event_name      varchar2(128),
  workspace_name  varchar2(128),
  parent_workspace_name varchar2(128),
  user_name       varchar2(128)
);

CREATE OR REPLACE PROCEDURE scott.event_callback(
  context RAW , reginfo sys.aq$_reg_info, descr sys.aq$_descriptor,
  payload VARCHAR2, payload1 NUMBER)
AS
  deq_msgid      RAW(16);
  dopt           dbms_aq.dequeue_options_t;
  mprop         dbms_aq.message_properties_t;
  event         WMSYS.WM$EVENT_TYPE;
  no_messages   exception;
  pragma exception_init(no_messages, -25228);

BEGIN
  dopt.consumer_name := 'MERGE_LISTENER';
  dopt.wait := 30;
  dopt.msgid := descr.msg_id;

  dbms_aq.dequeue(
    queue_name => 'WMSYS.WM$EVENT_QUEUE',
    dequeue_options => dopt,
    message_properties => mprop,
    payload => event,
    msgid => deq_msgid);

  INSERT INTO merge_log VALUES (event.event_name, event.workspace_name,
    event.parent_workspace_name, event.user_name);

  /* Note: If there are additional parameters stored in
  "aux_params" attribute, it can be accessed using
```

```
        event.aux_params(1).name, event.aux_params(1).value,
        event.aux_params(2).name ... and so on. The number of
        parameters can be accessed using event.aux_params.count
        when aux_params is not null.
    */
END;
/

grant execute on scott.event_callback to public ;

rem =====
rem Register a callback for the event
rem Queue name and subscriber name have to be specified
rem while registering for a callback
rem =====

DECLARE
    reginfo1          sys.aq$_reg_info;
    reginfo1list      sys.aq$_reg_info_list;
BEGIN
    reginfo1 := sys.aq$_reg_info('WMSYS.WM$EVENT_QUEUE:MERGE_LISTENER',1,'plsql://
scott.event_callback?PR=1',HEXTORAW('FF'));

    reginfo1list := sys.aq$_reg_info_list(reginfo1);

    sys.dbms_aq.register(reginfo1list, 1);

    COMMIT;

END;
/
```

# 3

## Workspace Manager Valid Time Support

This chapter describes the support for valid time, also known as effective dating, with version-enabled tables.

It contains the following major sections:

- [Valid Time Support: Introduction and Example](#)  
Some applications need to store data with an associated time range that indicates the validity of the data. That is, each record is valid only within the time range associated with the record.
- [WM\\_PERIOD Data Type](#)  
The `WM_PERIOD` data type is used to specify a valid time range for the session or workspace, and for a row in a version-enabled table.
- [Valid Time Constants](#)  
The following table lists constants that can be used in the `validFrom` and `validTill` timestamps of a `WM_PERIOD` specification.
- [API Features for Valid Time Support](#)  
The following table lists `DBMS_WM` subprograms that are devoted to valid time support or that have parameters related to valid time support.
- [Operators for Valid Time Support](#)  
Workspace Manager provides relationship checking operators and set operators that accept two time period parameters and that can be used to apply valid time filters in a query.
- [Queries and DML Operations with Valid Time Support](#)  
This section describes some behaviors and considerations for queries and data manipulation language (insert, update, and delete) operations related to valid time support.
- [Constraint Management for Valid Time Support](#)  
This section describes considerations related to valid time support that affect referential integrity constraints and unique constraints.
- [Locking with Valid Time Support](#)  
If a row in a version-enabled table with valid time support is locked, it is automatically locked for its entire valid time period. There is no way to lock a row for a specified time period.
- [Static Data Dictionary Views Affected by Valid Time Support](#)  
This section describes the effect on valid time support on Workspace Manager static data dictionary views.
- [Adding Valid Time Support to an Existing Table](#)  
You can add valid time support to an existing version-enabled table.
- [Merging and Refreshing Workspaces for Tables with Valid Time Support](#)  
When tables that have valid time support are merged or refreshed, you can determine the resulting rows by considering the distinct cases when there is an intersection between rows for the same primary key value.

## 3.1 Valid Time Support: Introduction and Example

Some applications need to store data with an associated time range that indicates the validity of the data. That is, each record is valid only within the time range associated with the record.

You can enable valid time support when you version-enable a table. (You can also add valid time support to an existing version-enabled table, as explained in [Adding Valid Time Support to an Existing Table](#).) If you enable valid time support, each row contains an added column to hold the valid time period associated with the row. You can specify a valid time range for the session, and Workspace Manager will ensure that queries and insert, update, and delete operations correctly reflect and accommodate the valid time range. The valid time range specified can be in the past or the future, or it can include the past, present, and future.

[Example 3-1](#) presents a simple example of valid time support. The example does the following:

1. Creates a table of employees and their salaries.
2. Version-enables the table, specifying valid time support, which causes a column named `WM_VALID` to be added to the table automatically.
3. Inserts rows into the table. For each row, it specifies the employee name, salary, and valid time period.
4. Sets the valid time range for the session.
5. Updates a row, specifying a new salary and valid time period for an employee.
6. Disables versioning on the table.

[Example 3-1](#):

- Refers to valid time support concepts and techniques that will be explained in other sections of this chapter.
- Assumes that you are familiar with the Workspace Manager concepts and techniques explained in [Introduction to Workspace Manager](#).
- Does not create workspaces or savepoints. (These are shown in [Example: Marketing Budget Options](#) and [Example: Warehouse Expansion Options in Simplified Examples Using Workspace Manager](#).)

### Example 3-1 Valid Time Support

```
-- Create a very simple employees table (deliberately oversimplified
-- for purposes of illustration).
CREATE TABLE employees (
  name VARCHAR2(16) PRIMARY KEY,
  salary NUMBER
);

-- Version-enable the table. Specify TRUE for valid time support.
EXECUTE DBMS_WM.EnableVersioning ('employees', 'VIEW_WO_OVERWRITE', FALSE, TRUE);

INSERT INTO employees VALUES (
  'Adams',
  30000,
  WMSYS.WM_PERIOD(TO_DATE('01-01-1990', 'MM-DD-YYYY'),
    TO_DATE('01-01-2005', 'MM-DD-YYYY'))
```

```

);
INSERT INTO employees VALUES (
  'Baxter',
  40000,
  WMSYS.WM_PERIOD(TO_DATE('01-01-2000', 'MM-DD-YYYY'), DBMS_WM.UNTIL_CHANGED)
);

INSERT INTO employees VALUES (
  'Coleman',
  50000,
  WMSYS.WM_PERIOD(TO_DATE('01-01-2003', 'MM-DD-YYYY'),
    TO_DATE('12-31-9999', 'MM-DD-YYYY'))
);

COMMIT;

-- Set valid time period to virtually all time.
EXECUTE DBMS_WM.SetValidTime(TO_DATE('01-01-1900', 'MM-DD-YYYY'),
  TO_DATE('01-01-9999', 'MM-DD-YYYY'));

-- Update the salary for an existing employee. Perform "sequenced" update, so
-- that existing time-related information is preserved. This results in two rows
-- for Baxter.
-- First, set valid time to the intended range for Baxter's raise.
EXECUTE DBMS_WM.SetValidTime(TO_DATE('01-01-2003', 'MM-DD-YYYY'),
  DBMS_WM.UNTIL_CHANGED);
-- Give Baxter a raise, effective 01-Jan-2003 until changed.
UPDATE employees SET salary = 45000 WHERE name = 'Baxter';

-- Disable versioning. By default (keepWMValid parameter value of TRUE),
-- the WM_VALID column is kept, with all its data.
COMMIT;
EXECUTE DBMS_WM.DisableVersioning ('employees');

```

## 3.2 WM\_PERIOD Data Type

The `WM_PERIOD` data type is used to specify a valid time range for the session or workspace, and for a row in a version-enabled table.

The `WM_PERIOD` type is defined as follows:

```

CREATE TYPE WM_PERIOD AS OBJECT (
  validFrom  TIMESTAMP WITH TIME ZONE,
  validTill  TIMESTAMP WITH TIME ZONE,
  MAP member function wm_period_map return varchar2);

```

The `validFrom` date is inclusive, and the `validTill` period is exclusive; that is, the valid date range starts on the `validFrom` date and extends up to but not including the `validTill` date.

The `wm_period_map` member function enables ordering (sorting) and the use of `DISTINCT` on columns of type `WM_PERIOD`.

### Example 3-2 Setting the Session Valid Time to a Specific Date

[Example 3-2](#) sets the session valid time range to 01-Jan-2003.

```

EXECUTE DBMS_WM.SetValidTime(TO_DATE('01-01-2003', 'MM-DD-YYYY'),
  TO_DATE('01-02-2003', 'MM-DD-YYYY'));

```

[Example 3-3](#) inserts a row that is valid from 01-Jan-2003 until it is changed.

**Example 3-3 Inserting a Row Valid for a Time Range**

```
INSERT INTO employees VALUES (
  'Baxter',
  40000,
  WMSYS.WM_PERIOD(TO_DATE('01-01-2003', 'MM-DD-YYYY'), DBMS_WM.UNTIL_CHANGED)
);
```

If you want the valid time range to be stored in views created on tables with valid time support, using two columns of type `TIMEZONE WITH TIMESTAMP` instead of a single column of type `WM_VALID`, you can set the Workspace Manager system parameter `USE_SCALAR_TYPES_FOR_VALIDTIME` to `ON`, as explained in [System Parameters for Workspace Manager](#).

## 3.3 Valid Time Constants

The following table lists constants that can be used in the `validFrom` and `validTill` timestamps of a `WM_PERIOD` specification.

(Workspace Manager uses these as constants, but they are implemented as functions.)

**Table 3-1 Constants for Valid Time Support**

Constant	Explanation
<code>DBMS_WM.MIN_TIME</code>	The minimum (earliest) timestamp value supported by Workspace Manager. Currently the beginning of the day on 01-Jan in the year -4712 (4712 BCE).
<code>DBMS_WM.MAX_TIME</code>	The maximum (latest) timestamp value supported by Workspace Manager. Currently the end of the day (11:59.999999000 pm) on 31-Dec-9999.
<code>DBMS_WM.UNTIL_CHANGED</code>	A timestamp that is treated as <code>DBMS_WM.MAX_TIME</code> until a subsequent modification overrides the value.

## 3.4 API Features for Valid Time Support

The following table lists `DBMS_WM` subprograms that are devoted to valid time support or that have parameters related to valid time support.

**Table 3-2 API Features for Valid Time Support**

Subprogram	Valid Time Support
<a href="#">EnableVersioning</a>	If the <code>validTime</code> parameter value is <code>TRUE</code> , the table is version-enabled with valid time support. A column named <code>WM_VALID</code> of type <code>WM_PERIOD</code> is added to the table. For any existing rows, the <code>WM_VALID</code> column is set with a <code>validFrom</code> timestamp of <code>SYSTIMESTAMP</code> and a <code>validTill</code> timestamp of <code>DBMS_WM.UNTIL_CHANGED</code> .
<a href="#">DisableVersioning</a>	The <code>keepWMValid</code> parameter determines whether to keep (the default) or delete the <code>WM_VALID</code> column and its data when the table is version-disabled.

**Table 3-2 (Cont.) API Features for Valid Time Support**

Subprogram	Valid Time Support
<a href="#">GetValidFrom</a>	Returns the <code>validFrom</code> timestamp from the session valid time period.
<a href="#">GetValidTill</a>	Returns the <code>validTill</code> timestamp from the session valid time period.
<a href="#">SetValidTime</a>	Sets the session valid time period to the specified range. You can execute the procedure with no parameters (to have the valid time range set as from the current time and until changed), with only the <code>validFrom</code> parameter, or with both the <code>validFrom</code> and <code>validTill</code> parameters.
<a href="#">SetValidTimeFilterOFF</a>	Removes the valid time filter for the current session.
<a href="#">SetValidTimeFilterON</a>	Sets a valid time filter for the current session (that is, a time to be applied to version-enabled tables).
<a href="#">SetWMValidUpdateModeOFF</a>	Disables sequenced and nonsequenced update operations and sequenced delete operations on tables that have valid time support.
<a href="#">SetWMValidUpdateModeON</a>	Enables sequenced and nonsequenced update operations and sequenced delete operations on tables that have valid time support.

## 3.5 Operators for Valid Time Support

Workspace Manager provides relationship checking operators and set operators that accept two time period parameters and that can be used to apply valid time filters in a query.

The relationship checking operators return the integer value 1 if the relationship between the two periods exists, and 0 if the relationship does not exist. The following relationship checking operators for are provided for valid time support:

- [WM\\_OVERLAPS](#) checks if two periods overlap.
- [WM\\_CONTAINS](#) checks if the first period contains the second period.
- [WM\\_MEETS](#) checks if the end of the first period is the start of the second period.
- [WM\\_EQUALS](#) checks if the two periods are equal (that is, their start and end times are the same).
- [WM\\_LESSTHAN](#) checks if the end of the first period is less than (that is, earlier than) the start of the second period.
- [WM\\_GREATERTHAN](#) checks if the start of the first period is greater than (that is, later than) the end of the second period.

The set operators return the period reflecting the relationship between the two periods, or a null value if the two periods do not have the specified relationship. The following set operators for are provided for valid time support:

- [WM\\_INTERSECTION](#) returns the intersection of the two periods, that is, the time range common to both periods.
- [WM\\_LDIFF](#) returns the difference between the two periods on the left (that is, earlier in time).

- **WM\_RDIFF** returns the difference between the two periods on the right (that is, later in time).

You can use the relationship checking operators as alternatives to using the `wm_valid.validFrom` and `wm_valid.validTill` attributes of the row. For example, the following two queries, which select data valid on 01-Jan-1991, are equivalent:

```
SELECT * FROM employees e WHERE WM_CONTAINS (e.wm_valid,
      WMSYS.WM_PERIOD(TO_DATE('01-01-1991', 'MM-DD-YYYY'),
      TO_DATE('01-02-1991', 'MM-DD-YYYY')) = 1;
SELECT * from employees e
      WHERE e.wm_valid.validFrom <= TO_DATE('01-01-1991', 'MM-DD-YYYY')
      AND e.wm_valid.validTill > TO_DATE('01-03-1991', 'MM-DD-YYYY');
```

The rest of this section contains additional information about each operator. The operators are listed in alphabetical order.

- **WM\_CONTAINS**
- **WM\_EQUALS**
- **WM\_GREATERTHAN**
- **WM\_INTERSECTION**
- **WM\_LDIFF**
- **WM\_LESSTHAN**
- **WM\_MEETS**
- **WM\_OVERLAPS**
- **WM\_RDIFF**

### 3.5.1 WM\_CONTAINS

The **WM\_CONTAINS** operator checks if the first period contains the second period. **WM\_CONTAINS**(p1, p2) returns 1 only if the period p1 contains the period p2; otherwise, it returns 0.

For example:

```
WM_CONTAINS (
  WM_PERIOD(
    TO_DATE('01-01-1980', 'MM-DD-YYYY'),
    TO_DATE('01-01-1990', 'MM-DD-YYYY')),
  WM_PERIOD(
    TO_DATE('01-01-1985', 'MM-DD-YYYY'),
    TO_DATE('01-01-1988', 'MM-DD-YYYY'))) = 1

WM_CONTAINS (
  WM_PERIOD(
    TO_DATE('01-01-1980', 'MM-DD-YYYY'),
    TO_DATE('01-01-1990', 'MM-DD-YYYY')),
  WM_PERIOD(
    TO_DATE('01-01-1985', 'MM-DD-YYYY'),
    TO_DATE('01-01-1995', 'MM-DD-YYYY'))) = 0
```

**Example 3-4** returns all rows in the `EMPLOYEES` table that were valid on 01-Jan-1995 (that is, where the `WM_VALID` column value contains the period for 01-Jan-1995).



**Example 3-4 WM\_CONTAINS Operator**

```
SELECT * FROM employees e
WHERE WM_CONTAINS(e.wm_valid,
  wm_period(TO_DATE('01-01-1995', 'MM-DD-YYYY'),
    TO_DATE('01-02-1995', 'MM-DD-YYYY'))) = 1;
```

NAME	SALARY
-----	-----
WM_VALID(VALIDFROM, VALIDTILL)	
-----	-----
Adams	30000
WM_PERIOD('01-JAN-1990 12:00:00 -04:00', '01-JAN-2005 12:00:00 -04:00')	

## 3.5.2 WM\_EQUALS

The WM\_EQUALS operator checks if the first period is equal to (that is, the same as) the second period. WM\_CONTAINS(p1, p2) returns 1 only if the period p1 is equal to the period p2; otherwise, it returns 0.

For example:

```
WM_EQUALS(
  WM_PERIOD(
    TO_DATE('01-01-1980', 'MM-DD-YYYY'),
    TO_DATE('01-01-1990', 'MM-DD-YYYY')),
  WM_PERIOD(
    TO_DATE('01-01-1980', 'MM-DD-YYYY'),
    TO_DATE('01-01-1990', 'MM-DD-YYYY'))) = 1
```

```
WM_EQUALS(
  WM_PERIOD(
    TO_DATE('01-01-1980', 'MM-DD-YYYY'),
    TO_DATE('01-01-1990', 'MM-DD-YYYY')),
  WM_PERIOD(
    TO_DATE('01-01-1985', 'MM-DD-YYYY'),
    TO_DATE('01-01-1995', 'MM-DD-YYYY'))) = 0
```

[Example 3-5](#) returns all rows in the EMPLOYEES table that are valid from 01-Jan-1990 until 01-Jan-2005 (that is, where the WM\_VALID column value is equal to that period).

**Example 3-5 WM\_EQUALS Operator**

```
SELECT * FROM employees e
WHERE WM_EQUALS(e.wm_valid,
  wm_period(TO_DATE('01-01-1990', 'MM-DD-YYYY'),
    TO_DATE('01-01-2005', 'MM-DD-YYYY'))) = 1;
```

NAME	SALARY
-----	-----
WM_VALID(VALIDFROM, VALIDTILL)	
-----	-----
Adams	30000
WM_PERIOD('01-JAN-1990 12:00:00 -04:00', '01-JAN-2005 12:00:00 -04:00')	

### 3.5.3 WM\_GREATERTHAN

The WM\_GREATERTHAN operator checks if the first period is greater than (that is, occurs after) the second period. WM\_CONTAINS(p1, p2) returns 1 only if the entire period p1 is later than the period p2; otherwise, it returns 0.

For example:

```
WM_GREATERTHAN(
  WM_PERIOD(
    TO_DATE('01-01-1980', 'MM-DD-YYYY'),
    TO_DATE('01-01-1990', 'MM-DD-YYYY')),
  WM_PERIOD(
    TO_DATE('01-01-1970', 'MM-DD-YYYY'),
    TO_DATE('01-01-1980', 'MM-DD-YYYY'))) = 1
```

```
WM_GREATERTHAN(
  WM_PERIOD(
    TO_DATE('01-01-1980', 'MM-DD-YYYY'),
    TO_DATE('01-01-1990', 'MM-DD-YYYY')),
  WM_PERIOD(
    TO_DATE('01-01-1970', 'MM-DD-YYYY'),
    TO_DATE('01-01-1981', 'MM-DD-YYYY'))) = 0
```

**Example 3-6** returns all rows in the EMPLOYEES table that are valid only after 01-Jan-2001 (that is, where the WM\_VALID column timestamps are both after 01-Jan-2001).

#### Example 3-6 WM\_GREATERTHAN Operator

```
SELECT * FROM employees e
  WHERE WM_GREATERTHAN(e.wm_valid,
    wm_period(TO_DATE('01-01-2001', 'MM-DD-YYYY'),
      TO_DATE('01-02-2001', 'MM-DD-YYYY'))) = 1;
```

NAME	SALARY
-----	
WM_VALID(VALIDFROM, VALIDTILL)	
-----	
Coleman	50000
WM_PERIOD('01-JAN-2003 12:00:00 -04:00', '31-DEC-9999 12:00:00 -04:00')	

### 3.5.4 WM\_INTERSECTION

The WM\_INTERSECTION operator returns the intersection of the two periods, that is, the period common to both specified periods. WM\_INTERSECTION(p1, p2) returns a period that is the intersection of periods p1 and p2.

The following example returns the period between 01-Jan-1985 to 01-Jan-1988:

```
WM_INTERSECTION(
  WM_PERIOD(
    TO_DATE('01-01-1980', 'MM-DD-YYYY'),
    TO_DATE('01-01-1990', 'MM-DD-YYYY')),
  WM_PERIOD(
    TO_DATE('01-01-1985', 'MM-DD-YYYY'),
    TO_DATE('01-01-1988', 'MM-DD-YYYY')))
```

The following example returns the period between 01-Jan-1985 to 01-Jan-1990:

```
WM_INTERSECTION(
  WM_PERIOD(
    TO_DATE('01-01-1980', 'MM-DD-YYYY'),
    TO_DATE('01-01-1990', 'MM-DD-YYYY')),
  WM_PERIOD(
    TO_DATE('01-01-1985', 'MM-DD-YYYY'),
    TO_DATE('01-01-1995', 'MM-DD-YYYY')))
```

The following example returns a null value, because there is no intersection of the periods:

```
WM_INTERSECTION(
  WM_PERIOD(
    TO_DATE('01-01-1980', 'MM-DD-YYYY'),
    TO_DATE('01-01-1990', 'MM-DD-YYYY')),
  WM_PERIOD(
    TO_DATE('01-01-1992', 'MM-DD-YYYY'),
    TO_DATE('01-01-1995', 'MM-DD-YYYY')))
```

**Example 3-7** returns, for each row in the `EMPLOYEES` table, the employee name and the period in which the `WM_PERIOD` column value intersects the period on 01-Jan-1995.

### Example 3-7 WM\_INTERSECTION Operator

```
SELECT e.name, WM_INTERSECTION(e.wm_valid,
  wm_period(TO_DATE('01-01-1995', 'MM-DD-YYYY'),
    TO_DATE('01-02-1995', 'MM-DD-YYYY')))
FROM employees e;
```

```
NAME
-----
WM_INTERSECTION(E.WM_VALID,WM_PERIOD(TO_DATE('01-01-1995','MM-DD-
-----
Adams
WM_PERIOD('01-JAN-1995 12:00:00 -04:00', '02-JAN-1995 12:00:00 -04:00')

Baxter

Coleman
```

As you can see in the output of [Example 3-7](#), only Adams has a row that is valid on 01-Jan-1995.

## 3.5.5 WM\_LDIFF

The `WM_LDIFF` operator returns the difference between the two periods on the left (that is, earlier in time). `WM_LDIFF(p1, p2)` returns a period that is the difference between periods `p1` and `p2` on the left.

The following example returns the period between 01-Jan-1980 to 01-Jan-1985:

```
WM_LDIFF(
  WM_PERIOD(
    TO_DATE('01-01-1980', 'MM-DD-YYYY'),
    TO_DATE('01-01-1990', 'MM-DD-YYYY')),
  WM_PERIOD(
    TO_DATE('01-01-1985', 'MM-DD-YYYY'),
    TO_DATE('01-01-1988', 'MM-DD-YYYY')))
```

The following example returns a null value because `p1.validFrom` is greater than `p2.validFrom`:

```
WM_LDIFF(
  WM_PERIOD(
    TO_DATE('01-01-1980', 'MM-DD-YYYY'),
    TO_DATE('01-01-1990', 'MM-DD-YYYY')),
  WM_PERIOD(
    TO_DATE('01-01-1975', 'MM-DD-YYYY'),
    TO_DATE('01-01-1995', 'MM-DD-YYYY')))
```

The following example returns a null value because `p2` is completely outside (in this case, later than) `p1`:

```
WM_LDIFF(
  WM_PERIOD(
    TO_DATE('01-01-1980', 'MM-DD-YYYY'),
    TO_DATE('01-01-1990', 'MM-DD-YYYY')),
  WM_PERIOD(
    TO_DATE('01-01-1992', 'MM-DD-YYYY'),
    TO_DATE('01-01-1995', 'MM-DD-YYYY')))
```

**Example 3-8** returns, for each row in the `EMPLOYEES` table, the employee name and the period in which the `WM_PERIOD` column value is different on the left from 01-Jan-1995.

#### Example 3-8 WM\_LDIFF Operator

```
SELECT e.name, WM_LDIFF(e.wm_valid,
  wm_period(TO_DATE('01-01-1995', 'MM-DD-YYYY'),
    TO_DATE('01-02-1995', 'MM-DD-YYYY')))
FROM employees e;
```

```
NAME
-----
WM_LDIFF(E.WM_VALID,WM_PERIOD(TO_DATE('01-01-1995','MM-DD-YYYY'),
-----
Adams
WM_PERIOD('01-JAN-1990 12:00:00 -04:00', '01-JAN-1995 12:00:00 -04:00')

Baxter

Coleman
```

As you can see in the output of **Example 3-8**, only Adams has a row that is valid during the period of difference on the left.

## 3.5.6 WM\_LESSTHAN

The `WM_LESSTHAN` operator checks if the first period is less than (that is, occurs before) the second period. `WM_CONTAINS(p1, p2)` returns 1 only if the entire period `p1` is less than the period `p2`; otherwise, it returns 0.

For example:

```
WM_LESSTHAN(
  WM_PERIOD(
    TO_DATE('01-01-1980', 'MM-DD-YYYY'),
    TO_DATE('01-01-1990', 'MM-DD-YYYY')),
  WM_PERIOD(
```

```

    TO_DATE('01-01-1991', 'MM-DD-YYYY'),
    TO_DATE('01-01-1992', 'MM-DD-YYYY')) = 1

WM_LESSTHAN(
  WM_PERIOD(
    TO_DATE('01-01-1980', 'MM-DD-YYYY'),
    TO_DATE('01-01-1990', 'MM-DD-YYYY')),
  WM_PERIOD(
    TO_DATE('01-01-1989', 'MM-DD-YYYY'),
    TO_DATE('01-01-1992', 'MM-DD-YYYY')) = 0

```

**Example 3-9** returns all rows in the `EMPLOYEES` table that are valid only before 01-Jan-2010 (that is, where the `WM_VALID` column timestamps are both before 01-Jan-2001).

### Example 3-9 WM\_LESSTHAN Operator

```

SELECT * FROM employees e
  WHERE WM_LESSTHAN(e.wm_valid,
    wm_period(TO_DATE('01-01-2010', 'MM-DD-YYYY'),
      TO_DATE('01-02-2010', 'MM-DD-YYYY'))) = 1;

NAME                SALARY
-----
WM_VALID(VALIDFROM, VALIDTILL)
-----
Adams                30000
WM_PERIOD('01-JAN-1990 12:00:00 -04:00', '01-JAN-2005 12:00:00 -04:00')

```

## 3.5.7 WM\_MEETS

The `WM_MEETS` operator checks if the end of the first period is the start of the second period. `WM_MEETS(p1, p2)` returns 1 only if `p1.validTill = p2.validFrom`; otherwise, it returns 0.

For example:

```

WM_MEETS(
  WM_PERIOD(
    TO_DATE('01-01-1980', 'MM-DD-YYYY'),
    TO_DATE('01-01-1990', 'MM-DD-YYYY')),
  WM_PERIOD(
    TO_DATE('01-01-1990', 'MM-DD-YYYY'),
    TO_DATE('01-01-1995', 'MM-DD-YYYY'))) = 1

WM_MEETS(
  WM_PERIOD(
    TO_DATE('01-01-1980', 'MM-DD-YYYY'),
    TO_DATE('01-01-1990', 'MM-DD-YYYY')),
  WM_PERIOD(
    TO_DATE('01-01-1992', 'MM-DD-YYYY'),
    TO_DATE('01-01-1995', 'MM-DD-YYYY'))) = 0

```

**Example 3-10** returns all rows in the `EMPLOYEES` table that are valid only if the ending timestamp of the valid date period is the same as the start of the period from 01-Jan-2005 until 01-Jan-2006 (that is, if `WM_VALID.validTill` is equal to the start of the specified period).

### Example 3-10 WM\_MEETS Operator

```

SELECT * FROM employees e
  WHERE WM_MEETS(e.wm_valid,

```

```

wm_period(TO_DATE('01-01-2005', 'MM-DD-YYYY'),
          TO_DATE('01-01-2006', 'MM-DD-YYYY')) = 1;

NAME                SALARY
-----
WM_VALID(VAIDFROM, VAIDTILL)
-----
Adams                30000
WM_PERIOD('01-JAN-1990 12:00:00 -04:00', '01-JAN-2005 12:00:00 -04:00')

```

### 3.5.8 WM\_OVERLAPS

The WM\_OVERLAPS operator checks if two periods overlap. WM\_OVERLAPS(p1, p2) returns 1 if the periods p1 and p2 overlap; otherwise, it returns 0.

For example:

```

WM_OVERLAPS (
  WM_PERIOD (
    TO_DATE('01-01-1980', 'MM-DD-YYYY'),
    TO_DATE('01-01-1990', 'MM-DD-YYYY')),
  WM_PERIOD (
    TO_DATE('01-01-1985', 'MM-DD-YYYY'),
    TO_DATE('01-01-1995', 'MM-DD-YYYY')) = 1

WM_OVERLAPS (
  WM_PERIOD (
    TO_DATE('01-01-1980', 'MM-DD-YYYY'),
    TO_DATE('01-01-1990', 'MM-DD-YYYY')),
  WM_PERIOD (
    TO_DATE('01-01-1970', 'MM-DD-YYYY'),
    TO_DATE('01-01-1980', 'MM-DD-YYYY')) = 0

```

**Example 3-11** returns all rows in the EMPLOYEES table whose valid date range overlaps the period from 01-Jan-1990 until 01-Jan-2000.

#### Example 3-11 WM\_OVERLAPS Operator

```

SELECT * FROM employees e
  WHERE WM_OVERLAPS(e.wm_valid,
    wm_period(TO_DATE('01-01-1990', 'MM-DD-YYYY'),
              TO_DATE('01-01-2000', 'MM-DD-YYYY'))) = 1;

NAME                SALARY
-----
WM_VALID(VAIDFROM, VAIDTILL)
-----
Adams                30000
WM_PERIOD('01-JAN-1990 12:00:00 -04:00', '01-JAN-2005 12:00:00 -04:00')

```

### 3.5.9 WM\_RDIFF

The WM\_RDIFF operator returns the difference between the two periods on the right (that is, later in time). WM\_RDIFF(p1, p2) returns a period that is the difference between periods p1 and p2 on the right.

The following example returns the period between 01-Jan-1988 to 01-Jan-1990:

```

WM_RDIFF (
  WM_PERIOD (

```

```

    TO_DATE('01-01-1980', 'MM-DD-YYYY'),
    TO_DATE('01-01-1990', 'MM-DD-YYYY')),
WM_PERIOD(
    TO_DATE('01-01-1985', 'MM-DD-YYYY'),
    TO_DATE('01-01-1988', 'MM-DD-YYYY'))

```

The following example returns a null value because `p1.validTill` is less than `p2.validTill`:

```

WM_RDIFF(
  WM_PERIOD(
    TO_DATE('01-01-1980', 'MM-DD-YYYY'),
    TO_DATE('01-01-1990', 'MM-DD-YYYY')),
  WM_PERIOD(
    TO_DATE('01-01-1975', 'MM-DD-YYYY'),
    TO_DATE('01-01-1995', 'MM-DD-YYYY')))

```

**Example 3-12** returns, for each row in the `EMPLOYEES` table, the employee name and the period in which the `WM_PERIOD` column value is different on the right from 01-Jan-1995.

### Example 3-12 WM\_RDIFF Operator

```

SELECT e.name, WM_RDIFF(e.wm_valid,
  wm_period(TO_DATE('01-01-1995', 'MM-DD-YYYY'),
    TO_DATE('01-02-1995', 'MM-DD-YYYY')))
FROM employees e;

```

```

NAME
-----
WM_RDIFF(E.WM_VALID,WM_PERIOD(TO_DATE('01-01-1995','MM-DD-YYYY'),
-----
Adams
WM_PERIOD('02-JAN-1995 12:00:00 -04:00', '01-JAN-2005 12:00:00 -04:00')

Baxter

Coleman
WM_PERIOD('01-JAN-2003 12:00:00 -04:00', '31-DEC-9999 12:00:00 -04:00')

```

As you can see in the output of **Example 3-12**, only Adams and Coleman have rows that are valid during the period of difference on the right.

## 3.6 Queries and DML Operations with Valid Time Support

This section describes some behaviors and considerations for queries and data manipulation language (insert, update, and delete) operations related to valid time support.

- [Queries](#)
- [Data Manipulation \(DML\) Operations](#)

### 3.6.1 Queries

All queries issued against a version-enabled table with valid time support take into account the current session's valid time setting (set using the [SetValidTime](#) or [SetValidTimeFilterON](#) procedure). Unless the query specifies otherwise (for example, by using one of the valid time support operators described in [Operators for Valid Time Support](#)), each query displays all rows from the underlying table having a valid time range that overlaps the session valid time or valid time filter, and that satisfy the other conditions of the query.

By default (that is, if the [SetValidTime](#) procedure has not been invoked in the session or if it was invoked with no parameters), all rows that are valid at the current time are considered valid, and the valid time period is considered to be from the current time forward without limit.

## 3.6.2 Data Manipulation (DML) Operations

All DML statements (INSERT, UPDATE, and DELETE) issued against a version-enabled table with valid time support take into account the current session's valid time setting and update mode. (The update mode is controlled by the [SetWMValidUpdateModeON](#) and [SetWMValidUpdateModeOFF](#) procedures.) The DML statements can affect all rows that are valid for the valid time period.

By default (that is, if the [SetValidTime](#) procedure has not been invoked in the session or if it was invoked with no parameters), all rows that are valid at the current time can be affected by DML statements, and all modified rows have their valid time range timestamps set as from the current time until changed.

The following sections describe additional considerations that apply to specific kinds of DML operations.

- [Update Operations](#)
- [Insert Operations](#)

### 3.6.2.1 Update Operations

Update operations to version-enabled tables with valid time support can be sequenced or nonsequenced.

A **sequenced update** operation occurs when you do not specify a change to the `WM_VALID` column in the UPDATE statement. In a sequenced update operation, the `WM_VALID.ValidTill` value for the row is changed to the `ValidFrom` timestamp of the current session valid time range, and a new row is created in which the `WM_VALID` period reflects the current session valid time range. Sequenced updates ensure that no duplicate records are created by an UPDATE statement, because the `WM_VALID` column values are different.

[Example 3-13](#) shows a sequenced update operation, in which employee Baxter is given a raise. Before the update, there is one row for Baxter, with a salary of 40000 and a valid time period from 01-Jan-2000 until changed.

#### Example 3-13 Sequenced Update Operation

```
-- Update the salary for an existing employee. Perform "sequenced" update, so
-- that existing time-related information is preserved. This results in two rows
-- for Baxter.
-- First, set valid time to the intended range for Baxter's raise.
EXECUTE DBMS_WM.SetValidTime(TO_DATE('01-01-2003', 'MM-DD-YYYY'),
    DBMS_WM.UNTIL_CHANGED);
-- Give Baxter a raise, effective 01-Jan-2003 until changed.
UPDATE employees SET salary = 45000 WHERE name = 'Baxter';
```

The update operation in [Example 3-13](#) modifies the `WM_VALID` value of the existing row and creates a new row with the new salary value and the `WM_VALID` value reflecting the session valid time range, as shown by the following statements:



```
-- Set valid time to encompass virtually all time.
EXECUTE DBMS_WM.SetValidTime(TO_DATE('01-01-1900', 'MM-DD-YYYY'),
TO_DATE('01-02-9999', 'MM-DD-YYYY'));

-- See what data exists for Baxter.
SELECT * FROM employees WHERE name = 'Baxter';
```

NAME	SALARY
-----	
WM_VALID(VALIDFROM, VALIDTILL)	
-----	
Baxter	45000
WM_PERIOD('01-JAN-2003 12:00:00 -04:00', NULL)	
Baxter	40000
WM_PERIOD('01-JAN-2000 12:00:00 -04:00', '01-JAN-2003 12:00:00 -04:00')	

A **sequenced delete** operation deletes the portion of a row that falls within the session valid time range; that is, a new row is created in which the `WM_VALID` period reflects the current session valid time range, and then that row is deleted. If the `UPDATE` statement in [Example 3-13](#) had instead been `DELETE FROM employees WHERE name = 'Baxter';`, the new row for Baxter, valid from 01-Jan-2003 until changed, would have been deleted, but any rows for Baxter valid before 01-Jan-2003 would not be affected. There is no concept of a non-sequenced delete operation; for example, if a valid time was not set in [Example 3-13](#), a delete operation `WHERE name = 'Baxter'` would delete all rows for Baxter.

Sequenced update and delete operations are enabled when a table is version-enabled with valid time support or when valid time support is added to a version-enabled table. However, you can disable support for sequenced update and delete operations (as well as for nonsequenced update operations) by using the [SetWMValidUpdateModeOFF](#) procedure, and you can re-enable support by using the [SetWMValidUpdateModeON](#) procedure. (Both procedures are described in [DBMS\\_WM Package: Reference](#).)

A **nonsequenced update** operation occurs when you specify a change to the `WM_VALID` column in the `UPDATE` statement. In a nonsequenced update operation, no additional row is created, and the `WM_VALID` column value of the updated row or rows reflects what you specified in the `UPDATE` statement. You must ensure that a nonsequenced update operation will not result in multiple rows with the same primary key value being valid in the period specified in the `UPDATE` statement; otherwise, the update fails because of a primary key constraint violation.

If the `UPDATE` statement in [Example 3-13](#) had been a nonsequenced update operation, the result would have been only one row for Baxter: the existing row would have had the salary set to 45000 and the `WM_VALID` column set to the period specified in the `UPDATE` statement.

### 3.6.2.2 Insert Operations

When you insert a row into a version-enabled table with valid time support, you can specify a valid time period for the row. If you specify null timestamps for the period, the session valid time period is used.

When a row is inserted into a version-enabled table with valid time support, Workspace Manager checks to ensure that no existing rows with the same primary key value have a valid time range that overlaps the valid time range of the newly inserted row. If such a row is found, an exception is raised. [Example 3-14](#) shows an attempted insert operation that violates a primary key constraint because overlapping valid time periods.

**Example 3-14 Insert Operation Failing Because of Overlapping Time Periods**

```
-- Insert. Should violate primary key constraint, because of overlapping times:
-- existing Coleman row is valid from 01-Jan-2003 until 31-Dec-9999.
INSERT INTO employees VALUES (
  'Coleman',
  55000,
  WMSYS.WM_PERIOD(TO_DATE('01-01-2004', 'MM-DD-YYYY'),
                  TO_DATE('12-31-9999', 'MM-DD-YYYY'))
);
)
*
ERROR at line 6:
ORA-20010: unique key violation
ORA-06512: at "WM_DEVELOPER.OVM_INSERT_10", line 1
ORA-04088: error during execution of trigger 'WM_DEVELOPER.OVM_INSERT_10'
```

To make the statement in [Example 3-14](#) succeed, first change the `WM_VALID.ValidTill` attribute for the Coleman row to a timestamp reflecting 01-Jan-2004 or an earlier date.

## 3.7 Constraint Management for Valid Time Support

This section describes considerations related to valid time support that affect referential integrity constraints and unique constraints.

- [Referential Integrity Constraints](#)
- [Unique Constraints](#)

### 3.7.1 Referential Integrity Constraints

If a referential integrity constraint exists between two version-enabled tables that have valid time support, the valid time periods of rows are considered when the constraint is enforced. For example, assume that a `DEPARTMENTS` table has a `MANAGER_ID` column that is a foreign key referencing the `EMPLOYEE_ID` column in an `EMPLOYEES` table (that is, the department manager must be an existing employee). If both tables are version-enabled with valid time support, and if an insert or update operation would result in a new `DEPARTMENTS.MANAGER_ID` value, the operation will fail if the `DEPARTMENTS.WM_VALID` value is not within the range of the `EMPLOYEES.WM_VALID` value for the employee who is being made the department manager. (That is, the operation will fail if the new department manager is not a valid employee for the time period specified or defaulted for the insert or update operation.)

If either or both tables in a referential integrity constraint are not version-enabled with valid time support, valid time periods are ignored in enforcing the constraint.

### 3.7.2 Unique Constraints

If a unique constraint exists in a version-enabled table with valid time support, the valid time periods of rows are considered when the constraint is enforced. For example, assume that an `EMPLOYEES` table has an `EMPLOYEE_ID` column that has a unique constraint. If an insert or update operation would result in a new `EMPLOYEE_ID` value that is the same as an existing `EMPLOYEE_ID` value, the operation will fail if the `WM_VALID` values of the existing and inserted rows overlap. (That is, the operation will fail if the new employee and an existing employee have the same ID numbers and

their rows are both valid at any given time. However, the operation will succeed if the valid time periods for the two employees do not overlap.)

## 3.8 Locking with Valid Time Support

If a row in a version-enabled table with valid time support is locked, it is automatically locked for its entire valid time period. There is no way to lock a row for a specified time period.

Any updates in a pessimistically locked workspace will lock the rows seen from an ancestor workspace as the updates are performed in the workspace. The locked rows in ancestor workspaces will not be further updatable in their valid time periods as long as they are locked.

For an explanation of Workspace Manager locking, see [Lock Management with Workspace Manager](#).

## 3.9 Static Data Dictionary Views Affected by Valid Time Support

This section describes the effect on valid time support on Workspace Manager static data dictionary views.

These views are documented in [Workspace Manager Static Data Dictionary Views](#).

- [xxx\\_CONF Views and Valid Time Support](#)
- [xxx\\_DIFF Views and Valid Time Support](#)
- [xxx\\_HIST Views and Valid Time Support](#)
- [xxx\\_LOCK Views and Valid Time Support](#)
- [xxx\\_MW Views and Valid Time Support](#)

### 3.9.1 xxx\_CONF Views and Valid Time Support

For a version-enabled table with valid time support, the `xxx_CONF` view (described in [xxx\\_CONF Views](#)) will include any temporal conflicts. Such a conflict results when the valid time of a row in a parent workspace, containing the same key as a row in its child workspace, overlaps with the valid time of that row in the child workspace. Setting the session context valid time has no effect on the results of the `xxx_CONF` views, because all applicable conflicts are displayed for the entire time dimension.

For a version-enabled table with valid time support, a column named `WM_VALID`, of type `WM_PERIOD`, is added to the `xxx_CONF` view, to indicate the time period during which the row is valid. A column named `WM_CONFLICTPERIOD`, of type `WM_PERIOD`, is also added to the view, to indicate the overlapping period of the rows for which conflicts were detected.

### 3.9.2 xxx\_DIFF Views and Valid Time Support

For a version-enabled table with valid time support, the `xxx_DIFF` view (described in [xxx\\_DIFF Views](#)) will include temporal differences for a primary key between two distinct workspaces or savepoints. Such a difference occurs when a row is modified (inserted, updated, or deleted) in either a parent or child workspace. If two rows with the same primary key value are modified in both a parent and child workspace, the two rows are only correlated in the `xxx_DIFF` view when the valid time ranges of the rows overlap. Setting the session context valid time has no effect on the results of the `xxx_DIFF` views, because all applicable differences are displayed for the entire time dimension.

For a version-enabled table with valid time support, a column named `WM_VALID`, of type `WM_PERIOD`, is added to the `xxx_DIFF` view, to indicate the time period during which the row is valid. A column named `WM_DIFFPERIOD`, of type `WM_PERIOD`, is also added to the view, to indicate the overlapping period of the rows for which a difference was detected.

### 3.9.3 xxx\_HIST Views and Valid Time Support

The `xxx_HIST` views (described in [xxx\\_HIST Views](#)) include information about both valid times and transaction times. It also includes audit information, such as the name of the user that created the row. For a version-enabled table with valid time support, a column named `WM_VALID`, of type `WM_PERIOD`, is added to the `xxx_HIST` view, to indicate the time period during which the row is valid.

### 3.9.4 xxx\_LOCK Views and Valid Time Support

For a version-enabled table with valid time support, a column named `WM_VALID`, of type `WM_PERIOD`, is added to the `xxx_LOCK` view (described in [xxx\\_LOCK Views](#)), to indicate the time period during which the row is valid. The row is locked for its entire valid time period, so this is also the locking period.

### 3.9.5 xxx\_MW Views and Valid Time Support

For a version-enabled table with valid time support, a column named `WM_VALID`, of type `WM_PERIOD`, is added to the `xxx_MW` view (described in [xxx\\_MW Views](#)), to indicate the time period during which the row is valid. To see only the rows that are valid during a specific period, use the [WM\\_OVERLAPS](#) operator.

## 3.10 Adding Valid Time Support to an Existing Table

You can add valid time support to an existing version-enabled table.

To do so, use the [AlterVersionedTable](#) procedure. You can specify a valid time period to be set in the `WM_VALID` column of all existing rows, or you can accept the default period of the current timestamp until changed.

[Example 3-15](#) creates a table named `MY_TABLE`, version-enables it without valid time support, and then adds valid time support. After the valid time support is added, the `WM_VALID` column contains the default valid time period.

#### Example 3-15 Adding Valid Time Support to an Existing Version-Enabled Table

```
CREATE TABLE my_table (id NUMBER PRIMARY KEY);
EXECUTE DBMS_WM.EnableVersioning ('my_table');
INSERT INTO my_table VALUES (1);
SELECT * FROM my_table;

      ID
-----
      1

EXECUTE DBMS_WM.AlterVersionedTable('my_table', 'ADD_VALID_TIME');
SELECT * FROM my_table;

      ID
```

```

-----
WM_VALID (VALIDFROM, VALIDTILL)
-----
1
WM_PERIOD ('09-JUN-2003 10:04:13 -04:00', NULL)

```

## 3.11 Merging and Refreshing Workspaces for Tables with Valid Time Support

When tables that have valid time support are merged or refreshed, you can determine the resulting rows by considering the distinct cases when there is an intersection between rows for the same primary key value.

To determine if two row intersect, you can use the [WM\\_OVERLAPS](#) operator.

In the `validTill` cases shown in [Table 3-3](#):

- For a merge operation, the target workspace is the parent workspace.
- For a refresh operation, the target workspace is the child workspace of the specified workspace.
- Because of the intersecting nature of the rows, conflict resolution was performed before the merge or refresh operation. (A single source row might conflict with multiple target rows, and a single target row might conflict with multiple source rows.)

**Table 3-3 validTill Values and Intersection Result from Merge or Refresh Operation**

validTill Values	Intersection Result
Both the source and the target rows have a <code>validTill</code> value that is not <code>DBMS_WM.UNTIL_CHANGED</code> .	The intersecting portion of the row is modified in the target workspace to equal that of the source row.
The <code>validTill</code> value of the source row is not <code>DBMS_WM.UNTIL_CHANGED</code> , and the value for the target is <code>DBMS_WM.UNTIL_CHANGED</code> .	The intersecting portion of the row is modified and the resulting row has a <code>validTill</code> value equal to that of the source row.
The <code>validTill</code> value of the source row is <code>DBMS_WM.UNTIL_CHANGED</code> , and the value for the target row is not <code>DBMS_WM.UNTIL_CHANGED</code> .	The intersecting portion of the row is modified and the resulting row has a <code>validTill</code> value equal to that of the target row.
The <code>validTill</code> value of both the source and target rows are both <code>DBMS_WM.UNTIL_CHANGED</code> .	The intersecting portion is modified and the resulting row has a <code>validTill</code> value of <code>DBMS_WM.UNTIL_CHANGED</code> .

For all cases in [Table 3-3](#), any non-intersecting portion of the target row is left unchanged. When there is no overlapping row in the target workspace, then no special consideration needs to be made; the row is merged or refreshed as is without any additional changes.

# Part II

## Reference Information

This document has three parts:

- [Conceptual and Usage Information](#) provides conceptual and usage information about Workspace Manager.
- Part II provides reference information about the Workspace Manager PL/SQL API (DBMS\_WM package) and static data dictionary views.
- [Supplementary Information](#) provides supplementary information (appendixes and a glossary).

Part II contains the following chapters with reference information:

- [DBMS\\_WM Package: Reference](#)  
Workspace Manager includes PL/SQL subprograms (procedures and functions), in a package named `DBMS_WM`, that perform the available features of the product. This chapter provides reference information on each subprogram.
- [Workspace Manager Static Data Dictionary Views](#)  
Workspace Manager creates and maintains static data dictionary views to hold information about such things as version-enabled tables, workspaces, savepoints, users, privileges, locks, and conflicts.

# 4

## DBMS\_WM Package: Reference

Workspace Manager includes PL/SQL subprograms (procedures and functions), in a package named `DBMS_WM`, that perform the available features of the product. This chapter provides reference information on each subprogram.



### Note:

Most Workspace Manager subprograms are procedures, but a few are functions. (A function returns a value; a procedure does not return a value.)

Most functions have names starting with `Get` (such as [GetConflictWorkspace](#) and [GetWorkspace](#)).

The subprograms are presented in alphabetical order. For a brief description of subprograms according to their logical groupings, see [DBMS\\_WM Subprogram Categories](#).

Errors (exceptions) that can occur with Workspace Manager subprograms are documented in [Workspace Manager Error Messages](#), including the cause and suggested user action for each error.

Syntax notes:

- The `DBMS_WM` public synonym for the Workspace Manager PL/SQL package must be used with the subprogram name. The `DBMS_WM` public synonym is included in the format and in any examples.
- Subprogram calls are not case-sensitive, except for any quoted literal values. For example, the following code line excerpts are valid and semantically identical:

```
EXECUTE DBMS_WM.CreateWorkspace ('NEWWORKSPACE');  
EXECUTE dbms_wm.createworkspace ('NEWWORKSPACE');  
EXECUTE dBms_Wm.cReatEwoRksPace ('NEWWORKSPACE');
```



### Note:

When executing a `DBMS_WM` procedure from another procedure, the privilege checks take into account whether the procedure has definer's rights or the rights of the database user whose privileges are currently active.

- [Add\\_Topo\\_Geometry\\_Layer](#)
- [AddAsParentWorkspace](#)
- [AddUserDefinedHint](#)
- [AlterSavepoint](#)

- AlterVersionedTable
- AlterWorkspace
- BeginBulkLoading
- BeginDDL
- BeginResolve
- ChangeWorkspaceType
- CommitBulkLoading
- CommitDDL
- CommitResolve
- CompressWorkspace
- CompressWorkspaceTree
- CopyForUpdate
- CopyWorkspace
- CreateSavepoint
- CreateWorkspace
- Delete\_Topo\_Geometry\_Layer
- DeleteSavepoint
- DisableVersioning
- EnableVersioning
- Export
- Export\_Schemas
- FindRICSet
- FreezeWorkspace
- GetBulkLoadVersion
- GetConflictWorkspace
- GetDiffVersions
- GetLockMode
- GetMultiWorkspaces
- GetOpContext
- GetOriginalDDL
- GetPhysicalTableName
- GetPrivs
- GetSessionInfo
- GetSystemParameter
- GetValidFrom
- GetValidTill
- GetVersion



- GetWMMetadataSpace
- GetWorkspace
- GotoDate
- GotoSavepoint
- GotoWorkspace
- GrantGraphPriv
- GrantPrivsOnPolicy
- GrantSystemPriv
- GrantWorkspacePriv
- Import
- Import\_Schemas
- Initialize\_After\_Import
- IsWorkspaceOccupied
- LockRows
- MergeTable
- MergeWorkspace
- Move\_Proc
- PurgeTable
- RecoverAllMigratingTables
- RecoverFromDroppedUser
- RecoverMigratingTable
- RefreshTable
- RefreshWorkspace
- RemoveAsParentWorkspace
- RemoveDeferredWorkspaces
- RemoveUserDefinedHint
- RemoveWorkspace
- RemoveWorkspaceTree
- RenameSavepoint
- RenameWorkspace
- ResolveConflicts
- RevokeGraphPriv
- RevokeSystemPriv
- RevokeWorkspacePriv
- RollbackBulkLoading
- RollbackDDL
- RollbackResolve

- RollbackTable
- RollbackToSP
- RollbackWorkspace
- SetCaptureEvent
- SetCompressWorkspace
- SetConflictWorkspace
- SetDiffVersions
- SetLockingOFF
- SetLockingON
- SetMultiWorkspaces
- SetSystemParameter
- SetTriggerEvents
- SetValidTime
- SetValidTimeFilterOFF
- SetValidTimeFilterON
- SetWMValidUpdateModeOFF
- SetWMValidUpdateModeON
- SetWoOverwriteOFF
- SetWoOverwriteON
- SetWorkspaceLockModeOFF
- SetWorkspaceLockModeON
- UnfreezeWorkspace
- UnlockRows
- UseDefaultValuesForNulls

## 4.1 Add\_Topo\_Geometry\_Layer

Adds a topology geometry layer from a version-enabled feature table to a topology.

### Format

```
DBMS_WM.Add_Topo_Geometry_Layer(  
  topology      IN VARCHAR2,  
  table_name    IN VARCHAR2,  
  column_name   IN VARCHAR2,  
  tg_layer_type IN VARCHAR2);
```

## Parameters

**Table 4-1 Add\_Topo\_Geometry\_Layer Procedure Parameters**

Parameter	Description
topology	Topology to which to add the topology geometry layer containing the topology geometries in the specified column. The topology must have been created using the SDO_TOPO.CREATE_TOPOLOGY procedure.
table_name	Name of the topology geometry layer table containing the column specified in column_name.
column_name	Name of the column (of type SDO_TOPO_GEOMETRY) containing the topology geometries in the topology geometry layer to be added to the topology.
tg_layer_type	Type of topology geometry layer: POINT, LINE, CURVE, or POLYGON.

## Usage Notes

This procedure has the same format and meaning as the SDO\_TOPO.ADD\_TOPO\_GEOMETRY\_LAYER procedure, which is documented in *Oracle Spatial and Graph Topology Data Model and Network Data Model Graph Developer's Guide*. However, you must use DBMS\_WM.Add\_Topo\_Geometry\_Layer, and not SDO\_TOPO.ADD\_TOPO\_GEOMETRY\_LAYER, to add a topology geometry layer from a version-enabled feature table to a topology. For information about Workspace Manager support for topologies, see [Spatial and Graph Topology Support](#).

The first call to this procedure for a given topology creates the <topology-name>\_RELATION\$ table, which is described in *Oracle Spatial and Graph Topology Data Model and Network Data Model Graph Developer's Guide*.

An exception is raised if topology, table\_name, or column\_name does not exist, if topology or table\_name is not version-enabled, or if tg\_layer\_type is not one of the supported values.

## Examples

The following example adds a topology geometry layer to the CITY\_DATA topology. The topology geometry layer consists of polygon geometries in the FEATURE column of the LAND\_PARCELS table.

```
EXECUTE DBMS_WM.Add_Topo_Geometry_Layer('CITY_DATA', 'LAND_PARCELS', 'FEATURE',
'POLYGON');
```

## 4.2 AddAsParentWorkspace

Adds a workspace as a parent workspace to a child workspace in a multiparent workspace environment.

### Syntax

```
DBMS_WM.AddAsParentWorkspace(
    workspace          IN VARCHAR2,
    parent_workspace  IN VARCHAR2,
    auto_commit        IN BOOLEAN DEFAULT TRUE);
```

## Parameters

**Table 4-2 AddAsParentWorkspace Procedure Parameters**

Parameter	Description
<code>workspace</code>	Name of the workspace to which to add the parent workspace. The name is case-sensitive.
<code>parent_workspace</code>	Name of the workspace to add as a parent workspace of <code>workspace</code> . The name is case-sensitive.
<code>auto_commit</code>	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a> .

## Usage Notes

This procedure is part of the support for the multiparent workspaces feature, which is described in [Multiparent Workspaces](#). If `workspace` has only one parent workspace, this procedure makes `workspace` a multiparent workspace. If `workspace` is already a multiparent workspace, this procedure adds another parent workspace to `workspace`.

An exception is raised if one or more of the following apply:

- The value of the Workspace Manager system parameter `ALLOW_MULTI_PARENT_WORKSPACES` is OFF.
- The value of the Workspace Manager system parameter `CR_WORKSPACE_MODE` or `NONR_WORKSPACE_MODE` (whichever is applicable, depending on whether or not `workspace` is a continually refreshed workspace) is `OPTIMISTIC_LOCKING`.
- `workspace` or `parent_workspace` does not exist.
- `parent_workspace` is already in the ancestor hierarchy of `workspace`.
- `auto_commit` is TRUE and an open transaction exists in a parent or child workspace of any table that needs to be modified.
- There is a violation of a primary key constraint, referential integrity constraint, or unique constraint in the view of the data in a version-enabled table in `workspace`.

## Examples

The following example adds `Workspace4` as a parent workspace of `Workspace3`. (See the hierarchy illustration in [Multiparent Workspaces](#).)

```
-- Allow multiparent workspaces. (Required for AddAsParentWorkspace)
EXECUTE DBMS_WM.SetSystemParameter ('ALLOW_MULTI_PARENT_WORKSPACES', 'ON');
-- Make Workspace3 multiparent by adding Workspace4 as a parent.
EXECUTE DBMS_WM.AddAsParentWorkspace ('Workspace3', 'Workspace4');
```

## 4.3 AddUserDefinedHint

Adds a user-defined hint: that is, modifies (and thus overrides) a default optimizer hint, with the goal of improving the performance of SQL statements executed by the DBMS\_WM package on a specified version-enabled table or all version-enabled tables.

### Syntax

```
DBMS_WM.AddUserDefinedHint(  
    hint_id    IN NUMBER,  
    table_id   IN VARCHAR2 DEFAULT NULL,  
    hint       IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 4-3 AddUserDefinedHint Procedure Parameters**

Parameter	Description
hint_id	Numeric ID that uniquely identifies the user-defined hint. Must match an existing hint ID used by Workspace Manager for one or more SQL statements.
table_id	Name of the table to which to apply the hint. The name is not case-sensitive. If this value is null, the hint is used with all version-enabled tables for any SQL statements that specify the hint.
hint	The text of the optimizer hint. For an explanation of optimizer hints, see the chapter about using optimizer hints in <i>Oracle Database SQL Tuning Guide</i> .

### Usage Notes

Use this procedure only if you are dissatisfied with the performance of any DBMS\_WM package operations, and if you know how to use application tracing and SQL optimizer hints. For information about tracing, see the chapter about application tracing tools in *Oracle Database SQL Tuning Guide*.

In the trace output, any SQL statements using the DBMS\_WM package that allow a user-defined hint include one or more comments in the following format:

```
/* WM$SQL (hint_id) (table_id) */
```

If you have identified a statement that is performing poorly, and if you know an optimizer hint that will improve performance, you can use the `AddUserDefinedHint` procedure to specify the hint that should be used for the specified hint ID. You can also indicate whether to use the specified hint associated with the hint ID only for a specified table, or for all tables.

If you specify the `table_id` parameter, the specified hint will be used only when SQL statements that use the hint ID access the specified table, and the default Workspace Manager-supplied hint will be used with other tables. If the `table_id` parameter is null, the specified hint will be used when any DBMS\_WM statement use the hint ID.

If the `hint` parameter specifies an object name (such as an index name), the `table_id` parameter must not be null.

Any table aliases can be used within user-defined hints; however, standard scoping rules still apply.

To remove a user-defined hint (that is, to cause the default hint associated with a hint ID to be used), use the [RemoveUserDefinedHint](#) procedure.

### Examples

The following example specifies a full table scan on the TABLE1 table and any associated Workspace Manager infrastructure tables when a SQL statement specifies hint ID 1101 with the SCOTT.TABLE1 table.

```
EXECUTE DBMS_WM.AddUserDefinedHint (1101, 'scott.table1', 'full(t1)');
```

## 4.4 AlterSavepoint

Modifies the description of a savepoint.

### Syntax

```
DBMS_WM.AlterSavepoint(
  workspace      IN VARCHAR2,
  sp_name        IN VARCHAR2,
  sp_description IN VARCHAR2);
```

### Parameters

**Table 4-4 AlterSavepoint Procedure Parameters**

Parameter	Description
workspace	Name of the workspace in which the savepoint was created. The name is case-sensitive.
sp_name	Name of the savepoint. The name is case-sensitive.
sp_description	Description of the savepoint.

### Usage Notes

To see the current description of the savepoint, examine the `DESCRIPTION` column value for the savepoint in the [ALL\\_WORKSPACE\\_SAVEPOINTS](#) metadata view, which is described in [ALL\\_WORKSPACE\\_SAVEPOINTS](#).

An exception is raised if the user is not the workspace owner or savepoint owner or does not have the `WM_ADMIN` system privilege.

### Examples

The following example modifies the description of savepoint SP1 in the `NEWWORKSPACE` workspace.

```
EXECUTE DBMS_WM.AlterSavepoint ('NEWWORKSPACE', 'SP1', 'First set of changes for scenario');
```

## 4.5 AlterVersionedTable

Alters a version-enabled table to add valid time support, rename a constraint, or rename an index.

### Syntax

```
DBMS_WM.AlterVersionedTable(
    table_name          IN VARCHAR2,
    alter_option        IN VARCHAR2,
    parameter_options   IN VARCHAR2 DEFAULT NULL,
    ignore_last_error   IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 4-5 AlterVersionedTable Procedure Parameters**

Parameter	Description
table_name	Name of the version-enabled table to which to add valid time support. The name is not case-sensitive.
alter_option	One of the following values: <code>ADD_VALID_TIME</code> to add valid time support, <code>DDL</code> to make DDL changes, <code>RENAME_CONSTRAINT</code> to rename a constraint, <code>REBUILD_INDEX</code> to rebuild an index, <code>RENAME_INDEX</code> to rename an index, or either <code>USE_SCALAR_TYPES_FOR_VALIDTIME</code> or <code>USE_WM_PERIOD_FOR_VALIDTIME</code> to specify whether views on an existing version-enabled table should use two scalar columns for the valid time range. See the Usage Notes for information about these options, including when you must and can use this procedure to rename an index or a constraint.
parameter_options	A quoted string (in the general format 'keyword=value, keyword2=value2, ...') containing keywords valid for the specified <code>alter_option</code> parameter value. See the Usage Notes for keywords that are valid for each <code>alter_option</code> parameter value.
ignore_last_error	A Boolean value ( <code>TRUE</code> or <code>FALSE</code> ). <code>TRUE</code> ignores the last error, if any, that occurred during the previous call to the <code>AlterVersionedTable</code> procedure. Information about the last error is stored in the <code>USER_WM_VT_ERRORS</code> and <code>ALL_WM_VT_ERRORS</code> static data dictionary views, which are described in <a href="#">Workspace Manager Static Data Dictionary Views</a> . See the Usage Notes for more information. <code>FALSE</code> (the default) does not ignore the last error, if any, that occurred during the previous call to the <code>AlterVersionedTable</code> procedure.

### Usage Notes

Use this procedure to add valid time support, rename a constraint, or rename an index for an existing version-enabled table. For more information about adding valid time support, see [Adding Valid Time Support to an Existing Table](#).

If the `alter_option` value is `ADD_VALID_TIME`, you can specify none, one, or more of the following `parameter_options` keywords:

- `validFrom`: Starting time period to be set in the `WM_VALID` column of all existing rows. The default value is the current timestamp.

- `validTill`: Ending time period to be set in the `WM_VALID` column of all existing rows. The default value is `UNTIL_CHANGED`.
- `fmt`: Date format. The default value is `'mmddyyyyhh24miss'`. The options are the same as for the `TO_TIMESTAMP_TZ` function, which is described in *Oracle Database SQL Language Reference*.
- `nlsparam`: Globalization support options. The options and default are the same as for the `nlsparam` argument to the `TO_CHAR` function for date conversion, which is described in *Oracle Database SQL Language Reference*.

If the `alter_option` value is `DDL`, the currently supported operations for this procedure are adding, merging, and splitting table partitions. You must have `SYSDBA` privileges, and you must specify the following `parameter_options` keywords:

- `ddl`: The DDL (data definition language) statement to be executed. The DDL statement must refer to the fully qualified base table (for example, `SCOTT.EMP_LT` if `SCOTT.EMP` is the version-enabled table).
- `force`: A value of `true` causes Workspace Manager to attempt to execute the DDL statement, regardless of whether the operation is officially supported for this procedure; a value of `false` (the default) causes Workspace Manager not to attempt to execute the DDL statement. Thus, to execute the DDL statement, you must override the default value by explicitly specifying `'force=true'`; however, do not specify `'force=true'` unless you know what you are doing.

If the `alter_option` value is `RENAME_CONSTRAINT`, you must specify both of the following `parameter_options` keywords:

- `constraint_name`: The current name of the constraint to be renamed. The name is not case-sensitive.
- `new_constraint_name`: The new name for the constraint. The name is not case-sensitive.

If the `alter_option` value is `RENAME_INDEX`, you must specify all of the following `parameter_options` keywords:

- `index_owner`: The name of the schema that owns the index to be renamed. The schema name is not case-sensitive.
- `index_name`: The current name of the index to be renamed. The name is not case-sensitive.
- `new_index_name`: The new name for the index. The name is not case-sensitive.

If the name of a constraint or index on a version-enabled table is longer than 26 characters, you must use the `AlterVersionedTable` procedure if you want to rename the constraint or index; you cannot use the `ALTER TABLE` (for a constraint) or `ALTER INDEX` (for an index) statement with the `RENAME` clause. If you use the `AlterVersionedTable` procedure, you do not need to include it between calls to the [BeginDDL](#) and [CommitDDL](#) procedures.

If the name of the constraint or index on a version-enabled table is 26 or fewer characters long, you can do either of the following to rename the constraint or index: use the `AlterVersionedTable` procedure, or use the `ALTER TABLE` (for a constraint) or `ALTER INDEX` (for an index) statement with the `RENAME` clause between calls to the [BeginDDL](#) and [CommitDDL](#) procedures (as explained in [DDL Operations Related to Version-Enabled Tables](#)).



If the `alter_option` value is `REBUILD_INDEX`, you must specify the `index_owner` and `index_name` keywords to identify the database user that owns the index and the name of the index; and you can specify either the `reverse` or `noreverse` keyword, to specify whether or not to store the bytes of the index block in reverse order, excluding the rowid.

The `alter_option` values `USE_SCALAR_TYPES_FOR_VALIDTIME` and `USE_WM_PERIOD_FOR_VALIDTIME` can be used only to change the views on an existing version-enabled table to be consistent with the current setting for the Workspace Manager system parameter `USE_SCALAR_TYPES_FOR_VALIDTIME` (described in [System Parameters for Workspace Manager](#)). For example, if you set the Workspace Manager system parameter `USE_SCALAR_TYPES_FOR_VALIDTIME` to `ON`, but an existing version-enabled table named `MYTABLE` has views that use a single column named `WM_VALID` (of type `WM_PERIOD`) to indicate the valid time range, you can change the views on `MY_TABLE` to use two columns of type `TIMESTAMP WITH TIME ZONE` by calling the `AlterVersionedTable` procedure and specifying the `alter_option` value `USE_SCALAR_TYPES_FOR_VALIDTIME`.

The `alter_option` parameter cannot be used to override the current value of the Workspace Manager system parameter `USE_SCALAR_TYPES_FOR_VALIDTIME`. If the system parameter value is `ON`, the `alter_option` parameter value must be `USE_SCALAR_TYPES_FOR_VALIDTIME`; and if the system parameter value is `OFF`, the `alter_option` parameter value must be `USE_WM_PERIOD_FOR_VALIDTIME`.

You can use double quotation marks for parameter values within the `parameter_options` string. For example, the following two specifications are semantically identical:

```
'index_owner=scott, index_name=my_index, new_index_name=my_new_index'
'index_owner="scott", index_name="my_index", new_index_name="my_new_index"'
```

If a call to the `AlterVersionedTable` procedure fails, you should try to fix the cause of the error. Examine the [USER\\_WM\\_VT\\_ERRORS](#) and [ALL\\_WM\\_VT\\_ERRORS](#) static data dictionary views to see the SQL statement and error message. Fix the cause of the error, and then call the `AlterVersionedTable` procedure again with the default `ignore_last_error` parameter value of `FALSE`. However, if the call still fails and you cannot fix the cause of the error, and if you are sure that it is safe and appropriate to ignore this error, then you can ignore the error by calling the `AlterVersionedTable` procedure with the `ignore_last_error` parameter value of `TRUE`. Note that you are responsible for ensuring that it is safe and appropriate to ignore the error.

An exception is raised if one or more of the following apply:

- `table_name` does not exist.
- `alterOptions` is not `ADD_VALID_TIME`.

## Examples

The following example creates a table named `MY_TABLE`, version-enables it without valid time support, and then adds valid time support. After valid time support is added, the `WM_VALID` column contains the default valid time period.

```
CREATE TABLE my_table (id NUMBER PRIMARY KEY);
EXECUTE DBMS_WM.EnableVersioning ('my_table');
INSERT INTO my_table VALUES (1);
SELECT * FROM my_table;
```

```
          ID
-----
         1
```

```
EXECUTE DBMS_WM.AlterVersionedTable('my_table', 'ADD_VALID_TIME');
SELECT * FROM my_table;
```

```

          ID
-----
WM_VALID (VALIDFROM, VALIDTILL)
-----
          1
WM_PERIOD('09-JUN-2003 10:04:13 -04:00', NULL)
```

The following example creates a table named SCOTT.MY\_TABLE, creates an index named MY\_INDEX on the VALUE column in that table, version-enables the table, and then renames the index to MY\_NEW\_INDEX.

```
CREATE TABLE scott.my_table (id NUMBER PRIMARY KEY, value INTEGER);
CREATE INDEX scott.my_index on scott.my_table(value);
EXECUTE DBMS_WM.EnableVersioning ('scott.my_table');
EXECUTE DBMS_WM.AlterVersionedTable ('scott.my_table', 'RENAME_INDEX',
    'index_owner=scott, index_name=my_index, new_index_name=my_new_index');
```

## 4.6 AlterWorkspace

Modifies the description of a workspace.

### Syntax

```
DBMS_WM.AlterWorkspace(
    workspace          IN VARCHAR2,
    workspace_description IN VARCHAR2);
```

### Parameters

**Table 4-6** AlterWorkspace Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case-sensitive.
workspace_description	Description of the workspace.

### Usage Notes

To see the current description of the workspace, examine the DESCRIPTION column value for the savepoint in the [ALL\\_WORKSPACES](#) metadata view, which is described in [ALL\\_WORKSPACES](#).

An exception is raised if the user is not the workspace owner or does not have the WM\_ADMIN system privilege.

### Examples

The following example modifies the description of the NEWWORKSPACE workspace.

```
EXECUTE DBMS_WM.AlterWorkspace ('NEWWORKSPACE', 'Testing proposed scenario B');
```

## 4.7 BeginBulkLoading

Starts the bulk loading process for a version-enabled table.

### Syntax

```
DBMS_WM.BeginBulkLoading(
  table_name          IN VARCHAR2,
  workspace           IN VARCHAR2,
  version             IN INTEGER DEFAULT NULL,
  check_for_duplicates IN BOOLEAN DEFAULT TRUE,
  ignore_last_error   IN BOOLEAN DEFAULT FALSE,
  single_transaction  IN BOOLEAN DEFAULT FALSE,
  savepoint_name      IN DEFAULT LATEST);
```

### Parameters

**Table 4-7 BeginBulkLoading Procedure Parameters**

Parameter	Description
table_name	Name of the version-enabled table into which data will be bulk loaded. The name is not case-sensitive.
workspace	Name of the workspace in which bulk loading will be performed. The name is case-sensitive.
version	(Ignored for the current release. Before Release 12.1, this was the version number returned by the <a href="#">GetBulkLoadVersion</a> function.)
check_for_duplicates	A Boolean value (TRUE or FALSE). TRUE (the default) checks for rows in the data to be bulk loaded that have the same values in primary key columns. For any duplicate records, only the record with the lowest ROWID value is kept in the table, and the rest are moved to the discards table specified in the call to the <a href="#">CommitBulkLoading</a> procedure. See the Usage Notes for more information about this parameter. FALSE does not check if any rows in the data to be bulk loaded have the same values in primary key columns.
ignore_last_error	A Boolean value (TRUE or FALSE). TRUE ignores the last error, if any, that occurred during the previous call to the <a href="#">BeginBulkLoading</a> procedure. Information about the last error is stored in the <a href="#">USER_WM_VT_ERRORS</a> and <a href="#">ALL_WM_VT_ERRORS</a> static data dictionary views, which are described in <a href="#">Workspace Manager Static Data Dictionary Views</a> . See the Usage Notes for more information. FALSE (the default) does not ignore the last error, if any, that occurred during the previous call to the <a href="#">BeginBulkLoading</a> procedure.

Table 4-7 (Cont.) BeginBulkLoading Procedure Parameters

Parameter	Description
<code>single_transaction</code>	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE causes Workspace Manager not to perform an internal commit operation after each of several steps that it will perform after you call the <a href="#">CommitBulkLoading</a> procedure, but instead to perform a commit only after it has performed all the necessary steps. TRUE also allows queries to be made on the version-enabled table.</p> <p>FALSE (the default) causes Workspace Manager to perform an internal commit operation after each of several steps that it will perform after you call the <a href="#">CommitBulkLoading</a> procedure, and it also disallows queries to be made on the table until a <a href="#">CommitBulkLoading</a> or <a href="#">RollbackBulkLoading</a> operation is complete.</p> <p>See the Usage Notes for more information about this parameter.</p>
<code>savepoint_name</code>	<p>The version in the workspace in which data will be bulk loaded. If specified, must be one of the following: LATEST or ROOT_VERSION.</p> <p>LATEST (the default) is the current version in the workspace.</p> <p>ROOT_VERSION is the root version (version number 0, which is in the LIVE workspace). The root version is the ancestor of all other versions, so data in the root version is visible from all other workspaces (unless non-LIVE workspaces have updated the data). You can specify ROOT_VERSION only if workspace is LIVE.</p>

### Usage Notes

Before you can begin bulk loading data into a version-enabled table, you must call the [BeginBulkLoading](#) procedure. You must end the bulk loading session by calling either the [CommitBulkLoading](#) procedure (to commit changes made when the data was loaded) or the [RollbackBulkLoading](#) procedure (to roll back changes made when the data was loaded). For more information about bulk loading with Workspace Manager, see [Bulk Loading into Version-Enabled Tables](#).

If `single_transaction` is FALSE (the default), the [BeginBulkLoading](#) procedure drops some internal Workspace Manager views on the table, to prevent DML operations and certain Workspace Manager operations on the table; however, this also prevents any queries from being made using the specified version-enabled table. Regardless of the `single_transaction` parameter value, and especially if it is FALSE, you should complete the bulk loading as quickly as possible and at a time when applications and users will not need to access the table. The value of the `single_transaction` parameter must be the same for both the [BeginBulkLoading](#) and [CommitBulkLoading](#) procedures for a bulk loading session with a specified table.

A TRUE value for the `check_for_duplicates` parameter does not cause any existing data in the version-enabled table to be checked. If an existing row in the version in which data is being bulk loaded (which could be the latest version of a workspace or the root version) has the same primary key values as a row in the data to be bulk loaded, the behavior depends on the history option setting for the table: if `VIEW_WO_OVERWRITE` is set, the newly loaded row is chained to the existing row that has the same primary key values; if `VIEW_WO_OVERWRITE` is not set, the new data is not bulk loaded but is instead moved to the discards table.

If a call to the `BeginBulkLoading` procedure fails, you should try to fix the cause of the error. Examine the `USER_WM_VT_ERRORS` and `ALL_WM_VT_ERRORS` static data dictionary views to see the SQL statement and error message. Fix the cause of the error, and then call the `BeginBulkLoading` procedure again with the default `ignore_last_error` parameter value of `FALSE`. However, if the call still fails and you cannot fix the cause of the error, and if you are sure that it is safe and appropriate to ignore this error, then you have the option to ignore the error by calling the `BeginBulkLoading` procedure with the `ignore_last_error` parameter value of `TRUE`. Note that you are responsible for ensuring that it is safe and appropriate to ignore the error.

If performance is an issue, carefully consider whether or not you need to check for duplicate records, because a `check_for_duplicates` value of `TRUE` (the default) causes Workspace Manager to perform additional internal processing.

An exception is raised if one or more of the following apply:

- `table_name` does not exist.
- `table_name` is not version-enabled.
- The user does not own the table or does not have the `WM_ADMIN` system privilege.

### Examples

The following example starts the bulk load operation into the `EMP` table in the `W1` workspace.

```
EXECUTE DBMS_WM.BeginBulkLoading ('EMP', 'W1');
```

## 4.8 BeginDDL

Starts a DDL (data definition language) session for a specified table.

### Syntax

```
DBMS_WM.BeginDDL(
    table_name IN VARCHAR2);
```

### Parameters

**Table 4-8 BeginDDL Procedure Parameters**

Parameter	Description
<code>table_name</code>	Name of the version-enabled table. The name is not case-sensitive.

### Usage Notes

This procedure starts a DDL session, and it creates a special table whose name is the same as `table_name` but with `_LTS` added to the table name. After calling this procedure, you can perform one or more DDL operations on the table or any indexes or triggers that are based on the table, and then call either the `CommitDDL` or `RollbackDDL` procedure.

In addition to creating the special `<table-name>_LTS` table, the procedure creates other objects:

- The `<table-name>_LTS` table has the same triggers, columns, and indexes as the `<table-name>` table.

- For each parent table with which the `<table-name>` table has a referential integrity constraint, the same constraint is defined for the `<table-name>_LTS` table.
- Triggers, columns, and referential integrity constraints on the `<table-name>_LTS` table have the same names as the corresponding ones on the `<table-name>` table.
- For each index on the `<table-name>` table, the corresponding index on the `<table-name>_LTS` table has a name in the form `<index-name>_LTS`.
- The primary key constraint on the `<table-name>_LTS` table has a name in the form `<primary-key>_LTS`.
- All unique constraints on the `<table-name>_LTS` table have a name in the form `<unique-constraint-name>_LTS`.

For detailed information about performing DDL operations related to version-enabled tables, see [DDL Operations Related to Version-Enabled Tables](#).

An exception is raised if one or more of the following apply:

- `table_name` does not exist or is not version-enabled.
- `table_name` has a domain index defined on it, and the user has not been directly granted the `CREATE TABLE` and `CREATE SEQUENCE` privileges.
- An open DDL session exists for `table_name`. (That is, the `BeginDDL` procedure has already been called specifying this table, and the [CommitDDL](#) or [RollbackDDL](#) procedure has not been called specifying this table.)

### Examples

The following example begins a DDL session, adds a column named `COMMENTS` to the `COLA_MARKETING_BUDGET` table by using the special table named `COLA_MARKETING_BUDGET_LTS`, and ends the DDL session by committing the change.

```
EXECUTE DBMS_WM.BeginDDL('COLA_MARKETING_BUDGET');
ALTER TABLE cola_marketing_budget_lts ADD (comments VARCHAR2(100));
EXECUTE DBMS_WM.CommitDDL('COLA_MARKETING_BUDGET');
```

## 4.9 BeginResolve

Starts a conflict resolution session.

### Syntax

```
DBMS_WM.BeginResolve(
    workspace IN VARCHAR2);
```

### Parameters

**Table 4-9 BeginResolve Procedure Parameters**

Parameter	Description
<code>workspace</code>	Name of the workspace. The name is case-sensitive.

## Usage Notes

This procedure starts a conflict resolution session. While this procedure is executing, the workspace is frozen in `lWRITER` mode, as explained in [Freezing and Unfreezing Workspaces](#).

After calling this procedure, you can execute the [ResolveConflicts](#) procedure as needed for various tables that have conflicts, and then call either the [CommitResolve](#) or [RollbackResolve](#) procedure. For more information about conflict resolution, see [Resolving Conflicts Before a Merge or Refresh Operation](#).

An exception is raised if one or more of the following apply:

- There are one or more open database transactions in `workspace`.
- The user executing the [BeginResolve](#) procedure does not have the privilege to access `workspace` and its parent workspace.

## Examples

The following example starts a conflict resolution session in `Workspace1`.

```
EXECUTE DBMS_WM.BeginResolve ('Workspace1');
```

# 4.10 ChangeWorkspaceType

Changes a workspace from not continually refreshed to continually refreshed. (*Continually refreshed workspaces* are explained in [Continually Refreshed Workspaces](#).)

## Syntax

```
DBMS_WM.ChangeWorkspaceType(
  workspace      IN VARCHAR2,
  workspace_type IN VARCHAR2 DEFAULT DBMS_WM.CR_WORKSPACE_TYPE,
  auto_commit    IN BOOLEAN DEFAULT TRUE);
```

## Parameters

**Table 4-10** ChangeWorkspaceType Procedure Parameters

Parameter	Description
<code>workspace</code>	Name of the workspace. The name is case-sensitive.
<code>workspace_type</code>	Must be <code>DBMS_WM.CR_WORKSPACE_TYPE</code> (the default), for continually refreshed.
<code>auto_commit</code>	A Boolean value ( <code>TRUE</code> or <code>FALSE</code> ).  <code>TRUE</code> (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes.  <code>FALSE</code> causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a> .

## Usage Notes

For this release, you can only change a workspace that is not continually refreshed to continually refreshed; you cannot change a continually refreshed workspace to not continually refreshed.

An exception is raised if one or more of the following occur:

- The user is not the owner of `workspace`, and the user does not have the `WM_ADMIN` system privilege.
- `workspace_type` is not valid.
- `auto_commit` is `TRUE` and an open transaction exists in a parent or child workspace of any table that needs to be modified.
- The workspace type cannot be changed. For example, the change cannot be made if the Workspace Manager system parameter `CR_WORKSPACE_MODE` is set to `PESSIMISTIC_LOCKING`, but the `NONCR_WORKSPACE_MODE` parameter is set to `OPTIMISTIC_LOCKING` and there is versioned data in any continually refreshed workspace.

## Examples

The following example changes the `NEWWORKSPACE` workspace type from not continually refreshed to continually refreshed.

```
EXECUTE DBMS_WM.ChangeWorkspaceType ('NEWWORKSPACE');
```

## 4.11 CommitBulkLoading

Ends the bulk loading process for a version-enabled table by committing the bulk load changes.

### Syntax

```
DBMS_WM.CommitBulkLoading(
  table_name          IN VARCHAR2,
  discards_table      IN VARCHAR2,
  check_for_duplicates IN BOOLEAN DEFAULT TRUE,
  enforceUCFlag       IN BOOLEAN DEFAULT TRUE,
  enforceRICFlag      IN BOOLEAN DEFAULT TRUE,
  ignore_last_error   IN BOOLEAN DEFAULT FALSE,
  single_transaction  IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 4-11** CommitBulkLoading Procedure Parameters

Parameter	Description
<code>table_name</code>	Name of the version-enabled table into which data has been bulk loaded. The name is not case-sensitive.
<code>discards_table</code>	Name of the table into which discard records are inserted. The name is not case-sensitive. If the table does not already exist, it is created.



**Table 4-11 (Cont.) CommitBulkLoading Procedure Parameters**

Parameter	Description
<code>check_for_duplicates</code>	<p>A Boolean value (TRUE or FALSE). Note: Effective with Release 12.1, this parameter is ignored; the value from the call to the <a href="#">BeginBulkLoading</a> procedure is used.</p> <p>TRUE (the default) checks for rows in the data to be bulk loaded that have the same values in primary key columns. For any duplicate records, only the record with the lowest ROWID value is kept in the table, and the rest are moved to the discards table. See the Usage Notes for more information about this parameter.</p> <p>FALSE does not check if any rows in the data to be bulk loaded have the same values in primary key columns.</p>
<code>enforceUCFlag</code>	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE (the default) enforces any unique constraints defined on <code>to_table</code>, ensuring that the bulk load operation does not violate any such constraints.</p> <p>FALSE does not enforce any unique constraints defined on <code>to_table</code> for the bulk load operation.</p>
<code>enforceRICFlag</code>	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE (the default) enforces any referential integrity constraints defined on <code>to_table</code>, ensuring that the bulk load operation does not violate any such constraints.</p> <p>FALSE does not enforce any referential integrity constraints defined on <code>to_table</code> for the bulk load operation.</p>
<code>ignore_last_error</code>	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE ignores the last error, if any, that occurred during the previous call to the <code>CommitBulkLoading</code> procedure. Information about the last error is stored in the <code>USER_WM_VT_ERRORS</code> and <code>ALL_WM_VT_ERRORS</code> static data dictionary views, which are described in <a href="#">Workspace Manager Static Data Dictionary Views</a>. See the Usage Notes for more information.</p> <p>FALSE (the default) does not ignore the last error, if any, that occurred during the previous call to the <code>CommitBulkLoading</code> procedure.</p>
<code>single_transaction</code>	<p>A Boolean value (TRUE or FALSE). Note: Effective with Release 12.1, this parameter is ignored; the value from the call to the <a href="#">BeginBulkLoading</a> procedure is used.</p> <p>TRUE causes Workspace Manager not to perform an internal commit operation after each of several steps that it performs after you call the <code>CommitBulkLoading</code> procedure, but instead to perform a commit only after it has performed all the necessary steps.</p> <p>FALSE (the default) causes Workspace Manager to perform an internal commit operation after each of several steps that it performs after you call the <code>CommitBulkLoading</code> procedure.</p> <p>The value of this parameter must be the same as when you called the <code>BeginBulkLoading</code> procedure specifying the table in <code>table_name</code>.</p>

## Usage Notes

For information about the requirements for bulk loading data into version-enabled tables, see [Bulk Loading into Version-Enabled Tables](#).

This procedure generates versioning metadata for newly loaded data and synchronizes the newly loaded data with the existing versioned data in the table. It can also enforce unique and referential constraints on the newly loaded data. It re-creates all the views that were dropped by the [BeginBulkLoading](#) procedure.

A `TRUE` value for the `check_for_duplicates` parameter does not cause any existing data in the version-enabled table to be checked. If an existing row in the version in which data is being bulk loaded (which could be the latest version of a workspace or the root version) has the same primary key values as a row in the data to be bulk loaded, the behavior depends on the history option setting for the table: if `VIEW_WO_OVERWRITE` is set, the newly loaded row is chained to the existing row that has the same primary key values; if `VIEW_WO_OVERWRITE` is not set, the new data is not bulk loaded but is instead moved to the discards table.

If a call to the `CommitBulkLoading` procedure fails, you should try to fix the cause of the error. Examine the [USER\\_WM\\_VT\\_ERRORS](#) and [ALL\\_WM\\_VT\\_ERRORS](#) static data dictionary views to see the SQL statement and error message. Fix the cause of the error, and then call the `CommitBulkLoading` procedure again with the default `ignore_last_error` parameter value of `FALSE`. However, if the call still fails and you cannot fix the cause of the error, and if you are sure that it is safe and appropriate to ignore this error, then you have the option to ignore the error by calling the `CommitBulkLoading` procedure with the `ignore_last_error` parameter value of `TRUE`. Note that you are responsible for ensuring that it is safe and appropriate to ignore the error.

Note the following performance considerations:

- A `TRUE` value for `check_for_duplicates` requires additional processing time, and a `TRUE` value for `enforceUCFlag` or `enforceRICFlag` may require additional processing time.
- If performance is an issue, carefully consider whether or not you need to check for duplicate records.
- If the table does not have unique or referential constraints, setting the `enforceUCFlag` or `enforceRICFlag` parameter to `TRUE` does not have a significant effect on performance.

An exception is raised if one or more of the following apply:

- `table_name` does not exist.
- `table_name` is not version-enabled.
- The [BeginBulkLoading](#) procedure has not been called on the table.
- The user does not own the table or does not have the `WM_ADMIN` system privilege.

## Examples

The following example commits changes made to the `EMP` table during a bulk load operation, and specifies `DISCARDS` as the table to hold discard records.

```
EXECUTE DBMS_WM.CommitBulkLoading ('EMP', 'DISCARDS');
```

## 4.12 CommitDDL

Commits DDL (data definition language) changes made during a DDL session for a specified table, and ends the DDL session.

### Syntax

```
DBMS_WM.CommitDDL(
  table_name           IN VARCHAR2,
  ignore_last_error   IN BOOLEAN DEFAULT FALSE,
  enforce_unique_constraints IN BOOLEAN DEFAULT FALSE,
  enforce_RICs        IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 4-12 CommitDDL Procedure Parameters**

Parameter	Description
table_name	Name of the version-enabled table. The name is not case-sensitive.
ignore_last_error	A Boolean value (TRUE or FALSE). TRUE ignores the last error, if any, that occurred during the previous call to the CommitDDL procedure. Information about the last error is stored in the <a href="#">USER_WM_VT_ERRORS</a> and <a href="#">ALL_WM_VT_ERRORS</a> static data dictionary views, which are described in <a href="#">Workspace Manager Static Data Dictionary Views</a> . See the Usage Notes for more information. FALSE (the default) does not ignore the last error, if any, that occurred during the previous call to the CommitDDL procedure.
enforce_unique_constraints	A Boolean value (TRUE or FALSE). TRUE enforces any unique constraints defined on table_name on existing versioned data in the table. This ensures that the DDL changes do not cause any such constraints to be violated, but it causes Workspace Manager to take additional time to perform the operation. FALSE (the default) does not enforce any unique constraints defined on table_name on existing versioned data in the table.
enforce_RICs	A Boolean value (TRUE or FALSE). TRUE enforces any referential integrity constraints defined on table_name on existing versioned data in the table. This ensures that the changes do not cause any such constraints to be violated, but it causes Workspace Manager to take additional time to perform the operation. FALSE (the default) does not enforce any referential integrity constraints defined on table_name on existing versioned data in the table.

### Usage Notes

This procedure commits changes that were made to a version-enabled table and to any indexes, triggers, and referential integrity constraints based on the version-enabled table

during a DDL session. It also deletes the special `<table-name>_LTS` table that was created by the [BeginDDL](#) procedure.

For detailed information about performing DDL operations related to version-enabled tables, see [DDL Operations Related to Version-Enabled Tables](#).

The `enforce_unique_constraints` and `enforce_RICs` parameter settings apply only to existing versioned data, and do not affect whether or not existing constraints are enforced for future DML operations on the table.

If a call to the `CommitDDL` procedure fails, the table is left in an inconsistent state. If this occurs, you should try to fix the cause of the error. Examine the [USER\\_WM\\_VT\\_ERRORS](#) and [ALL\\_WM\\_VT\\_ERRORS](#) static data dictionary views to see the SQL statement and error message. For example, the `CommitDDL` procedure might have failed because the tablespace was not large enough to add a column. Fix the cause of the error, and then call the `CommitDDL` procedure again with the default `ignore_last_error` parameter value of `FALSE`. However, if the call still fails and you cannot fix the cause of the error, and if you are sure that it is safe and appropriate to ignore this error, then you have the option to ignore the error by calling the `CommitDDL` procedure with the `ignore_last_error` parameter value of `TRUE`. Note that you are responsible for ensuring that it is safe and appropriate to ignore the error.

An exception is raised if one or more of the following apply:

- `table_name` does not exist or is not version-enabled.
- `table_name` has a domain index defined on it, and the user has not been directly granted the `CREATE TABLE` and `CREATE SEQUENCE` privileges.
- An open DDL session does not exist for `table_name`. (That is, the [BeginDDL](#) procedure has not been called specifying this table, or the [CommitDDL](#) or [RollbackDDL](#) procedure was already called specifying this table.)

Some invalid DDL operations also cause an exception when `CommitDDL` procedure is called. See [DDL Operations Related to Version-Enabled Tables](#) for information about DDL operations that are supported.

### Examples

The following example begins a DDL session, adds a column named `COMMENTS` to the `COLA_MARKETING_BUDGET` table by using the special table named `COLA_MARKETING_BUDGET_LTS`, and ends the DDL session by committing the change.

```
EXECUTE DBMS_WM.BeginDDL('COLA_MARKETING_BUDGET');  
ALTER TABLE cola_marketing_budget_lts ADD (comments VARCHAR2(100));  
EXECUTE DBMS_WM.CommitDDL('COLA_MARKETING_BUDGET');
```

## 4.13 CommitResolve

Ends a conflict resolution session and saves (makes permanent) any changes in the workspace since the [BeginResolve](#) procedure was executed.

### Syntax

```
DBMS_WM.CommitResolve(  
    workspace IN VARCHAR2);
```

## Parameters

**Table 4-13 CommitResolve Procedure Parameters**

Parameter	Description
workspace	Name of the workspace. The name is case-sensitive.

## Usage Notes

This procedure ends the current conflict resolution session (started by the [BeginResolve](#) procedure), and saves all changes in the workspace since the start of the conflict resolution session. Contrast this procedure with the [RollbackResolve](#) procedure, which discards all changes.

For more information about conflict resolution, see [Resolving Conflicts Before a Merge or Refresh Operation](#).

An exception is raised if one or more of the following apply:

- There are one or more open database transactions in `workspace`.
- The procedure was called by a user that does not have the `WM_ADMIN` system privilege or that did not execute the [BeginResolve](#) procedure on `workspace`.

## Examples

The following example ends the conflict resolution session in `Workspace1` and saves all changes.

```
EXECUTE DBMS_WM.CommitResolve ('Workspace1');
```

# 4.14 CompressWorkspace

Deletes removable savepoints in a workspace and minimizes the Workspace Manager metadata structures for the workspace. (*Removable savepoints* are explained in [Using Savepoints](#).)

## Syntax

```
DBMS_WM.CompressWorkspace (
  workspace           IN VARCHAR2,
  compress_view_wo_overwrite IN BOOLEAN,
  firstSP            IN VARCHAR2 DEFAULT NULL,
  secondSP           IN VARCHAR2 DEFAULT NULL,
  auto_commit        IN BOOLEAN DEFAULT TRUE,
  commit_in_batches  IN BOOLEAN DEFAULT FALSE,
  batch_size         IN VARCHAR2 DEFAULT 'PRIMARY_KEY_RANGE',
  remove_latest_deleted_rows IN BOOLEAN DEFAULT FALSE);
```

or

```
DBMS_WM.CompressWorkspace (
  workspace           IN VARCHAR2,
  firstSP            IN VARCHAR2 DEFAULT NULL,
  secondSP           IN VARCHAR2 DEFAULT NULL,
```

```

auto_commit          IN BOOLEAN DEFAULT TRUE,
commit_in_batches   IN BOOLEAN DEFAULT FALSE,
batch_size          IN VARCHAR2 DEFAULT 'PRIMARY_KEY_RANGE',
remove_latest_deleted_rows IN BOOLEAN DEFAULT FALSE);

```

## Parameters

**Table 4-14 CompressWorkspace Procedure Parameters**

Parameter	Description
workspace	Name of the workspace. The name is case-sensitive.
compress_view_wo_overwrite	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE causes history information between the affected savepoints to be deleted even if VIEW_WO_OVERWRITE was specified when versioning was enabled.</p> <p>FALSE causes history information (between the affected savepoints) for a table not to be deleted if VIEW_WO_OVERWRITE was specified when versioning was enabled. (If VIEW_WO_OVERWRITE was not specified for a table, history information for the table is deleted regardless of the parameter value.) FALSE is assumed if the procedure format without this parameter is used.</p>
firstSP	<p>First savepoint. Savepoint names are case-sensitive.</p> <p>If only workspace and firstSP are specified, all removable savepoints between workspace creation and firstSP (but not including firstSP) are deleted.</p> <p>If workspace, firstSP, and secondSP are specified, all removable savepoints from firstSP (and including firstSP if it is a removable savepoint) to secondSP (but not including secondSP) are deleted.</p> <p>If only workspace is specified (no savepoints), all removable savepoints in the workspace are deleted.</p>
secondSP	<p>Second savepoint. All removable savepoints from firstSP (and including firstSP if it is a removable savepoint) to secondSP (but not including secondSP) are deleted.</p> <p>However, if secondSP is LATEST, all removable savepoints from firstSP (and including firstSP if it is a removable savepoint) to the end of the workspace are deleted.</p> <p>Savepoint names are case-sensitive.</p>
auto_commit	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes.</p> <p>FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a>.</p>

**Table 4-14 (Cont.) CompressWorkspace Procedure Parameters**

Parameter	Description
<code>commit_in_batches</code>	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE causes an internal commit operation to be performed after compression operations on <code>batch_size</code> rows in version-enabled tables. Periodic commit operations can be useful or necessary if version-enabled tables have many rows affected by the compression, which can cause substantial Oracle database resources (such as rollback segments and undo tablespaces) to be used. If you specify TRUE, the <code>auto_commit</code> value must also be TRUE.</p> <p>FALSE (the default) causes internal commit operations not to be performed during the compression operation.</p>
<code>batch_size</code>	<p>Batch size for internal commit operations if <code>commit_in_batches</code> is TRUE; otherwise, the parameter is ignored. If specified, must be TABLE or PRIMARY_KEY_RANGE.</p> <p>TABLE causes an internal commit operation to be performed after compressing each version-enabled table that needs to be compressed.</p> <p>PRIMARY_KEY_RANGE specifies that each table is divided into batches of different ranges of primary key values, and an internal commit operation is to be performed after compressing each batch of rows in each version-enabled table that needs to be compressed. You must previously have generated statistics on the first column of the primary key, such as by using the <code>DBMS_STATS.GATHER_TABLE_STATS</code> procedure on the <code>&lt;table_name&gt;_LT</code> table associated with each affected version-enabled table. See the Usage Notes for more information. The following example generates histogram statistics:</p> <pre>EXECUTE DBMS_STATS.GATHER_TABLE_STATS('', 'cola_marketing_budget_lt', estimate_percent=&gt;50, method_opt=&gt;'FOR COLUMNS SIZE 50 product_id');</pre>
<code>remove_latest_deleted_rows</code>	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE causes any LATEST row that has been deleted and that will not adversely affect conflict resolution to be removed, if workspace is LIVE. A value of TRUE is ignored for other workspaces.)</p> <p>FALSE (the default) causes any LATEST row that has been deleted to be preserved.</p>

### Usage Notes

You can compress a workspace when the explicit savepoints (all or some of them) in the workspace are no longer needed. The compression operation is useful for the following reasons:

- You can reuse savepoint names after they are deleted. (You cannot create a savepoint that has the same name as an existing savepoint.)

- Less disk storage is used for Workspace Manager structures (fewer table rows, smaller indexes, less Workspace Manager metadata).
- Because of the reduction in disk space usage, runtime performance for Workspace Manager operations is improved.

This procedure deletes implicit savepoints only if they do not have any child dependencies, and the existence of any such non-removable savepoints will not allow the entire range to be compressed as a single unit. However, you can remove or move such savepoints by using the [RemoveWorkspace](#) or [RefreshWorkspace](#) procedure, respectively.

While this procedure is executing, the current workspace is frozen in `NO_ACCESS` mode, as explained in [Freezing and Unfreezing Workspaces](#).

A workspace cannot be compressed if there are any sessions in the workspace (except for the `LIVE` workspace), or if any user has executed a [GotoDate](#) operation or a [GotoSavepoint](#) operation specifying a savepoint in the workspace.

If the procedure format without the `compress_view_wo_overwrite` parameter is used, a value of `FALSE` is assumed for the parameter.

For information about `VIEW_WO_OVERWRITE` and other history options, see the information about the [EnableVersioning](#) procedure.

If you expect to purge a subset of your historical data periodically, such as removing historical data older than one year, plan to create a savepoint at each expected deletion point on the day it occurs. For example, if you plan to purge 2005 historical data when it is a year old, you need to create a savepoint on January 1, 2006. Then, on January 1, 2007 you can call the `CompressWorkspace` procedure, specifying the workspace name and the January 1, 2006 savepoint, to delete all history that occurred before 2006.

To see if a version-enabled table can be compressed in primary key range batches, check the value of the `BATCH_SIZE` column in the [WM\\_COMPRESS\\_BATCH\\_SIZES](#) metadata view, which is described in [WM\\_COMPRESS\\_BATCH\\_SIZES](#).

To specify a `batch_size` value of `PRIMARY_KEY_RANGE`, you must first generate either histogram statistics (for columns of type `NUMBER`, `INTEGER`, `DATE`, `TIMESTAMP`, `CHAR`, or `VARCHAR2`) or general statistics (for columns of type `NUMBER`, `INTEGER`, `DATE`, or `TIMESTAMP`) on the first column of the primary key. The procedure `DBMS_STATS.GATHER_TABLE_STATS` generates general statistics. If general but not histogram statistics are available for columns of type `NUMBER`, `INTEGER`, `DATE`, or `TIMESTAMP`, the Workspace Manager system parameter `NUMBER_OF_COMPRESS_BATCHES` is used to compute the number of batches when `batch_size` is specified as `PRIMARY_KEY_RANGE`. For more information about statistics, see *Oracle Database Performance Tuning Guide*.

If the current version within the specified workspace needs to be compressed, Workspace Manager attempts to acquire a Shared Sub eXclusive lock of the workspace. If the lock is not acquired, no error is raised, but the current version is not compressed. (See [Locks Taken for Workspace Manager Operations](#).)

An exception is raised if `auto_commit` is `TRUE` and an open transaction exists, if the user does not have sufficient privileges on all tables that need to be modified (including, for example, tables modified by triggers), or if the user does not have the privilege to access and merge changes in `workspace`.



To compress a workspace and all its descendant workspaces, use the [CompressWorkspaceTree](#) procedure.

### Examples

The following example compresses NEWWORKSPACE.

```
EXECUTE DBMS_WM.CompressWorkspace ('NEWWORKSPACE');
```

The following example compresses NEWWORKSPACE, deleting all explicit savepoints between the creation of the workspace and the savepoint SP1.

```
EXECUTE DBMS_WM.CompressWorkspace ('NEWWORKSPACE', 'SP1');
```

The following example compresses NEWWORKSPACE, deleting the explicit savepoint SP1 and all explicit savepoints up to but not including SP2.

```
EXECUTE DBMS_WM.CompressWorkspace ('NEWWORKSPACE', 'SP1', 'SP2');
```

The following example compresses B\_focus\_1, accepts the default values for the firstSP and secondSP parameters (that is, deletes all explicit savepoints), and specifies FALSE for the auto\_commit parameter.

```
EXECUTE DBMS_WM.CompressWorkspace ('B_focus_1', auto_commit => FALSE);
```

The following example analyzes the COLA\_MARKETING\_BUDGET\_LT table to generate the necessary histogram statistics for the next statement, and then it compresses B\_focus\_1. The call to the CompressWorkspace procedure accepts the default values for the firstSP, secondSP, and auto\_commit parameters; specifies TRUE for the commit\_in\_batches parameter; and specifies PRIMARY\_KEY\_RANGE for the batch\_size parameter.

```
EXECUTE DBMS_STATS.GATHER_TABLE_STATS(' ', 'cola_marketing_budget_lt',
estimate_percent=>50, method_opt=>'FOR COLUMNS SIZE 50 product_id');
EXECUTE DBMS_WM.CompressWorkspace ('B_focus_1', NULL, NULL, NULL, TRUE,
'PRIMARY_KEY_RANGE');
```

## 4.15 CompressWorkspaceTree

Deletes removable savepoints in a workspace and all its descendant workspaces. (*Removable savepoints* are explained in [Using Savepoints](#).) It also minimizes the Workspace Manager metadata structures for the affected workspaces, and eliminates any redundant data that might arise from the deletion of the savepoints.

### Syntax

```
DBMS_WM.CompressWorkspaceTree(
  workspace                IN VARCHAR2,
  compress_view_wo_overwrite IN BOOLEAN DEFAULT FALSE,
  auto_commit              IN BOOLEAN DEFAULT TRUE,
  commit_in_batches        IN BOOLEAN DEFAULT FALSE,
  batch_size               IN VARCHAR2 DEFAULT 'PRIMARY_KEY_RANGE',
  remove_latest_deleted_rows IN BOOLEAN DEFAULT FALSE);
```

## Parameters

**Table 4-15 CompressWorkspaceTree Procedure Parameters**

Parameter	Description
<code>workspace</code>	Name of the workspace. The name is case-sensitive.
<code>compress_view_wo_overwrite</code>	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE causes history information to be deleted even if <code>VIEW_WO_OVERWRITE</code> was specified when versioning was enabled.</p> <p>FALSE (the default) causes history information for a table not to be deleted if <code>VIEW_WO_OVERWRITE</code> was specified when versioning was enabled. (If <code>VIEW_WO_OVERWRITE</code> was not specified for a table, history information for the table is deleted regardless of the parameter value.)</p>
<code>auto_commit</code>	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes.</p> <p>FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a>.</p>
<code>commit_in_batches</code>	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE causes an internal commit operation to be performed after compression operations on <code>batch_size</code> rows in version-enabled tables. Periodic commit operations can be useful or necessary if version-enabled tables have many rows affected by the compression, which can cause substantial Oracle database resources (such as rollback segments and undo tablespaces) to be used. If you specify TRUE, the <code>auto_commit</code> value must also be TRUE.</p> <p>FALSE (the default) causes internal commit operations not to be performed during the compression operation.</p>

**Table 4-15 (Cont.) CompressWorkspaceTree Procedure Parameters**

Parameter	Description
<code>batch_size</code>	<p>Batch size for internal commit operations if <code>commit_in_batches</code> is TRUE; otherwise, the parameter is ignored. If specified, must be TABLE or PRIMARY_KEY_RANGE.</p> <p>TABLE causes an internal commit operation to be performed after compressing each version-enabled table that needs to be compressed.</p> <p>PRIMARY_KEY_RANGE specifies that each table is divided into batches of different ranges of primary key values, and an internal commit operation is to be performed after compressing each batch of rows in each version-enabled table that needs to be compressed. You must previously have generated statistics on the first column of the primary key, such as by using the DBMS_STATS.GATHER_TABLE_STATS procedure on the <code>&lt;table_name&gt;_LT</code> table associated with each affected version-enabled table. See the Usage Notes for more information. The following example generates histogram statistics:</p> <pre>EXECUTE DBMS_STATS.GATHER_TABLE_STATS('', 'cola_marketing_budget_lt', estimate_percent=&gt;50, method_opt=&gt;'FOR COLUMNS SIZE 50 product_id');</pre>
<code>remove_latest_deleted_rows</code>	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE causes any LATEST row that has been deleted and that will not adversely affect conflict resolution to be removed, if workspace is LIVE. A value of TRUE is ignored for other values of the workspace parameter.)</p> <p>FALSE (the default) causes any LATEST row that has been deleted to be preserved.</p>

### Usage Notes

You can compress a workspace and all its descendant workspaces when the explicit savepoints in the affected workspaces are no longer needed (for example, if you will not need to go to or roll back to any of these savepoints). For example, in the hierarchy shown in [Workspace Hierarchy](#), a CompressWorkspaceTree operation specifying Workspace1 compresses Workspace1, Workspace2, and Workspace3. (For an explanation of database workspace hierarchy, see [Workspace Hierarchy](#).)

The compression operation is useful for the following reasons:

- You can reuse savepoint names after they are deleted. (You cannot create a savepoint that has the same name as an existing savepoint.)
- Runtime performance for Workspace Manager operations is improved.
- Less disk storage is used for Workspace Manager structures.

While this procedure is executing, the current workspace is frozen in NO\_ACCESS mode, as explained in [Freezing and Unfreezing Workspaces](#).

A workspace cannot be compressed if there are any sessions in the workspace (except for the `LIVE` workspace), or if any user has executed a [GotoDate](#) operation or a [GotoSavepoint](#) operation specifying a savepoint in the workspace.

To see if a version-enabled table can be compressed in primary key range batches, check the value of the `BATCH_SIZE` column in the [WM\\_COMPRESS\\_BATCH\\_SIZES](#) metadata view, which is described in [WM\\_COMPRESS\\_BATCH\\_SIZES](#).

To specify a `batch_size` value of `PRIMARY_KEY_RANGE`, you must first generate either histogram statistics (for columns of type `NUMBER`, `INTEGER`, `DATE`, `TIMESTAMP`, `CHAR`, or `VARCHAR2`) or general statistics (for columns of type `NUMBER`, `INTEGER`, `DATE`, or `TIMESTAMP`) on the first column of the primary key. The procedure `DBMS_STATS.GATHER_TABLE_STATS` generates general statistics. If general but not histogram statistics are available for columns of type `NUMBER`, `INTEGER`, `DATE`, or `TIMESTAMP`, the Workspace Manager system parameter `NUMBER_OF_COMPRESS_BATCHES` is used to compute the number of batches when `batch_size` is specified as `PRIMARY_KEY_RANGE`. For more information about statistics, see *Oracle Database Performance Tuning Guide*.

An exception is raised if `auto_commit` is `TRUE` and an open transaction exists, if the user does not have sufficient privileges on all tables that need to be modified (including, for example, tables modified by triggers), or if the user does not have the privilege to access and merge changes in `workspace`.

If the `CompressWorkspaceTree` operation fails in any affected workspace, the entire operation is rolled back, and no workspaces are compressed.

When the current version within any affected workspace is contained in a compressible range with at least one other version, an attempt is made to acquire a Shared Sub eXclusive lock on that workspace. If it fails to be acquired, no error is raised, but the current version is not compressed. (See [Locks Taken for Workspace Manager Operations](#).)

To compress a single workspace (deleting all explicit savepoints or just some of them), use the [CompressWorkspace](#) procedure.

### Examples

The following example compresses `NEWWORKSPACE` and all its descendant workspaces.

```
EXECUTE DBMS_WM.CompressWorkspaceTree ('NEWWORKSPACE');
```

The following example compresses `NEWWORKSPACE` and all its descendant workspaces, accepts the default value for the `compress_view_wo_overwrite` parameter, and specifies `FALSE` for the `auto_commit` parameter.

```
EXECUTE DBMS_WM.CompressWorkspaceTree ('NEWWORKSPACE', auto_commit => FALSE);
```

The following example compresses `NEWWORKSPACE` and all its descendant workspaces; accepts the default value for the `compress_view_wo_overwrite` and `auto_commit` parameters; specifies `TRUE` for the `commit_in_batches` parameter; and specifies `PRIMARY_KEY_RANGE` for the `batch_size` parameter.

```
EXECUTE DBMS_WM.CompressWorkspaceTree ('NEWWORKSPACE', NULL, NULL, TRUE,  
'PRIMARY_KEY_RANGE');
```

## 4.16 CopyForUpdate

Allows LOB columns (BLOB, CLOB, or NCLOB) in version-enabled tables to be modified. Use this procedure only if a version-enabled table has any LOB columns.

### Syntax

```
DBMS_WM.CopyForUpdate(
  table_name      IN VARCHAR2,
  where_clause    IN VARCHAR2 DEFAULT '');
```

### Parameters

**Table 4-16 CopyForUpdate Procedure Parameters**

Parameter	Description
table_name	Name of the table containing one or more LOB columns. The name is not case-sensitive.
where_clause	The WHERE clause (excluding the WHERE keyword) identifying the rows affected. Example: 'department_id = 20'  Only primary key columns can be specified in the WHERE clause, except in a subquery. The subquery can refer to columns that are not primary keys, but it cannot refer to a version-enabled table.  If the where_clause parameter is not specified, all rows in table_name are affected.

### Usage Notes

This procedure is intended for use only with version-enabled tables containing one or more large object (LOB) columns. The CopyForUpdate procedure must be used because updates performed using the DBMS\_LOB package do not fire INSTEAD OF triggers on the versioning views. Workspace Manager creates INSTEAD OF triggers on the versioning views to implement the copy-on-write semantics. (For non-LOB columns, you can directly perform the update operation, and the triggers work.)

### Examples

The following example updates the SOURCE\_CLOB column of TABLE1 for the document with DOC\_ID = 1.

```
Declare
  clob_var
Begin
  /* This procedure copies the LOB columns if necessary, that is,
   if the row with doc_id = 1 has not been versioned in the
   current version */
  dbms_wm.copyForUpdate('table1', 'doc_id = 1');

  select source_clob into clob_var
  from   table1
  where  doc_id = 1 for update;

  dbms_lob.write(clob_var,<amount>, <offset>, buff);
```

```
End;
```

## 4.17 CopyWorkspace

Copies all of the rows modified in a specified workspace into a target workspace.

### Syntax

```
DBMS_WM.CopyWorkspace(  
    source_workspace IN VARCHAR2,  
    target_workspace IN VARCHAR2);
```

### Parameters

**Table 4-17 CopyWorkspace Procedure Parameters**

Parameter	Description
source_workspace	Name of the source workspace. The name is case-sensitive.
target_workspace	Name of the target workspace. The name is case-sensitive.

### Usage Notes

This procedure copies all of the rows that have been modified in the source workspace into the target workspace. Only tables that do not use the `validtime` option will be copied. There is no conflict checking done between the two workspaces, so any rows that already exist in the target workspace will be overwritten. For each row that exists in the source workspace, the appropriate DML will be performed in the target workspace. For example, if a row was inserted into the source but the row already exists in the target, then the row in the target will be updated with the column values from the source. Or, if a row is updated in the source but the row was deleted in the target, then the row will be inserted into the target.

Once all of the rows are copied into the target workspace, all unique, check, and foreign key constraints will be enforced on the target workspace to ensure the data remains valid. If a constraint violation is found, an error will be raised and the operation will be rolled back.

If a row in the source has been locked in exclusive or shared mode, it will not be possible to copy the row into the target, and an error will be raised. Similarly, if the target workspace has a default lock mode set by previously having executed the [SetWorkspaceLockModeON](#) procedure, then an error will be raised in this case as well, due to the row already having been versioned. As a result, it is recommended not to have locking enabled, other than workspace-exclusive, on either the source or target workspaces. In addition, if the target workspace uses the pessimistic locking setting, then an error will be raised when attempting to copy the rows to the target workspace.

The changes to the target workspace are performed in the currently open transaction, or a new transaction is started if one does not yet exist.

An exception is raised if one or more of the following apply:

- The target and source workspace are the same workspace .
- The target or source workspace parameter is 'LIVE' .
- The target or source workspace parameter is NULL.
- The user does not have ACCESS privileges on both workspaces and all versioned tables that have been modified in the source workspace.

### Examples

The following example copies any rows modified in the `child_1` workspace into `child_2`. Both workspace are child workspace of LIVE.

```
EXECUTE DBMS_WM.CopyWorkspace('child_1', 'child_2');
```

## 4.18 CreateSavepoint

Creates a savepoint for the current version.

### Syntax

```
DBMS_WM.CreateSavepoint(
  workspace      IN VARCHAR2,
  savepoint_name IN VARCHAR2,
  description    IN VARCHAR2 DEFAULT NULL,
  auto_commit    IN BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 4-18 CreateSavepoint Procedure Parameters**

Parameter	Description
<code>workspace</code>	Name of the workspace in which to create the savepoint. The name is case-sensitive.
<code>savepoint_name</code>	Name of the savepoint to be created. The name is case-sensitive.
<code>description</code>	Description of the savepoint to be created.
<code>auto_commit</code>	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a> .

### Usage Notes

There are no explicit privileges associated with savepoints; any user who can access a workspace can create a savepoint in the workspace.

This procedure can be performed while there are users in the workspace. There can be open database transactions, but only if these transactions have not modified a versioned table.

While this procedure is executing, the current workspace is frozen in `READ_ONLY` mode, as explained in [Freezing and Unfreezing Workspaces](#).

An exception is raised if one or more of the following apply:

- The user is not in the latest version in the workspace (for example, if the user has called the [GotoDate](#) procedure).
- `workspace` does not exist.
- `savepoint_name` already exists.
- `auto_commit` is `TRUE` and an open transaction exists in a parent or child workspace of any table that needs to be modified.
- The user does not have the privilege to go to the specified workspace.

### Examples

The following example creates a savepoint named `Savepoint1` in the `NEWWORKSPACE` workspace.

```
EXECUTE DBMS_WM.CreateSavepoint ('NEWWORKSPACE', 'Savepoint1');
```

## 4.19 CreateWorkspace

Creates a new workspace in the database.

### Syntax

```
DBMS_WM.CreateWorkspace(
  workspace      IN VARCHAR2,
  description    IN VARCHAR2 DEFAULT NULL,
  auto_commit    IN BOOLEAN DEFAULT TRUE);
```

or

```
DBMS_WM.CreateWorkspace(
  workspace      IN VARCHAR2,
  isrefreshed    IN BOOLEAN,
  description    IN VARCHAR2 DEFAULT NULL,
  auto_commit    IN BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 4-19 CreateWorkspace Procedure Parameters**

Parameter	Description
<code>workspace</code>	Name of the workspace. The name is case-sensitive, and it must be unique (no other workspace of the same name). The name must not contain any of the following characters: " (double quotes), ' (single quote), ` (grave accent), or   (vertical bar).



**Table 4-19 (Cont.) CreateWorkspace Procedure Parameters**

Parameter	Description
<code>isrefreshed</code>	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE causes the workspace to be continually refreshed. In a <b>continually refreshed workspace</b> (described in <a href="#">Continually Refreshed Workspaces</a>), changes made in the parent workspace are automatically applied to the workspace whenever data changes are committed in the parent workspace or are merged into the parent workspace from another child workspace. That is, you do not need to call the <a href="#">RefreshWorkspace</a> procedure to apply the changes. See the Usage Notes for more information about continually refreshed workspaces.</p> <p>FALSE causes the workspace not to be continually refreshed. To refresh the workspace, you must call the <a href="#">RefreshWorkspace</a> procedure.</p> <p>If you use the syntax without the <code>isrefreshed</code> parameter, the workspace is not continually refreshed.</p>
<code>description</code>	Description of the workspace.
<code>auto_commit</code>	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes.</p> <p>FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a>.</p>

### Usage Notes

The new workspace is a child of the current workspace. If the session has not explicitly entered a workspace, it is in the `LIVE` database workspace, and the new workspace is a child of the `LIVE` workspace. For an explanation of database workspace hierarchy, see [Workspace Hierarchy](#).

The owner of the workspace is the user that executed the `CreateWorkspace` procedure (or another procedure that executed the `CreateWorkspace` procedure), not the user that had the active permissions at the time the workspace was being created.

An implicit savepoint is created in the current version of the current workspace. (The current version does not have to be the latest version in the current workspace.) For an explanation of savepoints (explicit and implicit), see [Using Savepoints](#).

While this procedure is executing, the current workspace is frozen in `READ_ONLY` mode, as explained in [Freezing and Unfreezing Workspaces](#).

This procedure does not implicitly go to the workspace created. To go to the workspace, use the [GotoWorkspace](#) procedure.

The following rules apply to continually refreshed workspaces (`isrefreshed` value of `TRUE`):

- The session must be on the latest version in order to create a continually refreshed workspace.
- You cannot turn off locking using the [SetLockingOFF](#) or [SetWorkspaceLockModeOFF](#) procedure for a continually refreshed workspace.

An exception is raised if one or more of the following apply:

- `workspace` already exists.
- `auto_commit` is `TRUE` and an open transaction exists in a parent or child workspace of any table that needs to be modified.
- The user does not have the privilege to create a workspace.

### Examples

The following example creates a workspace named `NEWWORKSPACE` in the database.

```
EXECUTE DBMS_WM.CreateWorkspace ('NEWWORKSPACE');
```

## 4.20 Delete\_Topo\_Geometry\_Layer

Deletes a topology geometry layer from a topology.

### Format

```
DBMS_WM.Delete_Topo_Geometry_Layer(
    topology      IN VARCHAR2,
    table_name    IN VARCHAR2,
    column_name   IN VARCHAR2);
```

### Parameters

**Table 4-20 Delete\_Topo\_Geometry\_Layer Procedure Parameters**

Parameter	Description
<code>topology</code>	Topology from which to delete the topology geometry layer containing the topology geometries in the specified column. The topology must have been created using the <code>SDO_TOPO.CREATE_TOPOLOGY</code> procedure.
<code>table_name</code>	Name of the topology geometry layer table containing the column specified in <code>column_name</code> .
<code>column_name</code>	Name of the column (of type <code>SDO_TOPO_GEOMETRY</code> ) containing the topology geometries in the topology geometry layer to be deleted from the topology.

### Usage Notes

This procedure has the same format and meaning as the `SDO_TOPO.DELETE_TOPO_GEOMETRY_LAYER` procedure, which is documented in *Oracle Spatial and Graph Topology Data Model and Network Data Model Graph Developer's Guide*. However, you must use `DBMS_WM.Delete_Topo_Geometry_Layer`, and not `SDO_TOPO.DELETE_TOPO_GEOMETRY_LAYER`, to delete a topology geometry layer from a version-enabled feature table from a topology. For information about Workspace Manager support for topologies, see [Spatial and Graph Topology Support](#).

This procedure deletes data associated with the specified topology geometry layer from the edge, node, and face tables (described in *Oracle Spatial and Graph Topology Data Model and Network Data Model Graph Developer's Guide*).

An exception is generated if `topology` or `table_name` is not version-enabled, or if `table_name` is the only feature table in `topology`.

### Examples

The following example deletes the topology geometry layer that is based on the geometries in the `FEATURE` column of the `LAND_PARCELS` table from the topology named `CITY_DATA`.

```
EXECUTE DBMS_WM.Delete_Topo_Geometry_Layer('CITY_DATA', 'LAND_PARCELS', 'FEATURE');
```

## 4.21 DeleteSavepoint

Deletes a savepoint and associated rows in version-enabled tables.

### Syntax

```
DBMS_WM.DeleteSavepoint(
  workspace           IN VARCHAR2,
  savepoint_name     IN VARCHAR2,
  compress_view_wo_overwrite IN BOOLEAN DEFAULT FALSE,
  auto_commit        IN BOOLEAN DEFAULT TRUE,
  commit_in_batches  IN BOOLEAN DEFAULT FALSE,
  batch_size         IN VARCHAR2 DEFAULT 'PRIMARY_KEY_RANGE');
```

### Parameters

**Table 4-21 DeleteSavepoint Procedure Parameters**

Parameter	Description
<code>workspace</code>	Name of the workspace in which the savepoint was created. The name is case-sensitive.
<code>savepoint_name</code>	Name of the savepoint to be deleted. The name is case-sensitive.
<code>compress_view_wo_overwrite</code>	A Boolean value (TRUE or FALSE). TRUE causes history information to be deleted even if <code>VIEW_WO_OVERWRITE</code> was specified when versioning was enabled. FALSE (the default) causes history information for a table not to be deleted if <code>VIEW_WO_OVERWRITE</code> was specified when versioning was enabled. (If <code>VIEW_WO_OVERWRITE</code> was not specified for a table, history information for the table is deleted regardless of the parameter value.)

**Table 4-21 (Cont.) DeleteSavepoint Procedure Parameters**

Parameter	Description
auto_commit	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes.</p> <p>FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a>.</p>
commit_in_batches	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE causes an internal commit operation to be performed after compression operations on <code>batch_size</code> rows in version-enabled tables. Periodic commit operations can be useful or necessary if version-enabled tables have many rows affected by the savepoint deletion, which can cause substantial Oracle database resources (such as rollback segments and undo tablespaces) to be used. If you specify TRUE, the <code>auto_commit</code> value must also be TRUE.</p> <p>FALSE (the default) causes internal commit operations not to be performed during the savepoint deletion operation.</p>
batch_size	<p>Batch size for internal commit operations if <code>commit_in_batches</code> is TRUE; otherwise, the parameter is ignored. If specified, must be TABLE or PRIMARY_KEY_RANGE.</p> <p>TABLE causes an internal commit operation to be performed after compressing each version-enabled table that needs to be compressed.</p> <p>PRIMARY_KEY_RANGE specifies that each table is divided into batches of different ranges of primary key values, and an internal commit operation is to be performed after compressing each batch of rows in each version-enabled table that needs to be compressed. You must previously have generated statistics on the first column of the primary key, such as by using the DBMS_STATS.GATHER_TABLE_STATS procedure on the <code>&lt;table_name&gt;_LT</code> table associated with each affected version-enabled table. See the Usage Notes for more information. The following example generates histogram statistics:</p> <pre>EXECUTE DBMS_STATS.GATHER_TABLE_STATS('', 'cola_marketing_budget_lt', estimate_percent=&gt;50, method_opt=&gt;'FOR COLUMNS SIZE 50 product_id');</pre>

### Usage Notes

You can delete a savepoint when it is no longer needed (for example, you will not need to go to it or roll back to it).

Deleting a savepoint is useful for the following reasons:

- You can reuse a savepoint name after it is deleted. (You cannot create a savepoint that has the same name as an existing savepoint.)
- Runtime performance for Workspace Manager operations is improved.

- Less disk storage is used for Workspace Manager structures.

While this procedure is executing, the current workspace is frozen in `NO_ACCESS` mode, as explained in [Freezing and Unfreezing Workspaces](#).

To delete a savepoint, you must have the `WM_ADMIN` system privilege or be the owner of the workspace or the savepoint.

This procedure cannot be executed if there are any sessions with an open database transaction, or if any user has executed a [GotoDate](#) operation or a [GotoSavepoint](#) operation specifying a savepoint in the workspace.

To specify a `batch_size` value of `PRIMARY_KEY_RANGE`, you must first generate either histogram statistics (for columns of type `NUMBER`, `INTEGER`, `DATE`, `TIMESTAMP`, `CHAR`, or `VARCHAR2`) or general statistics (for columns of type `NUMBER`, `INTEGER`, `DATE`, or `TIMESTAMP`) on the first column of the primary key. The procedure `DBMS_STATS.GATHER_TABLE_STATS` generates general statistics. If general but not histogram statistics are available for columns of type `NUMBER`, `INTEGER`, `DATE`, or `TIMESTAMP`, the Workspace Manager system parameter `NUMBER_OF_COMPRESS_BATCHES` is used to compute the number of batches when `batch_size` is specified as `PRIMARY_KEY_RANGE`. For more information about statistics, see *Oracle Database Performance Tuning Guide*.

An exception is raised if one or more of the following apply:

- One or more sessions are already in `workspace` (unless the workspace is `LIVE`).
- `workspace` does not exist.
- `savepoint_name` does not exist.
- `savepoint_name` is not a removable savepoint. (Removable savepoints are explained in [Using Savepoints](#).)
- `auto_commit` is `TRUE` and an open transaction exists in a parent or child workspace of any table that needs to be modified.
- The user does not have the privilege to go to the specified workspace.

### Examples

The following example deletes a savepoint named `Savepoint1` in the `NEWWORKSPACE` workspace.

```
EXECUTE DBMS_WM.DeleteSavepoint ('NEWWORKSPACE', 'Savepoint1');
```

## 4.22 DisableVersioning

Deletes all support structures that were created to enable the table to support versioned rows.

### Syntax

```
DBMS_WM.DisableVersioning(
  table_name          IN VARCHAR2,
  force               IN BOOLEAN DEFAULT FALSE,
  ignore_last_error  IN BOOLEAN DEFAULT FALSE,
  isTopology          IN BOOLEAN DEFAULT FALSE,
  keepWMValid        IN BOOLEAN DEFAULT TRUE,
  undo_space         IN VARCHAR2 DEFAULT NULL;
```

## Parameters

Table 4-22 DisableVersioning Procedure Parameters

Parameter	Description
table_name	Name of the table or (if <code>isTopology</code> is <code>TRUE</code> ) Oracle Spatial and Graph topology, or a comma-delimited list of names of tables related by multilevel referential integrity constraints. (Multilevel referential integrity constraints are explained in <a href="#">Referential Integrity Support</a> .) Table names are not case-sensitive.
force	A Boolean value ( <code>TRUE</code> or <code>FALSE</code> ).  <code>TRUE</code> forces all data in workspaces other than <code>LIVE</code> to be discarded before versioning is disabled.  <code>FALSE</code> (the default) prevents versioning from being disabled if <code>table_name</code> was modified in any workspace other than <code>LIVE</code> and if the workspace that modified <code>table_name</code> still exists.
ignore_last_error	A Boolean value ( <code>TRUE</code> or <code>FALSE</code> ).  <code>TRUE</code> ignores the last error, if any, that occurred during the previous call to the <code>DisableVersioning</code> procedure. Information about the last error is stored in the <a href="#">USER_WM_VT_ERRORS</a> and <a href="#">ALL_WM_VT_ERRORS</a> static data dictionary views, which are described in <a href="#">Workspace Manager Static Data Dictionary Views</a> . See the Usage Notes for more information.  <code>FALSE</code> (the default) does not ignore the last error, if any, that occurred during the previous call to the <code>DisableVersioning</code> procedure.
isTopology	A Boolean value ( <code>TRUE</code> or <code>FALSE</code> ).  <code>TRUE</code> indicates that the value specified for the <code>table_name</code> parameter is the name of an Oracle Spatial and Graph topology (not a database table name), as explained in <a href="#">Spatial and Graph Topology Support</a> .  <code>FALSE</code> (the default) indicates that the value specified for the <code>table_name</code> parameter is not an Oracle Spatial and Graph topology name.
keepWMValid	A Boolean value ( <code>TRUE</code> or <code>FALSE</code> ). Applies only if valid time support (described in <a href="#">Workspace Manager Valid Time Support</a> ) has been enabled for the table.  <code>TRUE</code> (the default) causes the <code>WM_VALID</code> column and all data in that column to be kept in the table after the procedure completes.  <code>FALSE</code> causes the <code>WM_VALID</code> column to be dropped and all data in that column deleted as a result of the procedure. Only the current row for each primary key value is kept.
undo_space	The string <code>UNLIMITED</code> (for no specified limit) or a number representing the maximum number of bytes for undo space available for the version-enable operation. Example: <code>'1048576'</code> for 1 megabyte. Any value specified overrides the value of the <code>UNDO_SPACE</code> Workspace Manager system parameter (described in <a href="#">System Parameters for Workspace Manager</a> ).

## Usage Notes

This procedure is used to reverse the effect of the [EnableVersioning](#) procedure. It deletes the Workspace Manager infrastructure (support structures) for versioning of

rows, but does not affect any user data in the `LIVE` workspace. The workspace hierarchy and any savepoints still exist, but all rows are the same as in the `LIVE` workspace. (If there are multiple versions in the `LIVE` workspace of a row in the table for which versioning is disabled, only the most recent version of the row is kept.)

If `table_name` has valid time support (described in [Workspace Manager Valid Time Support](#)), this procedure deletes the `WM_VALID` column and all data in that column. If deleting the `WM_VALID` column would cause a primary key constraint violation, only the row valid at the current time is retained.

If a call to the `DisableVersioning` procedure fails, the table is left in an inconsistent state. If this occurs, you should try to fix the cause of the error (examine the [USER\\_WM\\_VT\\_ERRORS](#) and [ALL\\_WM\\_VT\\_ERRORS](#) static data dictionary views to see the SQL statement and error message), and then call the `DisableVersioning` procedure again with the default `ignore_last_error` parameter value of `FALSE`. However, if the call still fails and you cannot fix the cause of the error, and if you are sure that it is safe and appropriate to ignore this error, then you have the option to ignore the error by calling the `DisableVersioning` procedure with the `ignore_last_error` parameter value of `TRUE`. Note that you are responsible for ensuring that it is safe and appropriate to ignore the error.

Some causes for the failure of the `DisableVersioning` procedure include the following:

- The table contains much data in workspaces and the size of the undo tablespace required for the `DisableVersioning` procedure is not sufficient.
- A compilation error occurred while transferring user-defined triggers from the version-enabled table to the version-disabled table.

The `DisableVersioning` operation fails if the `force` value is `FALSE` and any of the following apply:

- The table is being modified by any user in any workspace other than the `LIVE` workspace.
- There are versioned rows of the table in any workspace other than the `LIVE` workspace.

Only the owner of a table or a user with the `WM_ADMIN` system privilege can disable versioning on the table.

Tables that are version-enabled and users that own version-enabled tables cannot be deleted. You must first disable versioning on the relevant table or tables.

An exception is raised if the table is not version-enabled.

For information about Workspace Manager support for tables in an Oracle Spatial and Graph topology, see [Spatial and Graph Topology Support](#).

## Examples

The following example disables the `EMPLOYEE` table for versioning.

```
EXECUTE DBMS_WM.DisableVersioning ('employee');
```

The following example disables the `EMPLOYEE` table for versioning and ignores the last error that occurred during the previous call to the `DisableVersioning` procedure.

```
EXECUTE DBMS_WM.DisableVersioning ('employee', ignore_last_error => true);
```

The following example disables the `EMPLOYEE`, `DEPARTMENT`, and `LOCATION` tables (which have multilevel referential integrity constraints) for versioning.

```
EXECUTE DBMS_WM.DisableVersioning ('employee, department, location');
```

## 4.23 EnableVersioning

Version-enables a table, creating the necessary structures to enable the table to support multiple versions of rows.

### Syntax

```
DBMS_WM.EnableVersioning(
    table_name      IN VARCHAR2,
    hist            IN VARCHAR2 DEFAULT 'NONE',
    isTopology      IN BOOLEAN DEFAULT FALSE,
    validTime       IN BOOLEAN DEFAULT FALSE,
    undo_space      IN VARCHAR2 DEFAULT NULL,
    validTimeRange IN WM_PERIOD DEFAULT NULL);
```

### Parameters

**Table 4-23 EnableVersioning Procedure Parameters**

Parameter	Description
table_name	Name of the table or (if <code>isTopology</code> is <code>TRUE</code> ) Oracle Spatial and Graph topology, or a comma-delimited list of names of tables related by multilevel referential integrity constraints. (Multilevel referential integrity constraints are explained in <a href="#">Referential Integrity Support</a> .) The length of a table name must not exceed 25 characters. The table must not contain any columns with names that start with <code>WM_</code> or <code>WM\$</code> . The table name and any column names must not contain any characters that need to be quoted, such as (but not restricted to) <code>!</code> , <code>?</code> , or <code>*</code> . The table name is not case-sensitive.
hist	History option, for tracking modifications to <code>table_name</code> . Must be one of the following values:  <b>NONE:</b> The timestamps for modifications to the table are not tracked. (This is the default.) A view named <code>&lt;table_name&gt;_HIST</code> (described in <a href="#">xxx_HIST Views</a> ) is created to contain limited history information, but it will show only the most recent modifications to the same version of the table, and it will not contain the <code>WM_CREATETIME</code> and <code>WM_RETIRETIME</code> columns. A history of modifications to the version is not maintained; that is, subsequent changes to a row in the same version overwrite earlier changes.  <b>VIEW_W_OVERWRITE:</b> The <i>with</i> overwrite ( <code>W_OVERWRITE</code> ) option. A view named <code>&lt;table_name&gt;_HIST</code> (described in <a href="#">xxx_HIST Views</a> ) is created to contain history information, but it will show only the most recent modifications to the same version of the table. A history of modifications to the version is not maintained; that is, subsequent changes to a row in the same version overwrite earlier changes.  <b>VIEW_WO_OVERWRITE:</b> The <i>without</i> overwrite ( <code>WO_OVERWRITE</code> ) option. A view named <code>&lt;table_name&gt;_HIST</code> (described in <a href="#">xxx_HIST Views</a> ) is created to contain history information, and it will show all modifications to the same version of the table. A history of modifications to the version is maintained; that is, subsequent changes to a row in the same version do not overwrite earlier changes.



**Table 4-23 (Cont.) EnableVersioning Procedure Parameters**

Parameter	Description
<code>isTopology</code>	A Boolean value (TRUE or FALSE). TRUE indicates that the value specified for the <code>table_name</code> parameter is the name of an Oracle Spatial and Graph topology (not a database table name), as explained in <a href="#">Spatial and Graph Topology Support</a> . FALSE (the default) indicates that the value specified for the <code>table_name</code> parameter is not an Oracle Spatial and Graph topology name.
<code>validTime</code>	A Boolean value (TRUE or FALSE). TRUE causes valid time support to be included. Workspace Manager valid time support is explained in <a href="#">Workspace Manager Valid Time Support</a> . FALSE (the default) causes valid time support not to be included.
<code>undo_space</code>	A string containing UNLIMITED (for no specified limit) or a number representing the maximum number of bytes for undo space available for the version-enable operation. Example: '1048576' for 1 megabyte. Any value specified overrides the value of the UNDO_SPACE Workspace Manager system parameter (described in <a href="#">System Parameters for Workspace Manager</a> ).
<code>validTimeRange</code>	An object of type WM_PERIOD (explained in <a href="#">WM_PERIOD Data Type</a> ) that specifies the initial valid time range for the WM_VALID column. If you specify a value, you must also specify the <code>validTime</code> parameter value as TRUE. By default, if valid time support is included, the valid time range is from the current system time and until changed.

### Usage Notes

The table that is being version-enabled must have a primary key defined. The primary key can be a composite (multicolumn) primary key.

Only the owner of a table or a user with the `WM_ADMIN` system privilege can enable versioning on the table.

Tables that are version-enabled and users that own version-enabled tables cannot be deleted. You must first disable versioning on the relevant table or tables.

Tables owned by `SYS` cannot be version-enabled, and version-enabled tables cannot have any associated indexes or triggers owned by `SYS`.

A table that has a redaction policy defined on it cannot be version-enabled.

A table that has an invisible column cannot be version-enabled.

A table that has a deferrable constraint cannot be version-enabled.

A table that has a “generated by default” identity column cannot be version-enabled. However, an identity column defined as “default by default on null” is allowed.

If the table has an identity column, the “by default on null” and “by default” options have the same behavior: whenever the value for the column in an INSERT statement evaluates to null, the sequence will be accessed to get the next value. (To change this behavior, use the [UseDefaultValuesForNulls](#) procedure.)

An exception is raised if one or more of the following apply:

- `table_name` is already version-enabled.
- `table_name` contains a list of tables and any of the tables has a referential integrity constraint with a table that is not in the list.
- `table_name` contains any columns whose names start with `WM_` or `WM$`.
- `table_name` or the name of any related object of the table (including columns, indexes, and triggers) contains any quoted identifiers.

If the table is version-enabled with the `VIEW_WO_OVERWRITE hist` option specified, this option can later be disabled and re-enabled by calling the [SetWoOverwriteOFF](#) and [SetWoOverwriteON](#) procedures.

The history option enables you to log and audit modifications.

The history option affects the behavior of the [GotoDate](#) procedure. See the Usage Notes for that procedure.

If you expect to purge a subset of your historical data periodically, such as removing historical data older than one year, plan to create a savepoint at each expected deletion point on the day it occurs. For example, if you plan to purge 2005 historical data when it is a year old, you need to create a savepoint on January 1, 2006. Then, on January 1, 2007 you can call the [CompressWorkspace](#) procedure, specifying the workspace name and the January 1, 2006 savepoint, to delete all history that occurred before 2006

For information about Workspace Manager support for tables in an Oracle Spatial and Graph topology, see [Spatial and Graph Topology Support](#).

Current notes and restrictions include the following:

- If you have referential integrity constraints on version-enabled tables, note the considerations and restrictions in [Referential Integrity Support](#).
- If you have triggers defined on version-enabled tables, note the considerations and restrictions in [Triggers on Version-Enabled Tables](#).
- Constraints and privileges defined on the table are carried over to the version-enabled table.
- DDL operations on version-enabled tables are subject to the procedures and restrictions described in [DDL Operations Related to Version-Enabled Tables](#).
- Index-organized tables cannot be version-enabled.
- Object tables cannot be version-enabled.
- A table with one or more columns of LONG data type cannot be version-enabled.
- A table with one or more nested table columns cannot be version-enabled unless the `ALLOW_NESTED_TABLE_COLUMNS` Workspace Manager system parameter is set to ON.
- A table that has a redaction policy defined on it cannot be version-enabled.

## Examples

The following example enables versioning on the `EMPLOYEE` table.

```
EXECUTE DBMS_WM.EnableVersioning('employee');
```

The following example enables versioning on the `EMPLOYEE`, `DEPARTMENT`, and `LOCATION` tables, which have multilevel referential integrity constraints.

```
EXECUTE DBMS_WM.EnableVersioning('employee,department,location');
```

## 4.24 Export

Exports data from a version-enabled table (all rows, or as limited by any combination of several parameters) to a staging table.

### Syntax

```
DBMS_WM.Export(
  table_name           IN VARCHAR2,
  staging_table        IN VARCHAR2,
  workspace            IN VARCHAR2,
  where_clause         IN VARCHAR2 DEFAULT NULL,
  export_scope         IN VARCHAR2 DEFAULT DBMS_WM.EXPORT_MODIFIED_DATA_ONLY,
  after_savepoint_name IN VARCHAR2 DEFAULT NULL,
  as_of_savepoint_name IN VARCHAR2 DEFAULT NULL,
  after_instant        IN DATE DEFAULT NULL,
  as_of_instant        IN DATE DEFAULT NULL,
  versioned_db         IN BOOLEAN DEFAULT TRUE,
  overwrite_existing_data IN BOOLEAN DEFAULT FALSE,
  auto_commit          IN BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 4-24 Export Procedure Parameters**

Parameter	Description
<code>table_name</code>	Name of the table containing the data to be exported. The name is not case-sensitive.
<code>staging_table</code>	Name of the table to hold the exported data. Must not exceed 25 characters. The name is not case-sensitive. If the table does not exist, a new table with this name is created, with a structure suitable for Workspace Manager export and import operations. (See the Usage Notes for more information about the staging table.)
<code>workspace</code>	Name of the workspace from which to export the data. The name is case-sensitive.
<code>where_clause</code>	The <code>WHERE</code> clause (excluding the <code>WHERE</code> keyword) identifying the rows to be exported. Example: <code>'department_id = 20'</code> Only primary key columns can be specified in the <code>WHERE</code> clause, except in a subquery. The subquery can refer to columns that are not primary keys, but it cannot refer to a version-enabled table. If the <code>where_clause</code> parameter is not specified, all rows in <code>table_name</code> are exported.
<code>export_scope</code>	The scope (amount of data) for the export operation. <code>DBMS_WM.EXPORT_ALL_DATA</code> exports all relevant data in workspace. <code>DBMS_WM.EXPORT_MODIFIED_DATA_ONLY</code> (the default) exports only relevant data that was inserted, updated, or deleted in workspace.

**Table 4-24 (Cont.) Export Procedure Parameters**

Parameter	Description
<code>after_savepoint_name</code>	<p>Name of a savepoint: only data inserted, updated, or deleted after this savepoint is exported.</p> <p>If you do not specify <code>after_savepoint_name</code> or <code>as_of_savepoint_name</code>, savepoints are ignored in determining the data to be exported.</p> <p>See the Usage Notes for guidelines relating to the savepoint-related and instant-related parameters.</p>
<code>as_of_savepoint_name</code>	<p>Name of a savepoint: only data in the workspace at the time the savepoint was created is exported.</p> <p>If you do not specify <code>after_savepoint_name</code> or <code>as_of_savepoint_name</code>, savepoints are ignored in determining the data to be exported.</p> <p>See the Usage Notes for guidelines relating to the savepoint-related and instant-related parameters.</p>
<code>after_instant</code>	<p>Date/time specification: only data inserted, updated, or deleted after this time is exported.</p> <p>If you do not specify <code>after_instant</code> or <code>as_of_instant</code>, time is ignored in determining the data to be exported.</p> <p>See the Usage Notes for guidelines relating to the savepoint-related and instant-related parameters.</p>
<code>as_of_instant</code>	<p>Date/time specification: only data that was in the workspace at this time is exported.</p> <p>If you do not specify <code>after_instant</code> or <code>as_of_instant</code>, time is ignored in determining the data to be exported.</p> <p>See the Usage Notes for guidelines relating to the savepoint-related and instant-related parameters.</p>
<code>versioned_db</code>	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE (the default) creates a staging table that contains versioning information.</p> <p>FALSE creates a staging table that contains only user-defined columns and user-visible data.</p>
<code>overwrite_existing_data</code>	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE overwrites existing data in the staging table with the data that is exported.</p> <p>FALSE (the default) preserves all existing data in the staging table, and raises an exception if the exported data conflicts with the existing data.</p>
<code>auto_commit</code>	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes.</p> <p>FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a>.</p>

## Usage Notes

All data that satisfies the `where_clause` in the version-enabled table `table_name`, the `export_scope` parameter, and any parameters relating to a time or a savepoint in `workspace` is exported to the staging table (`staging_table` parameter).

Each row of data to be exported is considered to be one of the following: inserted, updated, or deleted in `workspace` (that is, *modified data*); or data that was not modified in `workspace` but can be seen in it (that is, *ancestor data*). If data is exported from the `LIVE` workspace, it is all modified data. If a workspace is created and no data has yet been versioned in it, and the Export procedure is called, all the data is ancestor data.

The first time you export data from a version-enabled table, the staging table should not exist; that is, do not try to create a staging table, but let the procedure create one for you using the name specified for the `staging_table` parameter. The staging table will contain all columns in the original table (`table_name` parameter), plus some columns for use by Workspace Manager.

After the staging table is created, you can use it for subsequent export operations from the original table, as long as you have not done any of the following DDL operations on the original table: altered any column names or data types, or modified or deleted the primary key constraint. If you have made any of these alterations to the original table, drop the staging table before you call the Export procedure, so that Workspace Manager can create a new staging table. (If you want to overwrite data in an existing staging table, you must also specify `overwrite_existing_data` as `TRUE`.)

The staging table must be in the current user's schema; or if it is in another schema, the current user must have the `CREATE ANY TABLE` and `INSERT ANY TABLE` privileges.

It is recommended that you specify no more than one of the following savepoint-related and instant-related parameters: `after_savepoint_name`, `as_of_savepoint_name`, `after_instant`, `as_of_instant`. If you specify `after_savepoint_name` and `after_instant`, the interaction of the two parameters can have complex results. You cannot specify the following parameter combinations: `after_savepoint_name` and `as_of_savepoint_name`, `after_instant` and `as_of_instant`, or `as_of_savepoint_name` and `as_of_instant`.

If `versioned_db` is `TRUE`, the staging table has three metadata columns in addition to all user-defined columns. The three added metadata columns are `WM$DELETEDROW`, and two invisible columns (`WM$FLAG` and `WM$WORKSPACE`) that are used internally. You can use the `WM$DELETEDROW` column to determine if the row was in a deleted form in the source workspace.

An exception is raised if one or more of the following apply:

- A specified table, workspace, or savepoint does not exist.
- `table_name` contains a nested table column.
- `table_name` contains a column named `WM_VALID` of type `WM_PERIOD`. (That is, this procedure is not supported for tables with valid time support, which is explained in [Workspace Manager Valid Time Support](#).)
- `staging_table` exists but is not in a valid format for the export operation.
- `staging_table` is not in the current user's schema and the current user does not have the `CREATE TABLE` and `INSERT TABLE` privileges.

- The user does not have the `ACCESS_WORKSPACE` privilege for `workspace` or the `ACCESS_ANY_WORKSPACE` privilege.
- `overwrite_existing_data` is `FALSE` and data that needs to be exported already exists in `staging_table`.
- `auto_commit` is `TRUE` and an open transaction exists in a parent or child workspace of any table that needs to be modified.

See also [Import and Export Considerations](#).

### Examples

The following example exports all data from the `COLA_MARKETING_BUDGET` table in workspace `B_Focus_2` into the staging table `COLA_MARKETING_BUDGET_STG`. (The `EXECUTE` statement is actually on a single line.)

```
EXECUTE DBMS_WM.Export(table_name => 'COLA_MARKETING_BUDGET', staging_table =>
'COLA_MARKETING_BUDGET_STG', workspace => 'B_focus_2');
```

## 4.25 Export\_Schemas

Creates a dump file containing everything related to Workspace Manager. Uses the Oracle Data Pump Export utility.

### Syntax

```
DBMS_WM.Export_Schemas(
  job_name          IN VARCHAR2,
  alt_schema        IN VARCHAR2 DEFAULT 'WMSYS_N',
  ignore_last_error IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 4-25 Export\_Schemas Procedure Parameters**

Parameter	Description
<code>job_name</code>	Name of the Data Pump job to be used for the export operation.
<code>alt_schema</code>	Name of the database schema to be used for temporary storage of Workspace Manager metadata. The default is <code>WMSYS_N</code> . This schema must not already exist when the procedure is called, and it is automatically dropped when the procedure completes successfully.
<code>ignore_last_error</code>	A Boolean value ( <code>TRUE</code> or <code>FALSE</code> ).  <code>TRUE</code> ignores the last error, if any, that occurred during the previous call to the <code>Export_Schemas</code> procedure. Information about the last error is stored in the <code>DBA_WM_VT_ERRORS</code> static data dictionary view, which is described in <a href="#">Workspace Manager Static Data Dictionary Views</a> . See the Usage Notes for more information.  <code>FALSE</code> (the default) does not ignore the last error, if any, that occurred during the previous call to the <code>Export_Schemas</code> procedure.

## Usage Notes

This procedure creates a dump file that contains all of the schemas that contain a version-enabled table or a parent table in a referential integrity constraint of a version-enabled table, as well as any internal Workspace Manager metadata. For any included schema, all objects and data within the schema are included in the dump file, not just the objects related to Workspace Manager. All other schemas are excluded.

This procedure makes use of an already existing Data Pump Export job. When you create this job using the `DBMS_DATAPUMP.OPEN` procedure, the `operation` parameter must be set to `EXPORT` and the `mode` parameter must be set to `SCHEMA`. The dump file(s) and log file should also be specified before you call `DBMS_WM.Export_Schemas`. No procedures that modify or limit what gets exported (such as `DBMS_DATAPUMP.METADATA_FILTER`) should be executed on this job. The Data Pump job should not be created while using `SYSDBA` privileges.

Because the `WMSYS` schema cannot be exported by the Oracle Data Pump Export utility, a temporary schema is required to hold some of the required data. This schema, specified by the `alt_schema` parameter, cannot exist before you call this procedure. Because this schema will be included within the generated dump file, it should be a schema that does *not* exist on the target database.

For information about using the Data Pump Utility, see *Oracle Database Utilities*.

If a call to the `Export_Schemas` procedure fails, you should try to fix the cause of the error. Examine the `DBA_WM_VT_ERRORS` static data dictionary view where the `STATE` column is equal to `EXPORT` to see the SQL statement and error message. Fix the cause of the error, and then call the `Export_Schemas` procedure again with the default `ignore_last_error` parameter value of `FALSE`. However, if the call still fails and you cannot fix the cause of the error, and if you are sure that it is safe and appropriate to ignore this error, then you can ignore the error by calling the `Export_Schemas` procedure with the `ignore_last_error` parameter value of `TRUE`. Note that you are responsible for ensuring that it is safe and appropriate to ignore the error.

If the `Export_Schemas` procedure fails and if you must execute the procedure again, and if a row for this procedure exists in the `DBA_WM_VT_ERRORS` static data dictionary view, the procedure will continue to use the original Data Pump job that was specified (you do not need to create a new job). However, if the SQL statements being executed attempt to use the original job but that job no longer exists, you must set the `ignore_last_error` parameter to `TRUE` and execute the `Export_Schemas` procedure; and after that succeeds, execute the `Export_Schemas` procedure again.

Before exporting a version-enabled topology using either a full database export or the `DBMS_WM.Export_Schemas` procedure, you must do the following:

1. Connect to the database as the owner of the topology.
2. Execute the `SDO_TOPO.PREPARE_FOR_EXPORT` procedure, to create the topology export information table, with a name in the format `<topology-name>_EXP$`. This table contains the same columns as the `USER_SDO_TOPO_INFO` and `ALL_SDO_TOPO_INFO` views.

An exception is raised if one or more of the following apply:

- `job_name` does not exist.
- `alt_schema` already exists.
- The executing user does not have the `DATAPUMP_EXP_FULL_DATABASE` role.

- Errors exist in the `WMSYS` schema or in any required user schemas.

See also [Import and Export Considerations](#).

### Examples

The following example exports the Workspace Manager metadata using the Oracle Data Pump job named `EXPORT_OWM_SCHEMAS`. It assumes that the `DUMP_DIR` directory has already been created.

```
DECLARE
  job_name varchar2(128) := 'EXPORT_OWM_SCHEMAS' ;
  dpj number ;
BEGIN
  dpj := dbms_datapump.open('EXPORT', 'SCHEMA', null, job_name, 'COMPATIBLE') ;
  dbms_datapump.add_file(dpj, 'owm_schema.dmp', 'DUMP_DIR') ;
  dbms_datapump.add_file(dpj, 'owm_schema_export.log', 'DUMP_DIR',
filetype=>dbms_datapump.KU$_FILE_TYPE_LOG_FILE) ;
  dbms_wm.export_schemas(job_name) ;
  dbms_datapump.detach(dpj) ;

exception when others then
  dbms_datapump.detach(dpj) ;
  raise ;
END;
/
```

The following example exports the Workspace Manager metadata to the Oracle Cloud Infrastructure Object Storage using the `dbms_datapump` API job. The example assumes that the Object Storage bucket and the credentials to access the object store are already created. Also, ensure that Oracle Workspace Manager is enabled on your Autonomous Database instance. See [Enable Oracle Workspace Manager on Autonomous Database](#) for more information.

#### Note:

Oracle Workspace Manager is enabled on Autonomous Database Serverless only in selected cloud regions. The following table lists the cloud regions that support Oracle Workspace Manager:

**Table 4-26 Supported Cloud Regions for Oracle Workspace Manager**

Region Name	Region Identifier
Singapore (Singapore)	ap-singapore-1
France South (Marseille)	eu-marseille-1
UAE Central (Abu Dhabi)	me-abudhabi-1
Italy Northwest (Milan)	eu-milan-1
Sweden Central (Stockholm)	eu-stockholm-1
South Africa Central (Johannesburg)	af-johannesburg-1
France Central (Paris)	eu-paris-1



```

DECLARE
    hdl          NUMBER;          -- Datapump handle
    dumpFile     VARCHAR2(1024)  := 'https://objectstorage.<region>.oraclecloud.com/n/
<namespace-string>/b/<bucket_name>/o/';
    dumpType     NUMBER          := dbms_datapump.ku$_file_type_uridump_file;
    credName     VARCHAR2(1024)  := 'CRED_NAME';
    logFile      VARCHAR2(1024)  := 'export_test.log';
    logDir       VARCHAR2(20)    := 'DATA_PUMP_DIR';
    logType      NUMBER          := dbms_datapump.ku$_file_type_log_file;
    jobName      varchar2(128)    := DBMS_SCHEDULER.GENERATE_JOB_NAME('EXP_SD_ ');
BEGIN
    dumpFile := dumpFile||jobName;
    dbms_output.put_line( jobName);
    hdl := dbms_datapump.open('EXPORT', 'SCHEMA', job_name=>jobName);
    dbms_output.put_line(' LOCAL ' || jobName);
    dbms_datapump.add_file(hdl, dumpFile, credName, '30MB', dumpType, 1);
    dbms_datapump.add_file(hdl, logfile, logdir, null, logType);
    dbms_wm.export_schemas('ADMIN.' || jobName, 'WMSYS_N') ;
EXCEPTION
    WHEN OTHERS THEN
        dbms_output.put_line(SQLERRM);
        dbms_output.put_line( dbms_utility.format_error_backtrace );
END;
/

```

## 4.26 FindRICSet

Finds tables that need to be version-enabled along with a specified table, due to referential integrity constraint relationships.

### Syntax

```

DBMS_WM.FindRICSet(
    table_name    IN VARCHAR2,
    result_table  IN VARCHAR2);

```

### Parameters

**Table 4-27 FindRICSet Procedure Parameters**

Parameter	Description
table_name	Name of the table for which to find all other tables that will need to be version-enabled along with it, because of referential integrity constraint relationships. The name is not case-sensitive.
result_table	Name of the table to hold the results. The name is not case-sensitive. This table must have two columns, TABLE_OWNER and TABLE_NAME, both of type VARCHAR2. If the table does not exist, a new table with this name and the required columns is created.

### Usage Notes

Workspace Manager has several considerations relating to referential integrity constraints, as explained in [Referential Integrity Support](#). Sometimes, before you can version-enable a table, you must version-enable other tables that are in referential integrity constraints affecting the table. The FindRICSet procedure enables you to find all these other tables.

To display the results, use the `SET SERVEROUTPUT ON` statement before calling this procedure.

If the result table is not in the current user's schema, the following requirements apply:

- If the result table does not exist, the current user must have the `CREATE ANY TABLE` privilege.
- If the result table already exists, the current user must have the required privileges to insert into the table.

An exception is raised if one or more of the following apply:

- `table_name` does not exist.
- `result_table` exists but is not in a valid format.
- `result_table` exists and the current user does not have the required privileges to insert into the table.
- `result_table` does not exist, is specified for a schema other than the current user's schema, and the current user does not have the `CREATE ANY TABLE` privilege.

### Examples

The following example creates two tables, `EMPLOYEES` and `DEPARTMENTS`, where `DEPARTMENTS.MANAGER_ID` has a foreign key relationship referencing `EMPLOYEES.EMPLOYEE_ID`. The example then finds all tables that would need to be version-enabled if `EMPLOYEES` and `DEPARTMENTS` were version-enabled.

The results show that if you want to version-enable the `EMPLOYEES` table, you must version-enable both the `EMPLOYEES` and `DEPARTMENTS` tables; but if you want to version-enable the `DEPARTMENTS` table, you do not need to version-enable any other tables.

```
create table employees (employee_id number primary key, employee_name
varchar2(30));
create table departments (dept_id number primary key, manager_id number
references employees(employee_id));
```

```
-- Check RICs; result table does not already exist.
EXECUTE DBMS_WM.FindRICSet('EMPLOYEES', 'EMPLOYEES_RESULTS');
SELECT * FROM employees_results;
```

TABLE_OWNER	TABLE_NAME
WM_DEVELOPER	EMPLOYEES
WM_DEVELOPER	DEPARTMENTS

```
EXECUTE DBMS_WM.FindRICSet('DEPARTMENTS', 'DEPARTMENTS_RESULTS');
SELECT * FROM departments_results;
```

TABLE_OWNER	TABLE_NAME
WM_DEVELOPER	DEPARTMENTS

## 4.27 FreezeWorkspace

Restricts access to a workspace and the ability of users to make changes in the workspace.

## Syntax

```
DBMS_WM.FreezeWorkspace(
    workspace      IN VARCHAR2,
    freezemode     IN VARCHAR2 DEFAULT 'NO_ACCESS',
    freezewriter   IN VARCHAR2 DEFAULT NULL,
    force          IN BOOLEAN DEFAULT FALSE);
```

or

```
DBMS_WM.FreezeWorkspace(
    workspace      IN VARCHAR2,
    session_duration IN BOOLEAN,
    freezemode     IN VARCHAR2 DEFAULT 'NO_ACCESS',
    freezewriter   IN VARCHAR2 DEFAULT NULL,
    force          IN BOOLEAN DEFAULT FALSE);
```

## Parameters

**Table 4-28 FreezeWorkspace Procedure Parameters**

Parameter	Description
workspace	Name of the workspace. The name is case-sensitive.
session_duration	A Boolean value (TRUE or FALSE). TRUE causes the workspace to be unfrozen when the session that called the FreezeWorkspace procedure disconnects from the database. This value is valid for all freeze modes. FALSE causes the workspace not to be unfrozen when the session that called the FreezeWorkspace procedure disconnects from the database.
freezemode	Mode for the frozen workspace. Must be one of the following values: NO_ACCESS: No sessions are allowed in the workspace. (This is the default.) READ_ONLY: Sessions are allowed in the workspace, but no write operations (insert, update, delete) are allowed. 1WRITER: Sessions are allowed in the workspace, but only one user (see the freezewriter parameter) is allowed to perform write operations (insert, update, delete). 1WRITER_SESSION: Sessions are allowed in the workspace, but only the database session (as opposed to the database user) that called the FreezeWorkspace procedure is allowed to perform write operations (insert, update, delete). The workspace is unfrozen after the session that called the FreezeWorkspace procedure disconnects from the database. WM_ONLY: Only Workspace Manager operations are permitted. No sessions can directly modify data values; however, child workspaces can be merged into the workspace, and savepoints can be created in the workspace.
freezewriter	The user that is allowed to make changes in the workspace. Can be specified only if freezemode is 1WRITER. The default is USER (the current user).

**Table 4-28 (Cont.) FreezeWorkspace Procedure Parameters**

Parameter	Description
<code>force</code>	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE forces the workspace to be frozen even if it is already frozen. For example, this value enables you to freeze the workspace with a different <code>freezemode</code> parameter value without having first to call the <a href="#">UnfreezeWorkspace</a> procedure.</p> <p>FALSE (the default) prevents the workspace from being frozen if it is already frozen.</p>

### Usage Notes

If you specify the procedure syntax that does not include the `session_duration` parameter, it is equivalent to specifying `FALSE` for that parameter: that is, the workspace is not unfrozen when the session that called the `FreezeWorkspace` procedure disconnects from the database.

The operation fails if one or more of the following apply:

- `workspace` is already frozen (unless `force` is `TRUE`).
- Any sessions are in `workspace` and `freezemode` is `NO_ACCESS` (specified or defaulted).
- `session_duration` is `FALSE` and `freezemode` is `1WRITER_SESSION`.

If `freezemode` is `READ_ONLY` or `1WRITER`, the workspace cannot be frozen if there is an active database transaction.

You can freeze a workspace only if one or more of the following apply:

- You are the owner of the specified workspace.
- You have the `WM_ADMIN` system privilege, the `FREEZE_ANY_WORKSPACE` privilege, or the `FREEZE_WORKSPACE` privilege for the specified workspace.

The `LIVE` workspace can be frozen only if `freezemode` is `READ_ONLY` or `1WRITER`.

To reverse the effect of `FreezeWorkspace`, use the [UnfreezeWorkspace](#) procedure.

### Examples

The following example freezes the `NEWWORKSPACE` workspace.

```
EXECUTE DBMS_WM.FreezeWorkspace ('NEWWORKSPACE');
```

## 4.28 GetBulkLoadVersion

Returns a version number that can be specified in the call to the [BeginBulkLoading](#) procedure and in the SQL\*Loader control file.

**Note:**

Effective with Oracle Database Release 12.1, this function is not necessary, and it always returns a null value.

**Format**

```
DBMS_WM.GetBulkLoadVersion(
    workspace      IN VARCHAR2,
    savepoint_var  IN VARCHAR2 DEFAULT 'LATEST') RETURN INTEGER;
```

**Parameters****Table 4-29** GetBulkLoadVersion Function Parameters

Parameter	Description
workspace	Name of the workspace for which to return the bulk load version. The name is case-sensitive.
savepoint_var	The version in the workspace in which data will be bulk loaded. Must be one of the following: LATEST or ROOT_VERSION. LATEST (the default) is the current version in the workspace. ROOT_VERSION is into the root version (version number 0, which is in the LIVE workspace). The root version is the ancestor of all other versions, so data in the root version is visible from all other workspaces (unless non-LIVE workspaces have updated the data). You can specify ROOT_VERSION only if workspace is LIVE.

**Usage**

Effective with Oracle Database Release 12.1, this function is not necessary and it always returns a null value. The [BeginBulkLoading](#) procedure automatically determines the bulk load version based on the workspace name and the optional savepoint name. (However, the bulk loading process in effect for previous releases is still supported.)

Before you can begin bulk loading data into a version-enabled table, you must call the [BeginBulkLoading](#) procedure. You must end the bulk loading session by calling either the [CommitBulkLoading](#) procedure (to commit changes made when the data was loaded) or the [RollbackBulkLoading](#) procedure (to roll back changes made when the data was loaded). For more information about bulk loading with Workspace Manager, see [Bulk Loading into Version-Enabled Tables](#).

An exception is raised if one or more of the following apply:

- workspace does not exist.
- savepoint\_var is not a valid value.
- savepoint\_var is ROOT\_VERSION but workspace is not LIVE.

**Examples**

The following example gets a bulk load version number for the W1 workspace, and starts the bulk load operation into the EMP table in that workspace.

```
DECLARE
  version INTEGER;
BEGIN
  SELECT DBMS_WM.GetBulkLoadVersion ('W1') INTO version FROM DUAL;
  DBMS_WM.BeginBulkLoading ('EMP', 'W1', version);
END;
/
```

## 4.29 GetConflictWorkspace

Returns the name of the workspace on which the session has performed the [SetConflictWorkspace](#) procedure.

### Format

```
DBMS_WM.GetConflictWorkspace() RETURN VARCHAR2;
```

### Parameters

None.

### Usage Notes

If the [SetConflictWorkspace](#) procedure has not been executed, the name of the current workspace is returned.

### Examples

The following example displays the name of the workspace on which the session has performed the [SetConflictWorkspace](#) procedure.

```
SELECT DBMS_WM.GetConflictWorkspace FROM DUAL;
```

```
GETCONFLICTWORKSPACE
```

```
-----  
B_focus_2
```

## 4.30 GetDiffVersions

Returns the names of the (workspace, savepoint) pairs on which the session has performed the [SetDiffVersions](#) operation.

### Format

```
DBMS_WM.GetDiffVersions() RETURN VARCHAR2;
```

### Parameters

None.

### Usage Notes

The returned string is in the format '(WS1, SP1), (WS2, SP2)'. This format, including the parentheses, is intended to help you if you later want to use parts of the returned string in a call to the [SetDiffVersions](#) procedure.

### Examples

The following example displays the names of the (workspace, savepoint) pairs on which the session has performed the [SetDiffVersions](#) operation.

```
SELECT DBMS_WM.GetDiffVersions FROM DUAL;
```

```
GETDIFFVERSIONS
```

```
-----  
(B_focus_1, LATEST), (B_focus_2, LATEST)
```

## 4.31 GetLockMode

Returns the locking mode for the current session, which determines whether or not access is enabled to versioned rows and corresponding rows in the previous version.

### Format

```
DBMS_WM.GetLockMode() RETURN VARCHAR2;
```

### Parameters

None.

### Usage Notes

This function returns E, S, C, or NULL.

- For explanations of E (exclusive), S (shared), and C (carry-forward), see the description of the `lockmode` parameter of the [SetLockingON](#) procedure.
- NULL indicates that locking is not in effect. (Calling the [SetLockingOFF](#) procedure results in this setting.)

For an explanation of Workspace Manager locking, see [Lock Management with Workspace Manager](#). See also the descriptions of the [SetLockingON](#) and [SetLockingOFF](#) procedures.

### Examples

The following example displays the locking mode in effect for the session.

```
SELECT DBMS_WM.GetLockMode FROM DUAL;
```

```
GETLOCKMODE
```

```
-----  
C
```

## 4.32 GetMultiWorkspaces

Returns the names of workspaces visible in the multiworkspace views for version-enabled tables.

### Format

```
DBMS_WM.GetMultiWorkspaces() RETURN VARCHAR2;
```

### Parameters

None.

### Usage Notes

This procedure returns the names of workspaces visible in the multiworkspace views, which are described in [xxx\\_MW Views](#).

If no workspaces are visible in the multiworkspace views, `NULL` is returned. If more than one workspace name is returned, names are separated by a comma (for example: `workspace1,workspace2,workspace3`).

To make a workspace visible in the multiworkspace views, use the [SetMultiWorkspaces](#) procedure.

### Examples

The following example displays the names of workspaces visible in the multiworkspace views.

```
SELECT DBMS_WM.GetMultiWorkspaces FROM DUAL;
```

## 4.33 GetOpContext

Returns the context of the current operation for the current session.

### Format

```
DBMS_WM.GetOpContext() RETURN VARCHAR2;
```

### Parameters

None.

### Usage Notes

This function returns one of the following values:

- **DML**: The current operation is driven by data manipulation language (DML) initiated by the user.
- **IMPORT**: The current operation was initiated by a [Import](#) procedure call.
- **MERGE\_REMOVE**: The current operation was initiated by a [MergeWorkspace](#) procedure call with the `remove_workspace` parameter set to `TRUE` or a [MergeTable](#) procedure call with the `remove_data` parameter set to `TRUE`.
- **MERGE\_NOREMOVE**: The current operation was initiated by a [MergeWorkspace](#) procedure call with the `remove_workspace` parameter set to `FALSE` or a [MergeTable](#) procedure call with the `remove_data` parameter set to `FALSE`.
- **WORKSPACE\_COPY**: The current operation was initiated by a [CopyWorkspace](#) procedure call.

The returned value can be used in user-defined triggers to take appropriate action based on the current operation.



## Examples

The following example displays the context of the current operation.

```
SELECT DBMS_WM.GetOpContext FROM DUAL;
```

```
GETOPCONTEXT
```

```
-----  
DML
```

## 4.34 GetOriginalDDL

Returns the original DDL of the version-enabled table as it existed before the call to the [EnableVersioning](#) procedure.

### Format

```
DBMS_WM.GetOriginalDDL  
    table_id  IN VARCHAR2,  
    ddl_stmts IN OUT KU$_DDL$;
```

or

```
DBMS_WM.GetOriginalDDL  
    table_id  IN VARCHAR2,  
    ddl_clob  IN OUT CLOB;
```

### Parameters

**Table 4-30** GetOriginalDDL Procedure Parameters

Parameter	Description
table_id	Name of the table for which to return the original DDL for creating the table. The name is not case-sensitive.
ddl_stmts	The original DDL statements for creating the table and any indexes, triggers, and grants on the table.  The type <code>KU\$_DDL\$</code> is defined as <code>TABLE OF KU\$_DDL</code> . The type <code>KU\$_DDL</code> is defined as <code>(DDLTEXT CLOB, PARSEDITEMS KU\$_PARSED_ITEMS)</code> . The type <code>KU\$_PARSED_ITEMS</code> is defined as <code>TABLE OF KU\$_PARSED_ITEM</code> . The type <code>KU\$_PARSED_ITEM</code> is defined as <code>(ITEM VARCHAR2(30), VALUE VARCHAR2(4000), OBJECT_ROW NUMBER)</code> .
ddl_clob	(Same information as for <code>ddl_stmts</code> , but using the type <code>CLOB</code> .)

### Usage

When the [EnableVersioning](#) procedure is called, DDL statements are executed on the table that modify its structure and that of related objects. (Some of these changes are outlined in [Infrastructure for Version-Enabling of Tables](#).) The `GetOriginalDDL` procedure returns a series of DDL statements (`CREATE TABLE`, `CREATE INDEX`, `CREATE TRIGGER`, `GRANT`, and so on) that represent the table as if it was not a version-enabled table. These statements can then be used to create the table in a non-versioned form in another schema or in another database. This new table can then be version-enabled or used in its non-versioned form.

An exception is raised if either of the following applies:

- `table_id` does not exist.
- You do not have access to `table_id`.

### Examples

The following example returns the original DDL statements for the `COLA_MARKETING_BUDGET` table into a variable of type `KU$_DLS`.

```
DECLARE
    original_ddl KU$_DLS;
BEGIN
    DBMS_WM.GetOriginalDDL('cola_marketing_budget',
                          original_ddl);
END;
/
```

## 4.35 GetPhysicalTableName

Returns the name (`<table_name>_LT` form) of the physical table for a version-enabled table.

### Format

```
DBMS_WM.GetPhysicalTableName(
    table_owner IN VARCHAR2,
    table_name  IN VARCHAR2) RETURN VARCHAR2;
```

### Parameters

**Table 4-31** GetPhysicalTableName Function Parameters

Parameter	Description
<code>table_owner</code>	Name of the schema that owns <code>table_name</code> .
<code>table_name</code>	Name of the version-enabled table for which to return the name of its associated physical table.

### Usage

If `table_name` is a version-enabled table, this function returns the name of the table, whose name is in the form `<table_name>_LT`, that was created by Workspace Manager when the [EnableVersioning](#) procedure was called. For information about these `<table_name>_LT` tables, see [Infrastructure for Version-Enabling of Tables](#).

If `table_name` is not a version-enabled table, this function returns `table_name`. Thus, you can also use this function to check whether or not a table is version-enabled (that is, by checking whether a name in the form `<table_name>_LT` or the original table name is returned).

If the user executing the function does not have access to the table or the table does not exist, the function returns a null value.

## Examples

The following example displays the physical table name associated with the COLA\_MARKETING\_BUDGET table after that table is version-enabled.

```
SELECT DBMS_WM.GetPhysicalTableName('wm_developer', 'cola_marketing_budget')
       FROM DUAL;

DBMS_WM.GETPHYSICALTABLERNAME('WM_DEVELOPER','COLA_MARKETING_BUDGET')
-----
COLA_MARKETING_BUDGET_LT
```

## 4.36 GetPrivs

Returns a comma-delimited list of all privileges that the current user has for the specified workspace.

### Format

```
DBMS_WM.GetPrivs(
    workspace IN VARCHAR2) RETURN VARCHAR2;
```

### Parameters

**Table 4-32** GetPrivs Function Parameters

Parameter	Description
workspace	Name of the workspace for which to return the list of privileges. The name is case-sensitive.

### Usage

For information about Workspace Manager privileges, see [Privilege Management with Workspace Manager](#).

### Examples

The following example displays the privileges that the current user has for the B\_focus\_2 workspace.

```
SELECT DBMS_WM.GetPrivs ('B_focus_2') FROM DUAL;

DBMS_WM.GETPRIVS('B_FOCUS_2')
-----
ACCESS, MERGE, CREATE, REMOVE, ROLLBACK
```

## 4.37 GetSessionInfo

Retrieves information about the current workspace and session context.

### Format

```
DBMS_WM.GetSessionInfo(
    workspace OUT VARCHAR2,
```

```
context          OUT VARCHAR2,
context_type    OUT VARCHAR2);
```

## Parameters

**Table 4-33 GetSessionInfo Procedure Parameters**

Parameter	Description
workspace	Name of the workspace that the current session is in.
context	The context of the current session in the workspace, expressed as one of the following: <code>LATEST</code> , a savepoint name, or an instant (point in time) in 'DD-MON-YYYY HH24:MI:SS' date format. (See the Usage Notes for details.)
context_type	The type of context for the current session in the workspace. Specifically, one of the following values: <code>LATEST</code> (if context is <code>LATEST</code> ), <code>SAVEPOINT</code> (if context is a savepoint name), or <code>INSTANT</code> (if context is an instant).

## Usage Notes

This procedure is useful if you need to know where a session is (workspace and context) -- for example, after you have performed a combination of [GotoWorkspace](#), [GotoSavepoint](#), and [GotoDate](#) operations.

After the procedure successfully executes, the `context` parameter contains one of the following values:

- `LATEST`: The session is currently on the `LATEST` logical savepoint (explained in [Using Savepoints](#)), and it can see changes as they are made in the workspace. The context is automatically set to `LATEST` when the session enters the workspace (using the [GotoWorkspace](#) procedure).
- A savepoint name: The session is currently on a savepoint in the workspace. The session cannot see changes as they are made in the latest version of the workspace, but instead sees a static view of the data as of the savepoint creation time. The session context is set to the savepoint name after a call to the [GotoSavepoint](#) procedure.
- An instant (a point in time): The session is currently on a specific point in time. The session cannot see changes as they are made in the latest version of the workspace, but instead sees a static view of the data as of the specific time. The session context is set to an instant after a call to the [GotoDate](#) procedure.

For detailed information about the session context, see [Session Context Information for Workspace Manager](#).

## Examples

The following example retrieves and displays information about the current workspace and context in the session.

```
DECLARE
  current_workspace VARCHAR2(128);
  current_context   VARCHAR2(128);
  current_context_type VARCHAR2(128);
BEGIN
```

```

DBMS_WM.GetSessionInfo(current_workspace,
                       current_context,
                       current_context_type);
DBMS_OUTPUT.PUT_LINE('Session currently in workspace: ' ||current_workspace);
DBMS_OUTPUT.PUT_LINE('Session context is: ' ||current_context);
DBMS_OUTPUT.PUT_LINE('Session context is on: ' ||current_context_type);
END;
/
Session currently in workspace: B_focus_2
Session context is: LATEST
Session context is on: LATEST

PL/SQL procedure successfully completed.

```

## 4.38 GetSystemParameter

Returns the value of a Workspace Manager system parameter.

### Syntax

```

DBMS_WM.GetSystemParameter(
    name    IN VARCHAR2) RETURN VARCHAR2;

```

### Parameters

**Table 4-34** GetSystemParameter Procedure Parameters

Parameter	Description
name	Name of the Workspace Manager system parameter for which to set the value. The name must be one of the following: ALLOW_CAPTURE_EVENTS, ALLOW_MULTI_PARENT_WORKSPACES, ALLOW_NESTED_TABLE_COLUMNS, CR_WORKSPACE_MODE, FIRE_TRIGGERS_FOR_NONDML_EVENTS, NONCR_WORKSPACE_MODE.

### Usage Notes

For information about Workspace Manager system parameters, see [System Parameters for Workspace Manager](#).

An exception is raised if the `name` value is not valid.

### Examples

The following checks if multiparent workspaces (described in [Multiparent Workspaces](#)) are allowed.

```

SELECT DBMS_WM.GetSystemParameter ('ALLOW_MULTI_PARENT_WORKSPACES') FROM DUAL;

DBMS_WM.GETSYSTEMPARAMETER('ALLOW_MULTI_PARENT_WORKSPACES')
-----
ON

```

## 4.39 GetValidFrom

Returns the `ValidFrom` attribute of the current session valid time. (Valid time support is described in [Workspace Manager Valid Time Support](#).)

**Format**

```
DBMS_WM.GetValidFrom() RETURN TIMESTAMP WITH TIME ZONE;
```

**Parameters**

None.

**Usage Notes**

To set the session valid time period, use the [SetValidTime](#) procedure.

To get the `ValidTill` attribute of the current session valid time, use the [GetValidTill](#) function.

**Examples**

The following example displays the `ValidFrom` attribute of the current session valid time.

```
SELECT DBMS_WM.GetValidFrom FROM DUAL;
```

```
GETVALIDFROM
```

```
-----  
01-JAN-1995 12:00:00 -04:00
```

## 4.40 GetValidTill

Returns the `ValidTill` attribute of the current session valid time. (Valid time support is described in [Workspace Manager Valid Time Support](#).)

**Format**

```
DBMS_WM.GetValidTill() RETURN TIMESTAMP WITH TIME ZONE;
```

**Parameters**

None.

**Usage Notes**

To set the session valid time period, use the [SetValidTime](#) procedure.

To get the `ValidFrom` attribute of the current session valid time, use the [GetValidFrom](#) function.

**Examples**

The following example displays the `ValidTill` attribute of the current session valid time.

```
SELECT DBMS_WM.GetValidTill FROM DUAL;
```

```
GETVALIDTILL
```

```
-----  
01-JAN-1996 12:00:00 -04:00
```

## 4.41 GetVersion

Returns the current version of Workspace Manager.

### Format

```
DBMS_WM.GetVersion() RETURN VARCHAR2;
```

### Parameters

None.

### Usage Notes

The value returned is the same as that in the [WM\\_INSTALLATION](#) view for the VALUE column where the NAME column value is `OMW_VERSION`.

### Examples

The following example displays the Workspace Manager version number.

```
SELECT DBMS_WM.GetVersion FROM DUAL;
```

```
GETOPCONTEXT
```

```
-----  
12.2.0.1.0
```

## 4.42 GetWMMetadataSpace

Returns the number of bytes currently used to store the Workspace Manager metadata.

### Format

```
DBMS_WM.GetWMMetadataSpace() RETURN NUMBER;
```

### Parameters

None.

### Usage Notes

The Workspace Manager metadata (views, internal tables, and other objects) is by default stored in the default tablespace of the `WMSYS` user. You cannot directly control the size of the Workspace Manager metadata, but you can control its placement by using the [Move\\_Proc](#) procedure to move the metadata to a different tablespace. You can use the `GetWMMetadataSpace` function to determine the approximate minimum space that you will need to have available in the tablespace into which you are considering moving the Workspace Manager metadata.

### Examples

The following example displays the number of bytes currently used to store the Workspace Manager metadata.

```
SELECT DBMS_WM.GetWMMetadataSpace FROM DUAL;
```

```
GETWMMETADATASPACE
```

-----  
6750208

## 4.43 GetWorkspace

Returns the current workspace for the session.

### Format

```
DBMS_WM.GetWorkspace() RETURN VARCHAR2;
```

### Parameters

None.

### Usage Notes

None.

### Examples

The following example displays the current workspace for the session.

```
SELECT DBMS_WM.GetWorkspace FROM DUAL;
```

```
GETWORKSPACE
```

-----  
B\_focus\_2

## 4.44 GotoDate

Goes to a point at or near the specified date and time in the current workspace.

### Syntax

```
DBMS_WM.GotoDate(  
  in_date  IN VARCHAR2,  
  fmt      IN VARCHAR2 DEFAULT 'mmdyyyhh24miss',  
  nlsparam IN VARCHAR2 DEFAULT NULL,  
  tsWtz    IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 4-35 GotoDate Procedure Parameters**

Parameter	Description
in_date	Date and time for the read-only view of the workspace. (See the Usage Notes for details.) If in_date is a VARCHAR2 string, it is a date string or a timestamp with time zone, depending on the value of the tsWtz parameter.
fmt	Date format. The options are the same as for the TO_TIMESTAMP_TZ function, which is described in <i>Oracle Database SQL Language Reference</i> . Default: 'mmdyyyhh24miss'



**Table 4-35 (Cont.) GotoDate Procedure Parameters**

Parameter	Description
nlsparam	Globalization support options. The options are the same as for the <code>TO_TIMESTAMP_TZ</code> function, which is described in <i>Oracle Database SQL Language Reference</i> .
tsWtz	Timestamp with time zone flag. A Boolean value (TRUE or FALSE). TRUE means that <code>in_date</code> is considered a timestamp with time zone information. FALSE (the default) means that <code>in_date</code> is a date string.

### Usage Notes

You are presented a read-only view of the current workspace at or near the specified date and time. The exact time point depends on the history option for tracking changes to data in version-enabled tables, as set by the [EnableVersioning](#) procedure or modified by the [SetWoOverwriteOFF](#) or [SetWoOverwriteON](#) procedure:

- NONE: The read-only view reflects the first savepoint after `in_date`.
- VIEW\_W\_OVERWRITE: The read-only view reflects the data values in effect at `in_date`, except if `in_date` is between two savepoints and data was changed between the two savepoints. In this case, data that had been changed between the savepoints might be seen as empty or as having a previous value. To ensure the most complete and accurate view of the data, specify the `VIEW_WO_OVERWRITE` history option when version-enabling a table.
- VIEW\_WO\_OVERWRITE: The read-only view reflects the data values in effect at `in_date`.

For an explanation of the history options, see the description of the `hist` parameter for the [EnableVersioning](#) procedure.

The following example scenario shows the effect of the `VIEW_WO_OVERWRITE` setting. Assume the following sequence of events:

1. The `MANAGER_NAME` value in a row is Adams.
2. Savepoint `SP1` is created.
3. The `MANAGER_NAME` value is changed to Baxter.
4. The time point that will be specified as `in_date` (in step 7) occurs.
5. The `MANAGER_NAME` value is changed to Chang. (Thus, the value has been changed both before and after `in_date` since the first savepoint and before the second savepoint.)
6. Savepoint `SP2` is created.
7. A [GotoDate](#) operation is executed, specifying the time point in step 4 as `in_date`.

In the preceding scenario:

- If the history option in effect is `VIEW_WO_OVERWRITE`, the `MANAGER_NAME` value after step 7 is Baxter. After step 5, the versioned table has three rows, each with a different `MANAGER_NAME` value (Adams, Baxter, Chang), because each change is made in a new copy of the row.

- If the history option in effect is `VIEW_W_OVERWRITE`, no value is seen after step 7. The updates in steps 3 and 5 are made in the same copy of the row, and the update in step 5 overwrites the update in step 3. As a result, after step 5 the versioned table has two rows, with `MANAGER_NAME` values Adams and Chang. Because the `MANAGER_NAME` value (Baxter) that was in effect at the specified instant has been overwritten, no row is visible.
- If the history option in effect is `NONE`, the `MANAGER_NAME` value after step 7 is Chang, because the first savepoint after the specified instant is `SP2`. After step 5, the versioned table has two rows, with `MANAGER_NAME` values Adams and Chang.

The `GotoDate` procedure should be executed while users exist in the workspace. There are no explicit privileges associated with this procedure.

### Examples

The following example goes to a point at or near midnight at the start of 08-Jun-2004, depending on the history option currently in effect.

```
EXECUTE DBMS_WM.GotoDate ('08-JUN-04', 'DD-MON-YY');
```

## 4.45 GotoSavepoint

Goes to the specified savepoint in the current workspace.

### Syntax

```
DBMS_WM.GotoSavePoint(  
    savepoint_name IN VARCHAR2 DEFAULT 'LATEST');
```

### Parameters

**Table 4-36 GotoSavepoint Procedure Parameters**

Parameter	Description
<code>savepoint_name</code>	Name of the savepoint. The name is case-sensitive. If <code>savepoint_name</code> is not specified, the default is <code>LATEST</code> .

### Usage Notes

You are presented a read-only view of the workspace at the time of savepoint creation. This procedure is useful for examining the workspace from different savepoints before performing a rollback to a specific savepoint by calling the [RollbackToSP](#) procedure to delete all rows from that savepoint forward.

This operation can be executed while users exist in the workspace. There are no explicit privileges associated with this operation.

If you do not want to roll back to the savepoint, you can call the `GotoSavepoint` procedure with a null parameter to go to the currently active version in the workspace. (This achieves the same result as calling the [GotoWorkspace](#) procedure and specifying the workspace.)

For more information about savepoints, including the `LATEST` savepoint, see [Using Savepoints](#).

## Examples

The following example goes to the savepoint named `Savepoint1`.

```
EXECUTE DBMS_WM.GotoSavepoint ('Savepoint1');
```

## 4.46 GotoWorkspace

Moves the current session to the specified workspace.

### Syntax

```
DBMS_WM.GotoWorkspace(  
    workspace IN VARCHAR2);
```

### Parameters

**Table 4-37 GotoWorkspace Procedure Parameters**

Parameter	Description
<code>workspace</code>	Name of the workspace. The name is case-sensitive.

### Usage Notes

After a user goes to a workspace, modifications to data can be made there.

To go to the live database, specify `workspace` as `LIVE`. Because many operations are prohibited when any users (including you) are in the workspace, it is often convenient to go to the `LIVE` workspace before performing operations on created workspaces.

An exception is raised if one or more of the following apply:

- `workspace` does not exist.
- The user does not have `ACCESS_WORKSPACE` privilege for `workspace`.
- `workspace` has been frozen in `NO_ACCESS` mode (see the [FreezeWorkspace](#) procedure).

### Examples

The following example includes the user in the `NEWWORKSPACE` workspace. The user will begin to work in the latest version in that workspace.

```
EXECUTE DBMS_WM.GotoWorkspace ('NEWWORKSPACE');
```

The following example includes the user in the `LIVE` database workspace. By default, when users connect to a database, they are placed in this workspace.

```
EXECUTE DBMS_WM.GotoWorkspace ('LIVE');
```

## 4.47 GrantGraphPriv

Grants privileges on multiparent graph workspaces to users and roles. The `grant_option` parameter enables the grantee to grant the specified privileges to other users and roles.

## Syntax

```
DBMS_WM.GrantGraphPriv(
    priv_types      IN VARCHAR2,
    leaf_workspace IN VARCHAR2,
    grantee         IN VARCHAR2,
    node_types      IN VARCHAR2 DEFAULT (('R','I','L')),
    grant_option    IN VARCHAR2 DEFAULT 'NO',
    auto_commit     IN BOOLEAN  DEFAULT TRUE);
```

## Parameters

**Table 4-38 GrantGraphPriv Procedure Parameters**

Parameter	Description
priv_types	A string of one or more keywords representing privileges. ( <a href="#">Privilege Management with Workspace Manager</a> discusses Workspace Manager privileges.) Use commas to separate privilege keywords. The available keywords are ACCESS_WORKSPACE, MERGE_WORKSPACE, CREATE_WORKSPACE, REMOVE_WORKSPACE, ROLLBACK_WORKSPACE, and FREEZE_WORKSPACE.
leaf_workspace	Name of the leaf workspace in the directed acyclic graph. (Leaf workspaces, directed acyclic graphs, and other concepts related to multiparent workspaces are explained in <a href="#">Multiparent Workspaces</a> .) The name is case-sensitive.
grantee	Name of the user (can be the PUBLIC user group) or role to which to grant priv_types.
node_types	List of letters (in parentheses and comma-delimited) representing the types of nodes on which to grant the privileges: R for the root of the graph, I for the specified intermediate node, L for the leaf of the graph. The default is all types of nodes.
grant_option	Specify YES to enable the grant option for grantee, or NO (the default) to disable the grant option for grantee. The grant option allows grantee to grant the privileges specified in priv_types on the workspace specified in leaf_workspace to other users and roles.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a> .

## Usage Notes

Contrast this procedure with [GrantWorkspacePriv](#), which grants workspace-level Workspace Manager privileges on workspaces other than multiparent graph workspaces.

If a user gets a privilege from more than one source and if any of those sources has the grant option for that privilege, the user has the grant option for the privilege. For

example, assume that user `SCOTT` has been granted the `ACCESS_WORKSPACE` privilege with `grant_option` as `NO`, but that the `PUBLIC` user group has been granted the `ACCESS_WORKSPACE` privilege with `grant_option` as `YES`. Because user `SCOTT` is a member of `PUBLIC`, user `SCOTT` has the `ACCESS_WORKSPACE` privilege with the grant option.

The `WM_ADMIN_ROLE` role has all Workspace Manager privileges with the grant option. The `WM_ADMIN_ROLE` role is automatically given to the `DBA` role.

The `ACCESS_WORKSPACE` or `ACCESS_ANY_WORKSPACE` privilege is needed for all other Workspace Manager privileges.

To revoke workspace-level privileges on multiparent graph workspaces, use the [RevokeGraphPriv](#) procedure.

An exception is raised if one or more of the following apply:

- `grantee` is not a valid user or role in the database.
- You do not have the privilege to grant `priv_types`.
- `auto_commit` is `TRUE` and an open transaction exists in a parent or child workspace of any table that needs to be modified.

### Examples

The following example enables user `Smith` to access all types of nodes in the directed acyclic graph in which the `NEWWORKSPACE` workspace is the leaf workspace and to merge changes in these workspaces, and it allows `Smith` to grant the two specified privileges on the leaf workspace to other users.

```
DBMS_WM.GrantGraphPriv ('ACCESS_WORKSPACE, MERGE_WORKSPACE', 'NEWWORKSPACE', 'Smith',
'YES');
```

## 4.48 GrantPrivsOnPolicy

Grants the privileges required to call the [EnableVersioning](#) procedure on a table that contains the specified Oracle Label Security (OLS) policy.

### Syntax

```
DBMS_WM.GrantPrivsOnPolicy(
    policy_name IN VARCHAR2);
```

### Parameters

**Table 4-39 GrantPrivsOnPolicy Procedure Parameters**

Parameter	Description
<code>policy_name</code>	Name of the policy for which privileges need to be granted.

### Usage Notes

This procedure grants the necessary privileges on an OLS policy to the `WMSYS` schema. These privileges are required when executing workspace operations. If multiple tables protected by the same policy need to be version-enabled, this procedure only needs to be executed once.

## Examples

The following grants the necessary privileges on a policy named `my_policy`.

```
EXECUTE DBMS_WM.GrantPrivsOnPolicy('my_policy');
```

## 4.49 GrantSystemPriv

Grants system-level privileges (not restricted to a particular workspace) to users and roles. The `grant_option` parameter enables the grantee to grant the specified privileges to other users and roles.

### Syntax

```
DBMS_WM.GrantSystemPriv(  
    priv_types    IN VARCHAR2,  
    grantee       IN VARCHAR2,  
    grant_option  IN VARCHAR2 DEFAULT 'NO',  
    auto_commit   IN BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 4-40 GrantSystemPriv Procedure Parameters**

Parameter	Description
<code>priv_types</code>	A string of one or more keywords representing privileges. ( <a href="#">Privilege Management with Workspace Manager</a> discusses Workspace Manager privileges.) Use commas to separate privilege keywords. The available keywords are <code>ACCESS_ANY_WORKSPACE</code> , <code>MERGE_ANY_WORKSPACE</code> , <code>CREATE_ANY_WORKSPACE</code> , <code>REMOVE_ANY_WORKSPACE</code> , <code>ROLLBACK_ANY_WORKSPACE</code> , and <code>FREEZE_ANY_WORKSPACE</code> .
<code>grantee</code>	Name of the user (can be the <code>PUBLIC</code> user group) or role to which to grant <code>priv_types</code> .
<code>grant_option</code>	Specify <code>YES</code> to enable the grant option for grantee, or <code>NO</code> (the default) to disable the grant option for grantee. The grant option allows grantee to grant the privileges specified in <code>priv_types</code> to other users and roles.
<code>auto_commit</code>	A Boolean value ( <code>TRUE</code> or <code>FALSE</code> ).  <code>TRUE</code> (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes.  <code>FALSE</code> causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a> .

### Usage Notes

Contrast this procedure with [GrantWorkspacePriv](#), which grants workspace-level Workspace Manager privileges with keywords that do not contain `ANY` and which has a `workspace` parameter.

If a user gets a privilege from more than one source and if any of those sources has the grant option for that privilege, the user has the grant option for the privilege. For

example, assume that user SCOTT has been granted the ACCESS\_ANY\_WORKSPACE privilege with grant\_option as NO, but that the PUBLIC user group has been granted the ACCESS\_ANY\_WORKSPACE privilege with grant\_option as YES. Because user SCOTT is a member of PUBLIC, user SCOTT has the ACCESS\_ANY\_WORKSPACE privilege with the grant option.

The WM\_ADMIN\_ROLE role has all Workspace Manager privileges with the grant option. The WM\_ADMIN\_ROLE role is automatically given to the DBA role.

The ACCESS\_WORKSPACE or ACCESS\_ANY\_WORKSPACE privilege is needed for all other Workspace Manager privileges.

To see which users have been granted Workspace Manager system-level privileges, examine the DBA\_WM\_SYS\_PRIVS metadata view, which is described in DBA\_WM\_SYS\_PRIVS.

To revoke system-level privileges, use the RevokeSystemPriv procedure.

An exception is raised if one or more of the following apply:

- grantee is not a valid user or role in the database.
- You do not have the privilege to grant priv\_types.
- auto\_commit is TRUE and an open transaction exists in a parent or child workspace of any table that needs to be modified.

### Examples

The following example enables user Smith to access any workspace in the database, but does not allow Smith to grant the ACCESS\_ANY\_WORKSPACE privilege to other users.

```
EXECUTE DBMS_WM.GrantSystemPriv ('ACCESS_ANY_WORKSPACE', 'Smith', 'NO');
```

## 4.50 GrantWorkspacePriv

Grants workspace-level privileges to users and roles. The grant\_option parameter enables the grantee to grant the specified privileges to other users and roles.

### Syntax

```
DBMS_WM.GrantWorkspacePriv(
  priv_types      IN VARCHAR2,
  workspace      IN VARCHAR2,
  grantee        IN VARCHAR2,
  grant_option    IN VARCHAR2 DEFAULT 'NO',
  auto_commit    IN BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 4-41 GrantWorkspacePriv Procedure Parameters**

Parameter	Description
priv_types	A string of one or more keywords representing privileges. ( <a href="#">Privilege Management with Workspace Manager</a> discusses Workspace Manager privileges.) Use commas to separate privilege keywords. The available keywords are ACCESS_WORKSPACE, MERGE_WORKSPACE, CREATE_WORKSPACE, REMOVE_WORKSPACE, ROLLBACK_WORKSPACE, and FREEZE_WORKSPACE.

**Table 4-41 (Cont.) GrantWorkspacePriv Procedure Parameters**

Parameter	Description
workspace	Name of the workspace. The name is case-sensitive.
grantee	Name of the user (can be the PUBLIC user group) or role to which to grant <code>priv_types</code> .
grant_option	Specify YES to enable the grant option for grantee, or NO (the default) to disable the grant option for grantee. The grant option allows grantee to grant the privileges specified in <code>priv_types</code> on the workspace specified in <code>workspace</code> to other users and roles.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a> .

### Usage Notes

Contrast this procedure with [GrantSystemPriv](#), which grants system-level Workspace Manager privileges with keywords in the form `xxx_ANY_WORKSPACE` (`ACCESS_ANY_WORKSPACE`, `MERGE_ANY_WORKSPACE`, and so on). Contrast this procedure also with [GrantGraphPriv](#), which grants privileges on multiparent graph workspaces to users and roles.

If a user gets a privilege from more than one source and if any of those sources has the grant option for that privilege, the user has the grant option for the privilege. For example, assume that user SCOTT has been granted the `ACCESS_WORKSPACE` privilege with `grant_option` as NO, but that the PUBLIC user group has been granted the `ACCESS_WORKSPACE` privilege with `grant_option` as YES. Because user SCOTT is a member of PUBLIC, user SCOTT has the `ACCESS_WORKSPACE` privilege with the grant option.

The `WM_ADMIN_ROLE` role has all Workspace Manager privileges with the grant option. The `WM_ADMIN_ROLE` role is automatically given to the DBA role.

The `ACCESS_WORKSPACE` or `ACCESS_ANY_WORKSPACE` privilege is needed for all other Workspace Manager privileges.

To revoke workspace-level privileges, use the [RevokeWorkspacePriv](#) procedure.

An exception is raised if one or more of the following apply:

- `grantee` is not a valid user or role in the database.
- You do not have the privilege to grant `priv_types`.
- `auto_commit` is TRUE and an open transaction exists in a parent or child workspace of any table that needs to be modified.



## Examples

The following example enables user `Smith` to access the `NEWWORKSPACE` workspace and merge changes in that workspace, and allows `Smith` to grant the two specified privileges on `NEWWORKSPACE` to other users.

```
DBMS_WM.GrantWorkspacePriv ('ACCESS_WORKSPACE, MERGE_WORKSPACE', 'NEWWORKSPACE',
'Smith', 'YES');
```

## 4.51 Import

Imports data from a staging table (all rows, or as limited by any combination of several parameters) into a version-enabled table in a specified workspace.

### Syntax

```
DBMS_WM.Import(
  staging_table   IN VARCHAR2,
  to_table       IN VARCHAR2,
  to_workspace   IN VARCHAR2,
  from_workspace IN VARCHAR2 DEFAULT NULL,
  where_clause   IN VARCHAR2 DEFAULT NULL,
  import_scope   IN VARCHAR2 DEFAULT DBMS_WM.IMPORT_ALL_DATA,
  ancestor_savepoint_workspace IN VARCHAR2 DEFAULT NULL,
  ancestor_savepoint_name      IN VARCHAR2 DEFAULT NULL,
  apply_locks     IN BOOLEAN DEFAULT FALSE,
  enforceUCFlag   IN BOOLEAN DEFAULT TRUE,
  enforceRICFlag  IN BOOLEAN DEFAULT TRUE,
  auto_commit     IN BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 4-42 Import Procedure Parameters**

Parameter	Description
<code>staging_table</code>	Name of the table that holds the data that had previously been exported using the <a href="#">Export</a> procedure. The name is not case-sensitive.
<code>to_table</code>	Name of the table into which to import the data. The name is not case-sensitive.
<code>to_workspace</code>	Name of the workspace in which to import the data. The name is case-sensitive.
<code>from_workspace</code>	Name of the workspace from which to import the data. The name is case-sensitive. If the staging table contains versioning information, you must specify <code>from_workspace</code> .
<code>where_clause</code>	The <code>WHERE</code> clause (excluding the <code>WHERE</code> keyword) identifying the rows to be imported. Example: <code>'department_id = 20'</code> Only primary key columns can be specified in the <code>WHERE</code> clause, except in a subquery. The subquery can refer to columns that are not primary keys, but it cannot refer to a version-enabled table. If the <code>where_clause</code> parameter is not specified, all rows in <code>staging_table</code> are imported.

Table 4-42 (Cont.) Import Procedure Parameters

Parameter	Description
<code>import_scope</code>	<p>The scope (amount of data) for the import operation.</p> <p><code>DBMS_WM.IMPORT_ALL_DATA</code> (the default) imports all relevant data.</p> <p><code>DBMS_WM.IMPORT_MODIFIED_DATA_ONLY</code> imports only relevant data that has been inserted, updated, or deleted in <code>from_workspace</code>.</p>
<code>ancestor_savepoint_workspace</code>	<p>Name of the workspace containing the ancestor savepoint specified in <code>ancestor_savepoint_name</code>. For the current release, if you specify <code>ancestor_savepoint_workspace</code>, the value must be <code>LIVE</code>.</p> <p>If you specify this parameter, you must also specify <code>ancestor_savepoint_name</code>.</p>
<code>ancestor_savepoint_name</code>	<p>Name of a savepoint in <code>ancestor_savepoint_workspace</code>. All data that was ancestor data at the time of the export operation (see the Usage Notes for the <a href="#">Export</a> procedure) is imported to the specified savepoint. For the current version, if you specify <code>ancestor_savepoint_name</code>, the value must be <code>DBMS_WM.ROOT_VERSION</code>.</p> <p>If you specify this parameter, you must also specify <code>ancestor_savepoint_workspace</code>.</p>
<code>apply_locks</code>	<p>A Boolean value (<code>TRUE</code> or <code>FALSE</code>).</p> <p><code>TRUE</code> causes any locks that were present on the exported data to be applied to the data when importing, unless a more restrictive lock mode is in effect for the current session.</p> <p><code>FALSE</code> (the default) ignores any locks on rows in the staging table, but instead always uses the lock mode is in effect for the current session.</p>
<code>enforceUCFlag</code>	<p>A Boolean value (<code>TRUE</code> or <code>FALSE</code>).</p> <p><code>TRUE</code> (the default) enforces any unique constraints defined on <code>to_table</code>, ensuring that the import operation does not violate any such constraints.</p> <p><code>FALSE</code> does not enforce any unique constraints defined on <code>to_table</code> for the import operation.</p>
<code>enforceRICFlag</code>	<p>A Boolean value (<code>TRUE</code> or <code>FALSE</code>).</p> <p><code>TRUE</code> (the default) enforces any referential integrity constraints defined on <code>to_table</code>, ensuring that the import operation does not violate any such constraints.</p> <p><code>FALSE</code> does not enforce any referential integrity constraints defined on <code>to_table</code> for the import operation.</p>

**Table 4-42 (Cont.) Import Procedure Parameters**

Parameter	Description
auto_commit	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes.</p> <p>FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a>.</p>

### Usage Notes

All data that satisfies the `where_clause` parameter value in the staging table named `staging_table` and the `import_scope` parameter value is imported into the version-enabled table named `to_table`.

The data must have been previously exported to the staging table using the [Export](#) procedure.

Each row of data to be imported is considered to be one of the following: inserted, updated, or deleted in `from_workspace` (that is, *modified data*); or data that was not modified in `from_workspace` but can be seen in it (that is, *ancestor data*). If data is exported from the LIVE workspace, it is all modified data.

An exception is raised if one or more of the following apply:

- A specified table or workspace does not exist.
- `staging_table` is not in a valid format for the import operation.
- `to_table` is not a version-enabled table, or does not have an appropriate definition (for example, contains columns not in the staging table).
- `from_workspace` is null and `staging_table` contains versioning information.
- `ancestor_savepoint_name` is not a valid savepoint in `ancestor_savepoint_workspace`.
- `auto_commit` is TRUE and an open transaction exists in a parent or child workspace of any table that needs to be modified.

See also [Import and Export Considerations](#).

### Examples

The following example imports modified data from the staging table `COLA_MARKETING_BUDGET_STG` in workspace `B_focus_2` into the `COLA_MARKETING_BUDGET` table in workspace `B_Focus_1`. (The EXECUTE statement is actually on a single line.)

```
EXECUTE DBMS_WM.Import(staging_table => 'COLA_MARKETING_BUDGET_STG',
  to_table => 'COLA_MARKETING_BUDGET', to_workspace => 'B_focus_1',
  from_workspace => 'B_focus_2');
```

## 4.52 Import\_Schemas

Imports the entire Workspace Manager installation from a dump file that had been created by the [Export\\_Schemas](#) procedure. Uses the Oracle Data Pump Import utility.

### Syntax

```
DBMS_WM.Import_Schemas(
  job_name          IN VARCHAR2,
  alt_schema        IN VARCHAR2 DEFAULT 'WMSYS_N',
  ignore_last_error IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 4-43 Import\_Schemas Procedure Parameters**

Parameter	Description
job_name	Name of the Data Pump job to be used for the import operation.
alt_schema	Name of the database schema to be used for temporary storage of Workspace Manager metadata. The default is <code>WMSYS_N</code> . This schema must not already exist when the procedure is called, and it is automatically dropped when the procedure completes successfully.
ignore_last_error	A Boolean value ( <code>TRUE</code> or <code>FALSE</code> ).  <code>TRUE</code> ignores the last error, if any, that occurred during the previous call to the <code>Import_Schemas</code> procedure. Information about the last error is stored in the <a href="#">DBA_WM_VT_ERRORS</a> static data dictionary view, which is described in <a href="#">Workspace Manager Static Data Dictionary Views</a> . See the Usage Notes for more information.  <code>FALSE</code> (the default) does not ignore the last error, if any, that occurred during the previous call to the <code>Import_Schemas</code> procedure.

### Usage Notes

This procedure uses a dump file that had been created using the [Export\\_Schemas](#) procedure. There must be no existing version-enabled tables or workspaces, other than `LIVE`, before you call this procedure. All objects of the schemas contained in the dump file are imported. If any system or workspace privileges or any privileges on version-enabled tables were granted to users that were not contained in the generated dump file, those schemas must be created before you call this procedure; otherwise, the grants will be lost.

This procedure makes use of an already existing Data Pump Import job. When you create this job using the `DBMS_DATAPUMP.OPEN` procedure, the `operation` parameter must be set to `IMPORT` and the `mode` parameter must be set to `FULL`. The dump file(s) and log file should also be specified before you call this procedure. No procedures that modify or limit what gets imported (such as `DBMS_DATAPUMP.METADATA_FILTER`) should be executed on this job. . The Data Pump job should not be created while using `SYSDBA` privileges.

The schema specified by the `alt_schema` parameter cannot exist before you call this procedure. It must also be the same schema as specified for `alt_schema` when you called the [Export\\_Schemas](#) procedure.

For information about using the Data Pump Utility, see *Oracle Database Utilities*.

If a call to the `Import_Schemas` procedure fails, you should try to fix the cause of the error. Examine the [DBA\\_WM\\_VT\\_ERRORS](#) static data dictionary view where the `STATE` column is equal to `IMPORT` to see the SQL statement and error message. Fix the cause of the error, and then call the `Import_Schemas` procedure again with the default `ignore_last_error` parameter value of `FALSE`. However, if the call still fails and you cannot fix the cause of the error, and if you are sure that it is safe and appropriate to ignore this error, then you can ignore the error by calling the `Import_Schemas` procedure with the `ignore_last_error` parameter value of `TRUE`. Note that you are responsible for ensuring that it is safe and appropriate to ignore the error.

If the `Import_Schemas` procedure fails and if you must execute the procedure again, and if a row for this procedure exists in the [DBA\\_WM\\_VT\\_ERRORS](#) static data dictionary view, the procedure will continue to use the original Data Pump job that was specified (you do not need to create a new job). However, if the SQL statements being executed attempt to use the original job but that job no longer exists, you must set the `ignore_last_error` parameter to `TRUE` and execute the `Import_Schemas` procedure again.

An exception is raised if one or more of the following apply:

- `job_name` does not exist.
- `alt_schema` already exists.
- The executing user does not have the `DATAPUMP_IMP_FULL_DATABASE` role.
- Errors exist in the `WMSYS` or user schemas.

See also [Import and Export Considerations](#).

## Examples

The following example imports the Workspace Manager metadata using the Oracle Data Pump job named `IMPORT_OWM_SCHEMAS`.

```
DECLARE
  job_name varchar2(128) := upper('IMPORT_OWM_SCHEMAS') ;
  dpj number ;
BEGIN
  dpj := dbms_datapump.open('IMPORT', 'FULL', null, job_name, 'COMPATIBLE') ;
  dbms_datapump.add_file(dpj, 'owm_schema.dmp', 'DUMP_DIR') ;
  dbms_datapump.add_file(dpj, 'owm_schema_import.log', 'DUMP_DIR',
filetype=>dbms_datapump.KU$_FILE_TYPE_LOG_FILE) ;
  dbms_wm.import_schemas(job_name) ;
  dbms_datapump.detach(dpj);

exception when others then
  dbms_datapump.detach(dpj);
  raise ;
END;
/
```

The following example imports the Workspace Manager metadata from the Oracle Cloud Infrastructure Object Storage using the `dbms_datapump` API job. The example uses a pre-authenticated request URL to access the `owm_test.dmp` dump file in the Object Storage.

Ensure that Oracle Workspace Manager is enabled on your Autonomous Database instance. See [Enable Oracle Workspace Manager on Autonomous Database](#) for more information.

 **Note:**

Oracle Workspace Manager is enabled on Autonomous Database Serverless only in selected cloud regions. The following table lists the cloud regions that support Oracle Workspace Manager:

**Table 4-44 Supported Cloud Regions for Oracle Workspace Manager**

Region Name	Region Identifier
Singapore (Singapore)	ap-singapore-1
France South (Marseille)	eu-marseille-1
UAE Central (Abu Dhabi)	me-abudhabi-1
Italy Northwest (Milan)	eu-milan-1
Sweden Central (Stockholm)	eu-stockholm-1
South Africa Central (Johannesburg)	af-johannesburg-1
France Central (Paris)	eu-paris-1

```

DECLARE
    hdl          NUMBER;          -- Datapump handle
    dumpFile     VARCHAR2(1024)  := 'https://
objectstorage.<region>.oraclecloud.com/p/<encrypted_string>/n/<namespace-
string>/b/<bucket_name>/o/owm_test.dmp';
    dumpType     NUMBER          := dbms_datapump.ku$_file_type_uridump_file;
    credName     VARCHAR2(1024)  := 'CRED_NAME';
    logFile      VARCHAR2(1024)  := 'import_test.log';
    logDir       VARCHAR2(20)    := 'DATA_PUMP_DIR';
    logType      NUMBER          := dbms_datapump.ku$_file_type_log_file;
    job_name     VARCHAR2(128)   := DBMS_SCHEDULER.GENERATE_JOB_NAME('IMP_SD_');
BEGIN
    hdl := dbms_datapump.open('IMPORT', 'FULL', null, job_name, 'COMPATIBLE');
    dbms_output.put_line(' LOCAL ' || job_name);
    dbms_datapump.add_file(hdl, dumpFile, credName, null, dumpType);
    dbms_datapump.add_file(hdl, logfile, logdir, null, logType);
    dbms_wm.import_schemas('ADMIN.' || job_name, 'WMSYS_N');
EXCEPTION
    WHEN OTHERS THEN
        dbms_output.put_line(SQLERRM);
        dbms_output.put_line( dbms_utility.format_error_backtrace );
END;
/

```

## 4.53 Initialize\_After\_Import

Creates (initializes) a version enabled topology that was imported from another database.

**Format**

```
DBMS_WM.Add_Topo_Geometry_Layer(
  topology      IN VARCHAR2,
  tg_layer_owner IN VARCHAR2 DEFAULT NULL);
```

**Parameters****Table 4-45 Initialize\_After\_Import Procedure Parameters**

Parameter	Description
topology	Topology that was imported from another database.
tg_layer_owner	Owner of the topology layer. If this parameter is null (the default), the current user is the owner.

**Usage Notes**

This procedure creates the specified version-enabled topology and related database structures, adjusts the topology ID values in all feature tables, and creates the feature layers in the correct order. It also generates the necessary Workspace Manager metadata on the spatial topology index tables that get created. For information about Workspace Manager support for topologies, see [Spatial and Graph Topology Support](#).

After importing a version-enabled topology using either a full database import or the DBMS\_WM.[Import\\_Schemas](#) procedure, you must do the following (with the last step being to execute the DBMS\_WM.Initialize\_After\_Import procedure):

1. Connect to the target database, that is, the database in which to create a topology with the same name, structures, and data as the topology exported from the source database. Connect as the user for the schema that is to own the topology to be created.
2. Ensure that the target database does not already contain a topology with the same name as the topology in the .dmp file.
3. Perform the import using either a full database import or the DBMS.[Import\\_Schemas](#) procedure.
4. Execute the DBMS\_WM.Initialize\_After\_Import procedure. (Do **not** execute the SDO\_TOPO.Initialize\_After\_Import procedure.)

**Examples**

The following example creates the topology named CITY\_DATA using information from the imported tables, including CITY\_DATA\_EXP\$, which was created using the SDO\_TOPO.Prepare\_For\_Export procedure.

```
EXECUTE DBMS_WM.INITIALIZE_AFTER_IMPORT('CITY_DATA');
```

## 4.54 IsWorkspaceOccupied

Checks whether or not a workspace has any active sessions.

**Syntax**

```
DBMS_WM.IsWorkspaceOccupied(
    workspace IN VARCHAR2) RETURN VARCHAR2;
```

**Parameters****Table 4-46 IsWorkspaceOccupied Function Parameters**

Parameter	Description
workspace	Name of the workspace. The name is case-sensitive.

**Usage Notes**

This function returns **YES** if the workspace has any active sessions, and it returns **NO** if the workspace has no active sessions.

An exception is raised if the **LIVE** workspace is specified or if the user does not have the privilege to access the workspace.

**Examples**

The following example checks if any sessions are in the **B\_focus\_2** workspace.

```
SELECT DBMS_WM.IsWorkspaceOccupied('B_focus_2') FROM DUAL;
```

```
DBMS_WM.ISWORKSPACEOCCUPIED('B_FOCUS_2')
```

```
-----
YES
```

## 4.55 LockRows

Controls access to versioned rows in a specified table and to corresponding rows in the parent workspace.

**Syntax**

```
DBMS_WM.LockRows(
    workspace      IN VARCHAR2,
    table_name     IN VARCHAR2,
    where_clause   IN VARCHAR2 DEFAULT '',
    lock_mode      IN VARCHAR2 DEFAULT 'E',
    Xmin           IN NUMBER DEFAULT NULL,
    Ymin           IN NUMBER DEFAULT NULL,
    Xmax           IN NUMBER DEFAULT NULL,
    Ymax           IN NUMBER DEFAULT NULL);
```



## Parameters

Table 4-47 LockRows Procedure Parameters

Parameter	Description
workspace	<p>Name of the workspace. The latest versions of rows visible from the workspace are locked. If a row has not been modified in this workspace, the locked version could be in an ancestor workspace. The name is case-sensitive.</p> <p>A value of <code>NONE</code> can be used if <code>lock_mode</code> is set to <code>VE</code> (version-exclusive). This causes the latest versions of rows to be locked, regardless of the workspaces from which they are visible.</p>
table_name	<p>Name of the table or (if <code>Xmin</code>, <code>Ymin</code>, <code>Xmax</code>, and <code>Ymax</code> are specified) Spatial and Graph topology in which rows are to be locked. The name is not case-sensitive.</p>
where_clause	<p>The <code>WHERE</code> clause (excluding the <code>WHERE</code> keyword) identifying the rows to be locked. Example: <code>'department_id = 20'</code></p> <p>Only primary key columns can be specified in the <code>WHERE</code> clause, except in a subquery. The subquery can refer to columns that are not primary keys, but it cannot refer to a version-enabled table.</p> <p>If <code>where_clause</code> is not specified, all rows in <code>table_name</code> are locked.</p> <p>Do not specify the <code>where_clause</code> parameter if <code>table_name</code> specifies a Spatial and Graph topology name.</p>
lock_mode	<p>Mode with which to set the locks: <code>E</code> (exclusive), <code>WE</code> (workspace-exclusive), <code>VE</code> (version-exclusive), or <code>S</code> (shared). The default is <code>E</code>.</p> <p><code>E</code> (exclusive) mode locks the rows in the previous version and the corresponding rows in the current version; no other users in the workspace for either version can change any values.</p> <p><code>WE</code> (workspace-exclusive) mode locks the rows in the previous version and the corresponding rows in the current version such that only the user that set the lock can change the values in the current workspace; however, other users in other workspaces can change the values.</p> <p><code>VE</code> (version-exclusive) mode locks the rows in the previous version and the corresponding rows in the current version such that only the user that set the lock can change the values; no other users (in any workspace) can change the values.</p> <p><code>S</code> (shared) mode locks the rows in the previous version and the corresponding rows in the current version; however, other users in the workspace for the current version (but no users in the workspace for the previous version) can change values in these rows.</p>
Xmin, Ymin	<p>For Oracle Spatial and Graph topologies only (see <a href="#">Locking Considerations with Topologies</a>), the X and Y coordinate values, respectively, of the lower-left corner of the window containing the rows to be locked. You must specify these parameters if you specified a topology name for <code>table_name</code>; otherwise, do not specify these parameters.</p>
Xmax, Ymax	<p>For Oracle Spatial and Graph topologies only (see <a href="#">Locking Considerations with Topologies</a>), the X and Y coordinate values, respectively, of the upper-right corner of the window containing the rows to be locked. You must specify these parameters if you specified a topology name for <code>table_name</code>; otherwise, do not specify these parameters.</p>

## Usage Notes

This procedure affects Workspace Manager locking, which occurs in addition to any standard Oracle database locking. For an explanation of Workspace Manager locking, see [Lock Management with Workspace Manager](#).

This procedure does not affect whether Workspace Manager locking is set on or off (determined by the [SetLockingON](#) and [SetLockingOFF](#) procedures).

To unlock rows, use the [UnlockRows](#) procedure.

For information about Workspace Manager locking for tables in an Oracle Spatial and Graph topology, see [Locking Considerations with Topologies](#).

## Examples

The following example locks rows in the EMPLOYEES table where last\_name = 'Smith' in the NEWWORKSPACE workspace.

```
EXECUTE DBMS_WM.LockRows ('NEWWORKSPACE', 'employees', 'last_name = ''Smith''');
```

# 4.56 MergeTable

Applies changes to one or more tables (all rows or as specified in the WHERE clause) in a workspace to its parent workspace.

For a multiparent workspace (explained in [Multiparent Workspaces](#)), applies changes to one or more tables (all rows or as specified in the WHERE clause) from all non-root workspaces in the directed acyclic graph to the multiparent root workspace.

## Syntax

```
DBMS_WM.MergeTable(
  workspace      IN VARCHAR2,
  table_id       IN VARCHAR2,
  where_clause    IN VARCHAR2 DEFAULT '',
  create_savepoint IN BOOLEAN DEFAULT FALSE,
  remove_data    IN BOOLEAN DEFAULT FALSE,
  auto_commit     IN BOOLEAN DEFAULT TRUE);
```

## Parameters

**Table 4-48 MergeTable Procedure Parameters**

Parameter	Description
workspace	Name of the workspace. The name is case-sensitive.
table_id	Name of the table or tables containing rows to be merged into the parent workspace. To specify multiple tables, separate the names with commas (for example, 'table1, table2'). The names are not case-sensitive.

**Table 4-48 (Cont.) MergeTable Procedure Parameters**

Parameter	Description
<code>where_clause</code>	<p>The <code>WHERE</code> clause (excluding the <code>WHERE</code> keyword) identifying the rows to be merged into the parent workspace. Example: <code>'department_id = 20'</code></p> <p>Only primary key columns can be specified in the <code>WHERE</code> clause, except in a subquery. The subquery can refer to columns that are not primary keys, but it cannot refer to a version-enabled table.</p> <p>If the <code>where_clause</code> parameter is not specified, all rows in <code>table_name</code> are merged.</p>
<code>create_savepoint</code>	<p>A Boolean value (<code>TRUE</code> or <code>FALSE</code>).</p> <p><code>TRUE</code> creates an implicit savepoint in the parent workspace before the merge operation. For a multiparent workspace, creates an implicit savepoint in the multiparent root workspace before the merge operation. (Implicit and explicit savepoints are described in <a href="#">Using Savepoints</a>.)</p> <p><code>FALSE</code> (the default) does not create an implicit savepoint in the parent workspace before the merge operation.</p>
<code>remove_data</code>	<p>A Boolean value (<code>TRUE</code> or <code>FALSE</code>).</p> <p><code>TRUE</code> removes the data in the table (as specified by the <code>where_clause</code> parameter) in the child workspace. For a multiparent workspace, it removes the data in the table (as specified by the <code>where_clause</code> parameter) in the non-root workspaces in the directed acyclic graph. The <code>remove_data</code> option is permitted only if <code>workspace</code> has no child workspaces (that is, it is a leaf workspace).</p> <p><code>FALSE</code> (the default) does not remove the data in the table in the child workspace.</p>
<code>auto_commit</code>	<p>A Boolean value (<code>TRUE</code> or <code>FALSE</code>).</p> <p><code>TRUE</code> (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes.</p> <p><code>FALSE</code> causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a>.</p>

### Usage Notes

All data that satisfies the `where_clause` parameter value in the version-enabled table named `table_name` in `workspace` is applied to the parent workspace of `workspace`.

If the `create_savepoint` parameter value is `true`, an implicit savepoint is created only if one does not already exist on the intended version. This intended version is the LATEST version in the parent workspace if that version contains modified rows. Otherwise, this is the previous version to the LATEST version as long as the workspace has more than a single version.

Any locks that are held by rows being merged are released.

If there are conflicts between the workspace being merged and its parent workspace, the merge operation fails and the user must manually resolve conflicts using the <table\_name>\_CONF view. (Conflict resolution is explained in [Resolving Conflicts Before a Merge or Refresh Operation](#).)

A table cannot be merged in the LIVE workspace (because that workspace has no parent workspace).

An exception is raised if one or more of the following apply:

- The user does not have access to `table_id`.
- The user does not have the `MERGE_WORKSPACE` privilege for `workspace` or the `MERGE_ANY_WORKSPACE` privilege.
- `remove_data` is TRUE and there are any child workspaces of any workspace to be removed.
- `auto_commit` is TRUE and an open transaction exists in a parent or child workspace of any table that needs to be modified.
- The merge involving a multiparent workspace would cause the violation of a referential integrity constraint or unique constraint in any continually refreshed child workspace of the multiparent root workspace.

### Examples

The following example merges changes to the EMP table (in the USER3 schema) where `last_name = 'Smith'` in NEWWORKSPACE to its parent workspace.

```
EXECUTE DBMS_WM.MergeTable ('NEWWORKSPACE', 'user3.emp', 'last_name =
'Smith');
```

## 4.57 MergeWorkspace

Applies all changes in a workspace to its parent workspace, and optionally removes the workspace.

For a multiparent workspace (explained in [Multiparent Workspaces](#)), applies all changes in the workspace to all other workspaces in the directed acyclic graph, and optionally removes the non-root workspaces in the directed acyclic graph.

### Syntax

```
DBMS_WM.MergeWorkspace(
  workspace          IN VARCHAR2,
  create_savepoint  IN BOOLEAN DEFAULT FALSE,
  remove_workspace  IN BOOLEAN DEFAULT FALSE,
  auto_commit       IN BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 4-49 MergeWorkspace Procedure Parameters**

Parameter	Description
<code>workspace</code>	Name of the workspace. The name is case-sensitive.

**Table 4-49 (Cont.) MergeWorkspace Procedure Parameters**

Parameter	Description
<code>create_savepoint</code>	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE creates an implicit savepoint in the parent workspace before the merge operation. (Implicit and explicit savepoints are described in <a href="#">Using Savepoints</a>.)</p> <p>FALSE (the default) does not create an implicit savepoint in the parent workspace before the merge operation.</p>
<code>remove_workspace</code>	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE removes <code>workspace</code> after the merge operation. For a multiparent workspace, all non-root workspaces in the directed acyclic graph are removed.</p> <p>FALSE (the default) does not remove <code>workspace</code> after the merge operation; the workspace continues to exist.</p>
<code>auto_commit</code>	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes.</p> <p>FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a>.</p>

### Usage Notes

All data in all version-enabled tables in `workspace` is merged to the parent workspace of `workspace`, and `workspace` is removed if `remove_workspace` is TRUE.

If `workspace` is a continually refreshed child workspace, an exclusive lock is taken on the parent workspace. This exclusive lock blocks other operations on the parent workspace, such as [GotoWorkspace](#), which would try to take a shared lock.

If the `create_savepoint` parameter value is `true`, an implicit savepoint is created only if one does not already exist on the intended version. This intended version is the LATEST version in the parent workspace if that version contains modified rows. Otherwise, this is the previous version to the LATEST version as long as the workspace has more than a single version.

Only the current row version for any given row is merged into the parent workspace. To retain all intermediate row versions and historical copies in the child workspace, the value of `remove_workspace` must be FALSE (the default). For more information about how Workspace Manager creates row versions and manages historical copies, see [Creation of Row Versions and Historical Copies](#).

While this procedure is executing, the current workspace is frozen in NO\_ACCESS mode and the parent workspace is frozen in READ\_ONLY mode, as explained in [Freezing and Unfreezing Workspaces](#).

If there are conflicts between the workspace being merged and its parent workspace, the merge operation fails and the user must manually resolve conflicts using the

<table\_name>\_CONF view. (Conflict resolution is explained in [Resolving Conflicts Before a Merge or Refresh Operation](#).)

If the `remove_workspace` parameter value is `TRUE`, the workspace to be merged must be a leaf workspace, that is, a workspace with no descendant workspaces. (For an explanation of workspace hierarchy, see [Workspace Hierarchy](#).)

To update rows in the child workspace and merge those changes into the parent workspace in the same transaction, you must specify `autocommit=FALSE` and ensure that no other session (that is, other than the one performing the update transaction) is in the child workspace.

An exception is raised if one or more of the following apply:

- The user does not have the `MERGE_WORKSPACE` privilege for `workspace` or the `MERGE_ANY_WORKSPACE` privilege.
- The user does not have sufficient privileges on all tables that need to be modified (including, for example, tables modified by triggers).
- `auto_commit` is `TRUE` and there is an open database transaction in any workspace under `workspace` in the workspace hierarchy.
- `remove_workspace` is `TRUE` and there are any sessions in any workspaces under `workspace` in the workspace hierarchy.
- `remove_workspace` is `TRUE` and there are any child workspaces of any workspace to be removed.
- `auto_commit` is `TRUE` and an open transaction exists in a parent or child workspace of any table that needs to be modified.
- The merge of a multiparent workspace would cause the violation of a referential integrity constraint or unique constraint in any continually refreshed child workspace of the multiparent root workspace.

### Examples

The following example merges changes in `NEWWORKSPACE` to its parent workspace.

```
EXECUTE DBMS_WM.MergeWorkspace ('NEWWORKSPACE');
```

## 4.58 Move\_Proc

Moves the Workspace Manager metadata to a specified tablespace.

### Syntax

```
DBMS_WM.Move_Proc(
    dest_tablespace IN VARCHAR2 DEFAULT 'SYSAUX');
```

### Parameters

**Table 4-50 Move\_Proc Procedure Parameters**

Parameter	Description
<code>dest_tablespace</code>	The table space to which to move the Workspace Manager metadata. The default value is the <code>SYSAUX</code> tablespace.

## Usage Notes

The Workspace Manager metadata (views, internal tables, and other objects) is by default stored in the default tablespace of the WMSYS user. You cannot directly control the size of the Workspace Manager metadata, but you can control its placement by using this procedure to move the metadata from its current tablespace to a different tablespace. If you call this procedure without specifying the `dest_tablespace` parameter, the Workspace manager metadata is moved to the SYSAUX tablespace.

Before you move the metadata, you can use the [GetWMMetadataSpace](#) function to determine the approximate minimum space that you will need to have available in the tablespace into which you are considering moving the Workspace Manager metadata.

## Examples

The following example moves the Workspace Manager metadata to the `TBLSP_1` tablespace.

```
EXECUTE DBMS_WM.Move_proc('TBLSP_1');
```

## 4.59 PurgeTable

Removes rows (all rows, or as limited by any combination of several parameters) from a version-enabled table, and optionally inserts them into an archive table.

### Syntax

```
DBMS_WM.PurgeTable(
    table_id          IN VARCHAR2,
    archive_table     IN VARCHAR2,
    where_clause      IN VARCHAR2,
    workspace         IN VARCHAR2 DEFAULT 'LIVE',
    savepoint_name    IN VARCHAR2 DEFAULT NULL,
    instant           IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    purgeAfter       IN BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 4-51 PurgeTable Procedure Parameters**

Parameter	Description
<code>table_id</code>	Name of the table containing the data to be exported. The name is not case-sensitive.
<code>archive_table</code>	Name of the table into which to insert the purged rows. If this parameter is specified as NULL, purged rows are not archived. If this parameter is specified as other than NULL and if there is an open transaction, the transaction is committed before the table is created, and a new transaction is opened.
<code>where_clause</code>	The WHERE clause (excluding the WHERE keyword) identifying the rows to be purged. Example: 'department_id = 20'  Only primary key columns can be specified in the WHERE clause, except in a subquery. The subquery can refer to columns that are not primary keys, but it cannot refer to a version-enabled table.  If the <code>where_clause</code> parameter is not specified, all rows in <code>table_name</code> are purged.

**Table 4-51 (Cont.) PurgeTable Procedure Parameters**

Parameter	Description
workspace	Name of the workspace from which to purge the data. The name is case-sensitive.
savepoint_name	Name of a savepoint: only data in the workspace either after or before (depending on the <code>purgeAfter</code> value) the time the savepoint was created is purged. You cannot specify both the <code>savepoint_name</code> and <code>instant</code> parameters.
instant	Date/time specification: only data that was in the workspace either after or before (depending on the <code>purgeAfter</code> value) this time is purged. You cannot specify both the <code>savepoint_name</code> and <code>instant</code> parameters.
purgeAfter	A Boolean value (TRUE or FALSE). TRUE (the default) causes rows in the workspace <i>after</i> any specified <code>savepoint_name</code> or <code>instant</code> value to be purged. FALSE causes rows in the workspace <i>before</i> any specified <code>savepoint_name</code> or <code>instant</code> value to be purged.

### Usage Notes

This procedure removes rows from a version-enabled table that is rooted at workspace. If the `purgeAfter` parameter value is TRUE (the default), applicable *child* rows rooted at the specified workspace are removed; if the `purgeAfter` parameter value is FALSE, applicable *ancestor* rows rooted at the specified workspace are removed.

You can use the `where_clause` parameter and the `savepoint_name` or `instant` parameter to limit the rows that are purged. For most uses of the procedure, you will probably want to specify a `where_clause` value to limit the rows to be purged; otherwise all rows are purged (unless limited by the `savepoint_name` or `instant` parameter).

An exclusive lock is obtained on the version-enabled table for the duration of the procedure.

### Examples

The following example purges any rows where the ID (primary key) column value is 20 in the USER2.TEST table of the project workspace and its descendent workspaces. (The EXECUTE statement is actually on a single line.)

```
EXECUTE DBMS_WM.PurgeTable('user2.test', where_clause=>'id=20',
workspace=>'project', purgeAfter=>TRUE);
```

## 4.60 RecoverAllMigratingTables

Attempts to complete the migration process on all tables that were left in an inconsistent state after the Workspace Manager migration procedure failed.



## Syntax

```
DBMS_WM.RecoverAllMigratingTables(
    ignore_last_error IN BOOLEAN DEFAULT FALSE);
```

## Parameters

**Table 4-52 RecoverAllMigratingTables Procedure Parameters**

Parameter	Description
<code>ignore_last_error</code>	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE ignores the last error, if any, that occurred during the migration process. Information about the last error is stored in the <a href="#">USER_WM_VT_ERRORS</a> and <a href="#">ALL_WM_VT_ERRORS</a> static data dictionary views, which are described in <a href="#">Workspace Manager Static Data Dictionary Views</a> . See the Usage Notes for more information.</p> <p>FALSE (the default) does not ignore the last error, if any, that occurred during the migration process.</p>

## Usage Notes

If an error occurs while upgrading (migrating) to the current Workspace Manager release, one or more version-enabled tables can be left in an inconsistent state. If the upgrade procedure fails, you should try to fix the cause of the error (examine the [USER\\_WM\\_VT\\_ERRORS](#) or [ALL\\_WM\\_VT\\_ERRORS](#) metadata view to see the SQL statement and error message), and then call the [RecoverMigratingTable](#) procedure (for a single table) or [RecoverAllMigratingTables](#) procedure (for all tables) with the default `ignore_last_error` parameter value of `FALSE`, to try to complete the upgrade process.

However, if the call still fails and you cannot fix the cause of the error, and if you are sure that it is safe and appropriate to ignore this error, then you have the option to ignore the error by calling the [RecoverMigratingTable](#) or [RecoverAllMigratingTables](#) procedure with the `ignore_last_error` parameter value of `TRUE`. Note that you are responsible for ensuring that it is safe and appropriate to ignore the error.

## Examples

The following example attempts to recover all version-enabled tables that were left in an inconsistent state when the upgrade procedure failed.

```
EXECUTE DBMS_WM.RecoverAllMigratingTables;
```

The following example attempts to recover all version-enabled tables that were left in an inconsistent state when the upgrade procedure failed, and it ignores the last error that caused the upgrade procedure to fail.

```
EXECUTE DBMS_WM.RecoverAllMigratingTables(TRUE);
```

## 4.61 RecoverFromDroppedUser

Performs necessary operations after the dropping of one or more database users that owned one or more version-enabled tables.

## Syntax

```
DBMS_WM.RecoverFromDroppedUser (  
    ignore_last_error IN BOOLEAN DEFAULT FALSE);
```

## Parameters

**Table 4-53 RecoverFromDroppedUser Procedure Parameters**

Parameter	Description
ignore_last_error	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE ignores the last error, if any, that occurred during the previous call to the RecoverFromDroppedUser procedure. Information about the last error is stored in the <a href="#">USER_WM_VT_ERRORS</a> and <a href="#">ALL_WM_VT_ERRORS</a> static data dictionary views, which are described in <a href="#">Workspace Manager Static Data Dictionary Views</a> . See the Usage Notes for more information.</p> <p>FALSE (the default) does not ignore the last error, if any, that occurred during the previous call to the RecoverFromDroppedUser procedure.</p>

## Usage Notes

If a database user with one or more version-enabled tables is dropped, you must execute this procedure as soon as possible. This procedure removes any foreign key constraints in existing tables that depended on any of the version-enabled tables that were dropped as a result of dropping the user that owned these tables. This procedure also fixes any invalid database metadata.

If a call to the RecoverFromDroppedUser procedure fails, the table is left in an inconsistent state. If this occurs, you should try to fix the cause of the error (examine the [DBA\\_WM\\_VT\\_ERRORS](#) metadata view to see the SQL statement and error message), and then call the RecoverFromDroppedUser procedure again with the default `ignore_last_error` parameter value of `FALSE`. However, if the call still fails and you cannot fix the cause of the error, and if you are sure that it is safe and appropriate to ignore this error, then you have the option to ignore the error by calling the RecoverFromDroppedUser procedure with the `ignore_last_error` parameter value of `TRUE`. Note that you are responsible for ensuring that it is safe and appropriate to ignore the error.

To execute this procedure, you must connect to the database instance as a user with SYSDBA privileges.

## Examples

The following drops a user named `HERMAN` that owns one or more version-enabled tables, and then performs the necessary operations after the drop operation.

```
DROP USER herman CASCADE;  
EXECUTE DBMS_WM.RecoverFromDroppedUser;
```

## 4.62 RecoverMigratingTable

Attempts to complete the migration process on a table that was left in an inconsistent state after the Workspace Manager migration procedure failed.

### Syntax

```
DBMS_WM.RecoverMigratingTable(
    table_name          IN VARCHAR2,
    ignore_last_error  IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 4-54 RecoverMigratingTable Procedure Parameters**

Parameter	Description
table_name	Name of the version-enabled table to be recovered from the migration error. The name is not case-sensitive.
ignore_last_error	A Boolean value (TRUE or FALSE). TRUE ignores the last error, if any, that occurred during the migration process. Information about the last error is stored in the <a href="#">USER_WM_VT_ERRORS</a> and <a href="#">ALL_WM_VT_ERRORS</a> static data dictionary views, which are described in <a href="#">Workspace Manager Static Data Dictionary Views</a> . See the Usage Notes for more information. FALSE (the default) does not ignore the last error, if any, that occurred during the migration process.

### Usage Notes

If an error occurs while upgrading to the current Workspace Manager release, one or more version-enabled tables can be left in an inconsistent state. If the upgrade procedure fails, you should try to fix the cause of the error (examine the [USER\\_WM\\_VT\\_ERRORS](#) or [ALL\\_WM\\_VT\\_ERRORS](#) metadata view to see the SQL statement and error message), and then call the [RecoverMigratingTable](#) procedure (for a single table) or [RecoverAllMigratingTables](#) procedure (for all tables) with the default `ignore_last_error` parameter value of `FALSE`, to try to complete the upgrade process.

However, if the call still fails and you cannot fix the cause of the error, and if you are sure that it is safe and appropriate to ignore this error, then you have the option to ignore the error by calling the [RecoverMigratingTable](#) or [RecoverAllMigratingTables](#) procedure with the `ignore_last_error` parameter value of `TRUE`. Note that you are responsible for ensuring that it is safe and appropriate to ignore the error.

An exception is raised if `table_name` does not exist or is not version-enabled.

### Examples

The following example attempts to recover the `COLA_MARKETING_BUDGET` table from the error that caused the upgrade procedure to fail.

```
EXECUTE DBMS_WM.RecoverMigratingTable('COLA_MARKETING_BUDGET');
```

The following example attempts to recover the `COLA_MARKETING_BUDGET` table and ignores the last error that caused the upgrade procedure to fail.

```
EXECUTE DBMS_WM.RecoverMigratingTable('COLA_MARKETING_BUDGET', TRUE);
```

## 4.63 RefreshTable

Applies to a workspace all changes made to a table (all rows or as specified in the `WHERE` clause) in its parent workspace.

For a multiparent workspace (explained in [Multiparent Workspaces](#)), applies changes from the non-leaf workspaces in the directed acyclic graph to the specified leaf workspace for a specified table. (The table data in the intermediate workspaces is not changed.)

### Syntax

```
DBMS_WM.RefreshTable(
    workspace      IN VARCHAR2,
    table_id       IN VARCHAR2,
    where_clause   IN VARCHAR2 DEFAULT '',
    auto_commit    IN BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 4-55 RefreshTable Procedure Parameters**

Parameter	Description
<code>workspace</code>	Name of the workspace. The name is case-sensitive.
<code>table_id</code>	Name of the table containing the rows to be refreshed using values from the parent workspace. The name is not case-sensitive.
<code>where_clause</code>	<p>The <code>WHERE</code> clause (excluding the <code>WHERE</code> keyword) identifying the rows to be refreshed from the parent workspace. Example: <code>'department_id = 20'</code></p> <p>Only primary key columns can be specified in the <code>WHERE</code> clause, except in a subquery. The subquery can refer to columns that are not primary keys, but it cannot refer to a version-enabled table.</p> <p>If the <code>where_clause</code> parameter is not specified, all rows in <code>table_name</code> are refreshed.</p>
<code>auto_commit</code>	<p>A Boolean value (<code>TRUE</code> or <code>FALSE</code>).</p> <p><code>TRUE</code> (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes.</p> <p><code>FALSE</code> causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a>.</p>

### Usage Notes

This procedure applies to `workspace` all changes in rows that satisfy the `where_clause` parameter value in the version-enabled table named `table_id` in the parent workspace since the time when `workspace` was created or last refreshed.

If there are conflicts between the workspace being refreshed and its parent workspace, the refresh operation fails and the user must manually resolve conflicts using the <table\_name>\_CONF view. (Conflict resolution is explained in [Resolving Conflicts Before a Merge or Refresh Operation](#).)

This procedure is ignored if `workspace` is a continually refreshed workspace.

A table cannot be refreshed in the `LIVE` workspace (because that workspace has no parent workspace).

An exception is raised if the user does not have access to `table_id`, if the user does not have the `MERGE_WORKSPACE` privilege for `workspace` or the `MERGE_ANY_WORKSPACE` privilege, or if `auto_commit` is `TRUE` and an open transaction exists in a parent or child workspace of any table that needs to be modified.

### Examples

The following example refreshes `NEWWORKSPACE` by applying changes made to the `EMPLOYEES` table where `last_name = 'Smith'` in its parent workspace.

```
EXECUTE DBMS_WM.RefreshTable ('NEWWORKSPACE', 'employees', 'last_name = ''Smith'');
```

## 4.64 RefreshWorkspace

Applies to a workspace all changes made in its parent workspace.

For a multiparent workspace (explained in [Multiparent Workspaces](#)), applies changes from the non-leaf workspaces in the directed acyclic graph to the specified leaf workspace. The changes are propagated beginning with the multiparent root workspace and continuing with the intermediate workspaces.

### Syntax

```
DBMS_WM.RefreshWorkspace(
    workspace    IN VARCHAR2,
    auto_commit  IN BOOLEAN DEFAULT TRUE,
    copy_data    IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 4-56 RefreshWorkspace Procedure Parameters**

Parameter	Description
<code>workspace</code>	Name of the workspace. The name is case-sensitive.
<code>auto_commit</code>	A Boolean value ( <code>TRUE</code> or <code>FALSE</code> ). <code>TRUE</code> (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. <code>FALSE</code> causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a> .

**Table 4-56 (Cont.) RefreshWorkspace Procedure Parameters**

Parameter	Description
copy_data	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE causes all changes in the parent workspace since the creation or last refresh of the child workspace to be copied to the child workspace. No changes occur in any descendent of the child workspace, and the history of changes to the child workspace is preserved.</p> <p>FALSE (the default) causes minimal data to be copied to the child workspace. The parent version of the child workspace is updated in order for the child workspace and its descendents to have access to the modified rows from the parent workspace. No history of changes to the child workspace is recorded for the operation.</p>

### Usage Notes

This procedure applies to `workspace` all changes made to version-enabled tables in the parent workspace since the time when `workspace` was created or last refreshed.

If there are conflicts between the workspace being refreshed and its parent workspace, the refresh operation fails and the user must manually resolve conflicts using the `<table_name>_CONF` view. (Conflict resolution is explained in [Resolving Conflicts Before a Merge or Refresh Operation](#).)

The specified workspace and the parent workspace are frozen in `READ_ONLY` mode, as explained in [Freezing and Unfreezing Workspaces](#).

The `LIVE` workspace cannot be refreshed (because it has no parent workspace).

This procedure is ignored if `workspace` is a continually refreshed workspace.

An exception is raised if the user does not have the `MERGE_WORKSPACE` privilege for `workspace` or the `MERGE_ANY_WORKSPACE` privilege, if the user does not have sufficient privileges on all tables that need to be modified (including, for example, tables modified by triggers), or if `auto_commit` is `TRUE` and an open transaction exists in a parent or child workspace of any table that needs to be modified.

### Examples

The following example refreshes `NEWWORKSPACE` by applying changes made in its parent workspace.

```
EXECUTE DBMS_WM.RefreshWorkspace ('NEWWORKSPACE');
```

## 4.65 RemoveAsParentWorkspace

Removes a workspace as a parent workspace in a multiparent workspace environment.

### Syntax

```
DBMS_WM.RemoveAsParentWorkspace(
  mp_leafworkspace IN VARCHAR2,
  parent_workspace IN VARCHAR2,
  auto_commit      IN BOOLEAN DEFAULT TRUE);
```

## Parameters

**Table 4-57 RemoveAsParentWorkspace Procedure Parameters**

Parameter	Description
<code>mp_leaf_workspace</code>	Name of the child workspace (multiparent leaf workspace) from which to remove <code>parent_workspace</code> as a parent workspace. The name is case-sensitive.
<code>parent_workspace</code>	Name of the workspace to remove as a parent workspace of <code>mp_leaf_workspace</code> . The name is case-sensitive.
<code>auto_commit</code>	A Boolean value (TRUE or FALSE).  TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes.  FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a> .

## Usage Notes

This procedure is part of the support for the multiparent workspaces feature, which is described in [Multiparent Workspaces](#). This procedure must be used only on a parent workspace that was previously added to the child workspace using the [AddAsParentWorkspace](#) procedure.

This procedure does not remove any workspaces. It only makes `parent_workspace` no longer a parent workspace of `mp_leaf_workspace`.

An exception is raised if one or more of the following apply:

- `mp_leaf_workspace` or `parent_workspace` does not exist.
- `mp_leaf_workspace` has versioned any data in `parent_workspace` or an ancestor of `parent_workspace`, and this workspace would no longer be an ancestor of `mp_leaf_workspace` if the operation were to be performed.
- There are any sessions with open database transactions in `mp_leaf_workspace`.
- `auto_commit` is TRUE and an open transaction exists in a parent or child workspace of any table that needs to be modified.

## Examples

The following example removes `Workspace4` as a parent workspace of `Workspace3`. (See the hierarchy illustration in [Multiparent Workspaces](#).)

```
EXECUTE DBMS_WM.RemoveAsParentWorkspace ('Workspace3', 'Workspace4');
```

## 4.66 RemoveDeferredWorkspaces

Removes the rows and locks from any version enabled tables associated with workspaces that were removed by specifying either `FAST` or `REMOVE_LOCKS` for the `defer_option` parameter of the `RemoveWorkspace` or `RemoveWorkspaceTree` procedure.

### Syntax

```
DBMS_WM.RemoveDeferredWorkspaces (
    auto_commit IN BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 4-58 RemoveWorkspace Procedure Parameters**

Parameter	Description
<code>auto_commit</code>	<p>A Boolean value (<code>TRUE</code> or <code>FALSE</code>).</p> <p><code>TRUE</code> (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes.</p> <p><code>FALSE</code> causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a>.</p>

### Usage Notes

After a workspace has been removed using [RemoveWorkspace](#) or [RemoveWorkspaceTree](#) and specifying `FAST` or `REMOVE_LOCKS` for the `defer_option` parameter, the only way to remove the rows associated with that workspace is to execute this procedure. In addition, if `FAST` was used, any locks that still remain will be released.

This procedure removes the rows and any remaining metadata for all workspaces that had deferred removal. It is not possible to specify a subset of the deferred workspaces to be removed. Any workspaces that need to be removed after being deferred will appear in the `DBA_WORKSPACES` view with a `DEFERRED_REMOVAL` value for the `FREEZE_MODE` column. Until a workspace in this state is removed, it is not possible to create a new workspace using the same name.

`WM_ADMIN` privileges are necessary to execute this procedure.

### Examples

The following example removes the rows and locks from any version enabled tables associated with workspaces that were removed by specifying either `FAST` or `REMOVE_LOCKS` for the `defer_option` parameter of the [RemoveWorkspace](#) or [RemoveWorkspaceTree](#) procedure.

```
EXECUTE DBMS_WM.RemoveDeferredWorkspaces;
```



## 4.67 RemoveUserDefinedHint

Removes a user-defined hint: that is, causes the default optimizer hint to be used with SQL statements executed by the DBMS\_WM package on a specified version-enabled table or all version-enabled tables.

### Syntax

```
DBMS_WM.RemoveUserDefinedHint(
    hint_id    IN NUMBER,
    table_id   IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 4-59 RemoveUserDefinedHint Procedure Parameters**

Parameter	Description
hint_id	Numeric ID that uniquely identifies the user-defined hint. Must match an existing hint ID previously specified in a call to the <a href="#">AddUserDefinedHint</a> procedure.
table_id	Name of the table from which to remove the hint. The name is not case-sensitive. If this value is null and if the table_id parameter was null in the call to the <a href="#">AddUserDefinedHint</a> procedure that added the hint, the hint is no longer used with any version-enabled tables for any SQL statements that specify the hint ID. However, if this value is null and if the table_id parameter was <i>not</i> null in the call to the <a href="#">AddUserDefinedHint</a> procedure that added the hint, the <a href="#">RemoveUserDefinedHint</a> procedure will not remove any hints that were defined for the originally specified tables.

### Usage Notes

Use this procedure only to remove or modify the effect of a user-defined hint that you previously specified using the [AddUserDefinedHint](#) procedure. (See the Usage Notes for that procedure.)

### Examples

The following example removes, for the SCOTT.TABLE1 table, the user-defined hint from SQL statements associated with the hint with the hint ID 1101, and causes the default hint to be used instead.

```
EXECUTE DBMS_WM.RemoveUserDefinedHint (1101, 'scott.table1');
```

## 4.68 RemoveWorkspace

Discards all row versions associated with a workspace and deletes the workspace.

### Syntax

```
DBMS_WM.RemoveWorkspace(
    workspace    IN VARCHAR2,
    auto_commit  IN BOOLEAN DEFAULT TRUE,
    defer_option IN VARCHAR2 DEFAULT NULL);
```

## Parameters

**Table 4-60 RemoveWorkspace Procedure Parameters**

Parameter	Description
workspace	Name of the workspace. The name is case-sensitive.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a> .
defer_option	Specifies whether rows and locks associated with the workspace for version-enabled tables are removed or if the removal is to be deferred until later. This allows you to accept or override the value of the REMOVEWORKSPACE_DEFERRED system parameter. The value can be null, OFF, FAST, or REMOVE_LOCKS. For an explanation of the REMOVEWORKSPACE_DEFERRED system parameter and its possible values, see <a href="#">System Parameters for Workspace Manager</a> .

## Usage Notes

The RemoveWorkspace operation can only be performed on leaf workspaces (the bottom-most workspaces in a branch in the hierarchy). For an explanation of database workspace hierarchy, see [Workspace Hierarchy](#).

If the workspace being removed is a child workspace, its parent workspace is exclusively locked for the duration of the operation.

There must be no other users in the workspace being removed.

An exception is raised if the user does not have the REMOVE\_WORKSPACE privilege for workspace or the REMOVE\_ANY\_WORKSPACE privilege, if the user does not have sufficient privileges on all tables that need to be modified (including, for example, tables modified by triggers), or if auto\_commit is TRUE and an open transaction exists in a parent or child workspace of any table that needs to be modified.

## Examples

The following example removes the NEWWORKSPACE workspace.

```
EXECUTE DBMS_WM.RemoveWorkspace('NEWWORKSPACE');
```

# 4.69 RemoveWorkspaceTree

Discards all row versions associated with a workspace and its descendant workspaces, and deletes the affected workspaces.

## Syntax

```
DBMS_WM.RemoveWorkspaceTree (
    workspace    IN VARCHAR2,
    auto_commit  IN BOOLEAN DEFAULT TRUE,
    defer_option IN VARCHAR2 DEFAULT NULL;
```

## Parameters

**Table 4-61 RemoveWorkspaceTree Procedure Parameters**

Parameter	Description
workspace	Name of the workspace. The name is case-sensitive.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a> .
defer_option	Specifies whether rows and locks associated with the workspace for version-enabled tables are removed or if the removal is to be deferred until later. This allows you to accept or override the value of the REMOVEWORKSPACE_DEFERRED system parameter. The value can be null, OFF, FAST, or REMOVE_LOCKS. For an explanation of the REMOVEWORKSPACE_DEFERRED system parameter and its possible values, see <a href="#">System Parameters for Workspace Manager</a> .

## Usage Notes

The RemoveWorkspaceTree operation should be used with extreme caution, because it removes support structures and rolls back changes in a workspace and all its descendants down to the leaf workspace or workspaces. For example, in the hierarchy shown in [Workspace Hierarchy](#), a RemoveWorkspaceTree operation specifying `Workspace1` removes `Workspace1`, `Workspace2`, and `Workspace3`. (For an explanation of database workspace hierarchy, see [Workspace Hierarchy](#).)

There must be no other users in `workspace` or any of its descendant workspaces.

An exception is raised if the user does not have the `REMOVE_WORKSPACE` privilege for `workspace` or any of its descendant workspaces, if the user does not have sufficient privileges on all tables that need to be modified (including, for example, tables modified by triggers), or if `auto_commit` is `TRUE` and an open transaction exists in a parent or child workspace of any table that needs to be modified.

## Examples

The following example removes the `NEWWORKSPACE` workspace and all its descendant workspaces.

```
EXECUTE DBMS_WM.RemoveWorkspaceTree('NEWWORKSPACE');
```

## 4.70 RenameSavepoint

Renames a savepoint in a specified workspace.

### Syntax

```
DBMS_WM.RenameSavepoint(  
    workspace_name    IN VARCHAR2,  
    savepoint_name    IN VARCHAR2;  
    new_savepoint_name IN VARCHAR2;
```

### Parameters

**Table 4-62** RenameSavepoint Procedure Parameters

Parameter	Description
workspace_name	Name of the existing workspace in which the savepoint to be renamed exists. The name is case-sensitive.
savepoint_name	Name of the existing explicit savepoint to be renamed. (Must not be an implicit savepoint.)
new_savepoint_name	New name to be given to the savepoint. Must not be the name of an existing savepoint.

### Usage Notes

An exception is raised if the user does not own the workspace or savepoint or does not have the `WM_ADMIN` system privilege.

### Examples

The following example renames savepoint `SP1` in the `LIVE` workspace to `2009 milestone`.

```
EXECUTE DBMS_WM.RenameSavepoint('LIVE', 'SP1', '2009 milestone');
```

## 4.71 RenameWorkspace

Renames a workspace.

### Syntax

```
DBMS_WM.RenameWorkspace(  
    workspace_name    IN VARCHAR2,  
    new_workspace_name IN VARCHAR2;
```

## Parameters

**Table 4-63 RenameWorkspace Procedure Parameters**

Parameter	Description
workspace_name	Name of the existing workspace to be renamed. The name is case-sensitive.
new_workspace_name	New name to be given to the workspace. The new name must not be LIVE or the name of an existing workspace, and it must not contain any of the following characters: " (double quotes), ' (single quote), ` (grave accent), or   (vertical bar).

## Usage Notes

This procedure automatically updates the metadata for existing version-enabled tables to refer to the new workspace name. The time required for the procedure to complete will depend on the number of version-enabled tables.

An exception is raised if the user does not own the workspace or does not have the WM\_ADMIN system privilege.

## Examples

The following example renames workspace WS1 to Construction Project.

```
EXECUTE DBMS_WM.RenameWorkspace('WS1', 'Construction Project');
```

# 4.72 ResolveConflicts

Resolves conflicts between workspaces.

## Syntax

```
DBMS_WM.ResolveConflicts(
  workspace      IN VARCHAR2,
  table_name     IN VARCHAR2,
  where_clause   IN VARCHAR2,
  keep           IN VARCHAR2,
  resolve_base_ne IN BOOLEAN DEFAULT FALSE);
```

## Parameters

**Table 4-64 ResolveConflicts Procedure Parameters**

Parameter	Description
workspace	Name of the workspace to check for conflicts with other workspaces. The name is case-sensitive.
table_name	Name of the table to check for conflicts. The name is not case-sensitive.

**Table 4-64 (Cont.) ResolveConflicts Procedure Parameters**

Parameter	Description
<code>where_clause</code>	<p>The <code>WHERE</code> clause (excluding the <code>WHERE</code> keyword) identifying the rows to be refreshed from the parent workspace. Example: <code>'department_id = 20'</code></p> <p>Only primary key columns can be specified in the <code>WHERE</code> clause, except in a subquery. The subquery can refer to columns that are not primary keys, but it cannot refer to a version-enabled table.</p>
<code>keep</code>	<p>Workspace in favor of which to resolve conflicts: <code>PARENT</code>, <code>CHILD</code>, or <code>BASE</code>.</p> <p><code>PARENT</code> causes the parent workspace rows to be copied to the child workspace.</p> <p><code>CHILD</code> does not cause the child workspace rows to be copied immediately to the parent workspace. However, the conflict is considered resolved, and the child workspace rows are copied to the parent workspace when the child workspace is merged.</p> <p><code>BASE</code> causes the base rows to be copied to the child workspace but not to the parent workspace. However, the conflict is considered resolved; and when the child workspace is merged, the base rows are copied to the parent workspace. Note that <code>BASE</code> is ignored for insert-insert conflicts where a base row does not exist; in this case, the <code>keep</code> parameter value must be <code>PARENT</code> or <code>CHILD</code>.</p>
<code>resolve_bbase_ne</code>	<p>A Boolean value (<code>TRUE</code> or <code>FALSE</code>). Applies only if the value of <code>keep</code> is <code>BASE</code>.</p> <p><code>TRUE</code> allows the resolution of conflicts in favor of the base row, even when the base row was nonexistent, as is the case in an insert-insert conflict.</p> <p><code>FALSE</code> (the default) disallows the resolution of conflicts in favor of nonexistent base rows. If a nonexistent base row is encountered while other conflicts are being resolved, the conflict is effectively ignored and will not be considered resolved.</p>

### Usage Notes

This procedure checks the condition identified by the `table_name` and `where_clause` parameters, and it finds any conflicts between row values in `workspace` and its parent workspace. This procedure resolves conflicts by using the row values in the parent or child workspace, as specified in the `keep` parameter; however, the conflict resolution is not actually merged until you commit the transaction (standard database commit operation) and call the [CommitResolve](#) procedure to end the conflict resolution session. (For more information about conflict resolution, including an overall view of the process, see [Resolving Conflicts Before a Merge or Refresh Operation](#).)

For example, assume that for Department 20 (`DEPARTMENT_ID = 20`), the `MANAGER_NAME` in the `LIVE` and `Workspace1` workspaces is `Tom`. Then, the following operations occur:

1. The `manager_name` for Department 20 is changed in the `LIVE` database workspace from `Tom` to `Mary`.
2. The change is committed (a standard database commit operation).
3. The `manager_name` for Department 20 is changed in `Workspace1` from `Tom` to `Franco`.
4. The [MergeWorkspace](#) procedure is called to merge `Workspace1` changes to the `LIVE` workspace.

At this point, however, a conflict exists with respect to `MANAGER_NAME` for Department 20 in `Workspace1` (Franco, which conflicts with Mary in the `LIVE` workspace), and therefore the call to [MergeWorkspace](#) does not succeed.

5. The `ResolveConflicts` procedure is called with the following parameters: ('Workspace1', 'department', 'department\_id = 20', 'child').

After the [MergeWorkspace](#) operation in step 7, the `MANAGER_NAME` value will be Franco in both the `Workspace1` and `LIVE` workspaces.

6. The change is committed (a standard database commit operation).
7. The [MergeWorkspace](#) procedure is called to merge `Workspace1` changes to the `LIVE` workspace.

The following considerations apply during a conflict resolution session:

- A `ResolveConflicts` operation prevents other workspace operations (such as a merge, refresh, or removal) on the target workspace or table until after the [CommitResolve](#) or [RollbackResolve](#) procedure is executed.
- Multiple sessions can perform `ResolveConflicts` operations and perform insert, update, and delete operations on the same table. However, during such operations, the target rows are locked. If more than one session attempts to perform an insert, update, or delete operation on the same row or to resolve a conflict affecting the same row, the first session is allowed to continue; and after that session executes the [CommitResolve](#) or [RollbackResolve](#) procedure, another session is allowed to proceed.

For more information about conflict resolution, see [Resolving Conflicts Before a Merge or Refresh Operation](#).

### Examples

The following example resolves conflicts involving rows in the `DEPARTMENT` table in `Workspace1` where `DEPARTMENT_ID` is 20, and uses the values in the child workspace to resolve all such conflicts. It then merges the results of the conflict resolution by first committing the transaction (standard commit) and then calling the [MergeWorkspace](#) procedure.

```
EXECUTE DBMS_WM.BeginResolve ('Workspace1');
EXECUTE DBMS_WM.ResolveConflicts ('Workspace1', 'department', 'department_id = 20',
'child');
COMMIT;
EXECUTE DBMS_WM.CommitResolve ('Workspace1');
```

## 4.73 RevokeGraphPriv

Revokes (removes) privileges on multiparent graph workspaces from users and roles for a specified leaf workspace.

### Syntax

```
DBMS_WM.RevokeGraphPriv(
    priv_types      IN VARCHAR2,
    leaf_workspace  IN VARCHAR2,
    grantee         IN VARCHAR2,
    node_types      IN VARCHAR2 DEFAULT (('R','I','L')),
    auto_commit     IN BOOLEAN DEFAULT TRUE);
```

## Parameters

**Table 4-65 RevokeGraphPriv Procedure Parameters**

Parameter	Description
<code>priv_types</code>	A string of one or more keywords representing privileges. ( <a href="#">Privilege Management with Workspace Manager</a> discusses Workspace Manager privileges.) Use commas to separate privilege keywords. The available keywords are <code>ACCESS_WORKSPACE</code> , <code>MERGE_WORKSPACE</code> , <code>CREATE_WORKSPACE</code> , <code>REMOVE_WORKSPACE</code> , and <code>ROLLBACK_WORKSPACE</code> .
<code>leaf_workspace</code>	Name of the leaf workspace in the directed acyclic graph. (Leaf workspaces, directed acyclic graphs, and other concepts related to multiparent workspaces are explained in <a href="#">Multiparent Workspaces</a> .) The name is case-sensitive.
<code>grantee</code>	Name of the user (can be the <code>PUBLIC</code> user group) or role from which to revoke <code>priv_types</code> .
<code>node_types</code>	List of letters (in parentheses and comma-delimited) representing the types of nodes on which to revoke the privileges: <code>R</code> for the root of the graph, <code>I</code> for the specified intermediate node, <code>L</code> for the leaf of the graph. The default is all types of nodes.
<code>auto_commit</code>	A Boolean value ( <code>TRUE</code> or <code>FALSE</code> ).  <code>TRUE</code> (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes.  <code>FALSE</code> causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a> .

## Usage Notes

Contrast this procedure with [RevokeWorkspacePriv](#), which grants workspace-level Workspace Manager privileges on workspaces other than multiparent graph workspaces.

To grant workspace-level privileges on multiparent graph workspaces, use the [GrantGraphPriv](#) procedure.

An exception is raised if one or more of the following apply:

- `grantee` is not a valid user or role in the database.
- You were not the grantor of `priv_types` to `grantee`.
- `auto_commit` is `TRUE` and an open transaction exists in a parent or child workspace of any table that needs to be modified.

## Examples

The following example disallows user `Smith` from accessing all types of nodes in the directed acyclic graph in which the `NEWWORKSPACE` workspace is the leaf workspace and from merging changes in these workspaces.



```
EXECUTE DBMS_WM.RevokeWorkspacePriv ('ACCESS_WORKSPACE, MERGE_WORKSPACE',
'NEWWORKSPACE', 'Smith');
```

## 4.74 RevokeSystemPriv

Revokes (removes) system-level privileges from users and roles.

### Syntax

```
DBMS_WM.RevokeSystemPriv(
    priv_types    IN VARCHAR2,
    grantee       IN VARCHAR2,
    auto_commit   IN BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 4-66 RevokeSystemPriv Procedure Parameters**

Parameter	Description
priv_types	A string of one or more keywords representing privileges. ( <a href="#">Privilege Management with Workspace Manager</a> discusses Workspace Manager privileges.) Use commas to separate privilege keywords. The available keywords are ACCESS_ANY_WORKSPACE, MERGE_ANY_WORKSPACE, CREATE_ANY_WORKSPACE, REMOVE_ANY_WORKSPACE, and ROLLBACK_ANY_WORKSPACE.
grantee	Name of the user (can be the PUBLIC user group) or role from which to revoke priv_types.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a> .

### Usage Notes

Contrast this procedure with [RevokeWorkspacePriv](#), which revokes workspace-level Workspace Manager privileges with keywords in the form xxx\_WORKSPACE (ACCESS\_WORKSPACE, MERGE\_WORKSPACE, and so on).

To grant system-level privileges, use the [GrantSystemPriv](#) procedure.

An exception is raised if one or more of the following apply:

- grantee is not a valid user or role in the database.
- You were not the grantor of priv\_types to grantee.
- auto\_commit is TRUE and an open transaction exists in a parent or child workspace of any table that needs to be modified.

### Examples

The following example disallows user Smith from accessing workspaces and merging changes in workspaces.

```
EXECUTE DBMS_WM.RevokeSystemPriv ('ACCESS_ANY_WORKSPACE, MERGE_ANY_WORKSPACE',
'Smith');
```

## 4.75 RevokeWorkspacePriv

Revokes (removes) workspace-level privileges from users and roles for a specified workspace.

### Syntax

```
DBMS_WM.RevokeWorkspacePriv(
  priv_types IN VARCHAR2,
  workspace IN VARCHAR2,
  grantee IN VARCHAR2,
  auto_commit IN BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 4-67 RevokeWorkspacePriv Procedure Parameters**

Parameter	Description
priv_types	A string of one or more keywords representing privileges. ( <a href="#">Privilege Management with Workspace Manager</a> discusses Workspace Manager privileges.) Use commas to separate privilege keywords. The available keywords are ACCESS_WORKSPACE, MERGE_WORKSPACE, CREATE_WORKSPACE, REMOVE_WORKSPACE, and ROLLBACK_WORKSPACE.
workspace	Name of the workspace. The name is case-sensitive.
grantee	Name of the user (can be the PUBLIC user group) or role from which to revoke priv_types.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a> .

### Usage Notes

Contrast this procedure with [RevokeSystemPriv](#), which revokes system-level Workspace Manager privileges with keywords in the form xxx\_ANY\_WORKSPACE (ACCESS\_ANY\_WORKSPACE, MERGE\_ANY\_WORKSPACE, and so on). Also contrast this procedure with [RevokeGraphPriv](#), which grants workspace-level Workspace Manager privileges on multiparent graph workspaces

To grant workspace-level privileges, use the [GrantWorkspacePriv](#) procedure.

An exception is raised if one or more of the following apply:

- grantee is not a valid user or role in the database.
- You were not the grantor of priv\_types to grantee.

- `auto_commit` is `TRUE` and an open transaction exists in a parent or child workspace of any table that needs to be modified.

### Examples

The following example disallows user `Smith` from accessing the `NEWWORKSPACE` workspace and merging changes in that workspace.

```
EXECUTE DBMS_WM.RevokeWorkspacePriv ('ACCESS_WORKSPACE, MERGE_WORKSPACE',
'NEWWORKSPACE', 'Smith');
```

## 4.76 RollbackBulkLoading

Rolls back changes made to a version-enabled table during a bulk load operation.

### Syntax

```
DBMS_WM.RollbackBulkLoading(
    table_name          IN VARCHAR2,
    ignore_last_error  IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 4-68 RollbackBulkLoading Procedure Parameters**

Parameter	Description
<code>table_name</code>	Name of the version-enabled table into which data will be bulk loaded. The name is not case-sensitive.
<code>ignore_last_error</code>	A Boolean value ( <code>TRUE</code> or <code>FALSE</code> ). <code>TRUE</code> ignores the last error, if any, that occurred during the previous call to the <code>RollbackBulkLoading</code> procedure. Information about the last error is stored in the <code>USER_WM_VT_ERRORS</code> and <code>ALL_WM_VT_ERRORS</code> static data dictionary views, which are described in <a href="#">Workspace Manager Static Data Dictionary Views</a> . See the Usage Notes for more information. <code>FALSE</code> (the default) does not ignore the last error, if any, that occurred during the previous call to the <code>RollbackBulkLoading</code> procedure.

### Usage Notes

For information about the requirements for bulk loading data into version-enabled tables, see [Bulk Loading into Version-Enabled Tables](#).

This procedure re-creates all the views that were dropped by the [BeginBulkLoading](#) procedure.

If a call to the `RollbackBulkLoading` procedure fails, you should try to fix the cause of the error. Examine the `USER_WM_VT_ERRORS` and `ALL_WM_VT_ERRORS` static data dictionary views to see the SQL statement and error message. Fix the cause of the error, and then call the `RollbackBulkLoading` procedure again with the default `ignore_last_error` parameter value of `FALSE`. However, if the call still fails and you cannot fix the cause of the error, and if you are sure that it is safe and appropriate to ignore this error, then you have the option to ignore the error by calling the `RollbackBulkLoading` procedure with the

`ignore_last_error` parameter value of `TRUE`. Note that you are responsible for ensuring that it is safe and appropriate to ignore the error.

An exception is raised if one or more of the following apply:

- `table_name` does not exist.
- `table_name` is not version-enabled.
- The [BeginBulkLoading](#) procedure has not been called on the table.
- The user does not own the table or does not have the `WM_ADMIN` system privilege.

### Examples

The following example rolls back changes made to `EMP` table during a bulk load operation.

```
EXECUTE DBMS_WM.RollbackBulkLoading ('EMP');
```

## 4.77 RollbackDDL

Rolls back (cancels) DDL (data definition language) changes made during a DDL session for a specified table, and ends the DDL session.

### Syntax

```
DBMS_WM.RollbackDDL(
    table_name IN VARCHAR2);
```

### Parameters

**Table 4-69 RollbackDDL Procedure Parameters**

Parameter	Description
<code>table_name</code>	Name of the version-enabled table. The name is not case-sensitive.

### Usage Notes

This procedure rolls back (cancels) changes that were made to a version-enabled table and to any indexes and triggers based on the version-enabled table during a DDL session. It also deletes the `<table-name>_LTS` skeleton table that was created by the [BeginDDL](#) procedure.

For detailed information about performing DDL operations related to version-enabled tables, see [DDL Operations Related to Version-Enabled Tables](#).

An exception is raised if one or more of the following apply:

- `table_name` does not exist or is not version-enabled.
- An open DDL session does not exist for `table_name`. (That is, the [BeginDDL](#) procedure has not been called specifying this table, or the [CommitDDL](#) or [RollbackDDL](#) procedure was already called specifying this table.)

## Examples

The following example begins a DDL session, adds a column named `COMMENTS` to the `COLA_MARKETING_BUDGET` table by using the skeleton table named `COLA_MARKETING_BUDGET_LTS`, and ends the DDL session by canceling the change.

```
EXECUTE DBMS_WM.BeginDDL('COLA_MARKETING_BUDGET');
ALTER TABLE cola_marketing_budget_lts ADD (comments VARCHAR2(100));
EXECUTE DBMS_WM.RollbackDDL('COLA_MARKETING_BUDGET');
```

## 4.78 RollbackResolve

Quits a conflict resolution session and discards all changes in the workspace since the [BeginResolve](#) procedure was executed.

### Syntax

```
DBMS_WM.RollbackResolve(
    workspace IN VARCHAR2);
```

### Parameters

**Table 4-70 RollbackResolve Procedure Parameters**

Parameter	Description
<code>workspace</code>	Name of the workspace. The name is case-sensitive.

### Usage Notes

This procedure quits the current conflict resolution session (started by the [BeginResolve](#) procedure), and discards all changes in the workspace since the start of the conflict resolution session. Contrast this procedure with [CommitResolve](#), which saves all changes.

While the conflict resolution session is being rolled back, the workspace is frozen in `1WRITER` mode, as explained in [Freezing and Unfreezing Workspaces](#).

For more information about conflict resolution, see [Resolving Conflicts Before a Merge or Refresh Operation](#).

An exception is raised if one or more of the following apply:

- There are one or more open database transactions in `workspace`.
- The procedure was called by a user that does not have the `WM_ADMIN` system privilege or that did not execute the [BeginResolve](#) procedure on `workspace`.

### Examples

The following example quits the conflict resolution session in `Workspace1` and discards all changes.

```
EXECUTE DBMS_WM.RollbackResolve ('Workspace1');
```

## 4.79 RollbackTable

Discards all changes made in the workspace to a specified table (all rows or as specified in the `WHERE` clause).

### Syntax

```
DBMS_WM.RollbackTable (
  workspace      IN VARCHAR2,
  table_id       IN VARCHAR2,
  sp_name        IN VARCHAR2 DEFAULT '',
  where_clause   IN VARCHAR2 DEFAULT '',
  remove_locks  IN BOOLEAN DEFAULT TRUE,
  auto_commit    IN BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 4-71 RollbackTable Procedure Parameters**

Parameter	Description
<code>workspace</code>	Name of the workspace. The name is case-sensitive.
<code>table_id</code>	Name of the table containing rows to be discarded. The name is not case-sensitive.
<code>sp_name</code>	Name of the savepoint to which to roll back. The name is case-sensitive. The default is to discard all changes (that is, ignore any savepoints).
<code>where_clause</code>	The <code>WHERE</code> clause (excluding the <code>WHERE</code> keyword) identifying the rows to be discarded. Example: <code>'department_id = 20'</code> Only primary key columns can be specified in the <code>WHERE</code> clause, except in a subquery. The subquery can refer to columns that are not primary keys, but it cannot refer to a version-enabled table. If the <code>where_clause</code> parameter is not specified, all rows that meet the criteria of the other parameters are discarded.
<code>remove_locks</code>	A Boolean value ( <code>TRUE</code> or <code>FALSE</code> ). <code>TRUE</code> (the default) releases those locks on rows in the parent workspace that satisfy the condition in the <code>where_clause</code> parameter and that were not versioned in the child workspace. This option has no effect if a savepoint is specified ( <code>sp_name</code> parameter). <code>FALSE</code> does not release any locks in the parent workspace.
<code>auto_commit</code>	A Boolean value ( <code>TRUE</code> or <code>FALSE</code> ). <code>TRUE</code> (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. <code>FALSE</code> causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a> .

## Usage Notes

You cannot roll back to a savepoint if any implicit savepoints were created since the specified savepoint, unless you first merge or remove the descendant workspaces that caused the implicit savepoints to be created. For example, referring to [Figure 1-2](#) in [Using Savepoints](#), the user in Workspace1 cannot roll back to savepoint SP1 until Workspace3 (which caused implicit savepoint SPc to be created) is merged or removed.

An exception is raised if one or more of the following apply:

- `workspace` does not exist.
- You do not have the privilege to roll back `workspace` or any affected table.
- A database transaction affecting `table_id` is open in any workspace.
- `auto_commit` is `TRUE` and an open transaction exists in a parent or child workspace of any table that needs to be modified.

## Examples

The following example rolls back all changes made to the `EMP` table (in the `USER3` schema) in the `NEWWORKSPACE` workspace since that workspace was created.

```
EXECUTE DBMS_WM.RollbackTable ('NEWWORKSPACE', 'user3.emp');
```

# 4.80 RollbackToSP

Discards all data changes made in the workspace to version-enabled tables since the specified savepoint.

## Syntax

```
DBMS_WM.RollbackToSP(
  workspace      IN VARCHAR2,
  savepoint_name IN VARCHAR2,
  auto_commit    IN BOOLEAN DEFAULT TRUE);
```

## Parameters

**Table 4-72 RollbackToSP Procedure Parameters**

Parameter	Description
<code>workspace</code>	Name of the workspace. The name is case-sensitive.
<code>savepoint_name</code>	Name of the savepoint to which to roll back changes. The name is case-sensitive.
<code>auto_commit</code>	A Boolean value ( <code>TRUE</code> or <code>FALSE</code> ). <code>TRUE</code> (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. <code>FALSE</code> causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a> .

## Usage Notes

While this procedure is executing, the workspace is frozen in `NO_ACCESS` mode.

Contrast this procedure with [RollbackWorkspace](#), which rolls back all changes made since the creation of the workspace.

You cannot roll back to a savepoint if any implicit savepoints were created since the specified savepoint, unless you first merge or remove the descendant workspaces that caused the implicit savepoints to be created. For example, referring to [Figure 1-2 in Using Savepoints](#), the user in `Workspace1` cannot roll back to savepoint `SP1` until `Workspace3` (which caused implicit savepoint `SPc` to be created) is merged or removed.

An exception is raised if one or more of the following apply:

- `workspace` does not exist.
- `savepoint_name` does not exist.
- `auto_commit` is `TRUE` and an open transaction exists in a parent or child workspace of any table that needs to be modified.
- One or more implicit savepoints were created in `workspace` after `savepoint_name`, and the descendant workspaces that caused the implicit savepoints to be created still exist.
- You do not have the privilege to roll back `workspace` or any affected table.
- Any sessions are in `workspace`.

## Examples

The following example rolls back any changes made in the `NEWWORKSPACE` workspace to all tables since the creation of `Savepoint1`.

```
EXECUTE DBMS_WM.RollbackToSP ('NEWWORKSPACE', 'Savepoint1');
```

# 4.81 RollbackWorkspace

Discards all data changes made in the workspace to version-enabled tables.

## Syntax

```
DBMS_WM.RollbackWorkspace(  
    workspace    IN VARCHAR2,  
    auto_commit  IN BOOLEAN DEFAULT TRUE);
```

## Parameters

**Table 4-73 RollbackWorkspace Procedure Parameters**

Parameter	Description
<code>workspace</code>	Name of the workspace. The name is case-sensitive.



**Table 4-73 (Cont.) RollbackWorkspace Procedure Parameters**

Parameter	Description
auto_commit	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes.</p> <p>FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a>.</p>

### Usage Notes

Only leaf workspaces can be rolled back. That is, a workspace cannot be rolled back if it has any descendant workspaces. (For an explanation of workspace hierarchy, see [Workspace Hierarchy](#).)

Contrast this procedure with [RollbackToSP](#), which rolls back changes to a specified savepoint.

Like the [RemoveWorkspace](#) procedure, RollbackWorkspace deletes the data in the workspace; however, unlike the [RemoveWorkspace](#) procedure, RollbackWorkspace does not delete the Workspace Manager workspace structure.

While this procedure is executing, the specified workspace is frozen in NO\_ACCESS mode, as explained in [Freezing and Unfreezing Workspaces](#).

An exception is raised if one or more of the following apply:

- workspace has any descendant workspaces.
- workspace does not exist.
- auto\_commit is TRUE and an open transaction exists in a parent or child workspace of any table that needs to be modified.
- You do not have the privilege to roll back workspace or any affected table.
- Any sessions are in workspace.

### Examples

The following example rolls back any changes made in the NEWWORKSPACE workspace since that workspace was created.

```
EXECUTE DBMS_WM.RollbackWorkspace ('NEWWORKSPACE');
```

## 4.82 SetCaptureEvent

Enables or disables the capture of all Workspace Manager events or events of a specific type.

## Syntax

```
DBMS_WM.SetCaptureEvent(
    event_name IN VARCHAR2,
    capture    IN VARCHAR2 DEFAULT 'ON');
```

## Parameters

**Table 4-74 SetCaptureEvent Procedure Parameters**

Parameter	Description
event_name	<p>One of the following values: ALL_EVENTS, TABLE_MERGE_W_REMOVE_DATA, TABLE_MERGE_WO_REMOVE_DATA, TABLE_REFRESH, TABLE_ROLLBACK, WORKSPACE_COMPRESS, WORKSPACE_CREATE, WORKSPACE_MERGE_W_REMOVE, WORKSPACE_MERGE_WO_REMOVE, WORKSPACE_REFRESH, WORKSPACE_REMOVE, WORKSPACE_ROLLBACK, WORKSPACE_VERSION.</p> <p>ALL_EVENTS includes all Workspace Manager events. The other values reflect specific event types, which are listed and explained in <a href="#">List of Workspace Manager Events</a>.</p>
capture	<p>ON (the default) enables the capture of event_name events.</p> <p>OFF disables the capture of event_name events.</p>

## Usage Notes

For information about Workspace Manager events, see [Workspace Manager Events](#).

This procedure requires that the Workspace Manager system parameter `ALLOW_CAPTURE_EVENTS` be set to ON. To check the value of a Workspace Manager system parameter, use the [GetSystemParameter](#) procedure; to set a Workspace Manager system parameter, use the [SetSystemParameter](#) procedure.

You can use this procedure to control which types of events are captured. For example, you can enable the capture of all events, and then disable the capture of a few types of events; or you can disable the capture of all events, and then enable the capture of a few types of events.

To see which types of events are currently being captured, examine the [WM\\_EVENTS\\_INFO](#) metadata view, which is described in [WM\\_EVENTS\\_INFO](#).

If this procedure completes successfully, it commits the caller's open database transaction.

An exception is raised if one or more of the following apply:

- You do not have the `WM_ADMIN` system privilege.
- The value of the `ALLOW_CAPTURE_EVENTS` system parameter is OFF and you are trying to set `event_name` to ON (the default value for that parameter).
- `event_name` is not valid.

## Examples

The following example captures all Workspace Manager events except workspace compression events, by first specifying that all events are to be captured, and then excluding workspace compression events.

```
-- Allow Workspace Manager events to be captured. (Required for SetCaptureEvent)
EXECUTE DBMS_WM.SetSystemParameter ('ALLOW_CAPTURE_EVENTS', 'ON');
-- Start capturing all Workspace Manager events.
EXECUTE DBMS_WM.SetCaptureEvent ('ALL_EVENTS','ON');
-- Exclude workspace compression events.
EXECUTE DBMS_WM.SetCaptureEvent ('WORKSPACE_COMPRESS','OFF');
```

## 4.83 SetCompressWorkspace

Creates rows in the [WM\\_COMPRESSIBLE\\_TABLES](#) metadata view with information about version-enabled tables that need to be compressed if workspace compression operations are performed.

### Syntax

```
DBMS_WM.SetCompressWorkspace (
    workspace IN VARCHAR2,
    firstSP   IN VARCHAR2 DEFAULT NULL,
    secondSP  IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 4-75 SetCompressWorkspace Procedure Parameters**

Parameter	Description
workspace	Name of the workspace. The name is case-sensitive.
firstSP	Savepoint on the first version of the compression range. Savepoint names are case-sensitive. If only <code>workspace</code> and <code>firstSP</code> are specified, all rows in version-enabled tables affected between workspace creation and <code>firstSP</code> are checked to see if they need to be compressed in a workspace compression operation is performed. If <code>workspace</code> , <code>firstSP</code> , and <code>secondSP</code> are specified, all rows in version-enabled tables affected between <code>firstSP</code> and <code>secondSP</code> are checked. If only <code>workspace</code> is specified (no savepoints), all rows in version-enabled tables are checked.
secondSP	Savepoint on the first version of the compression range. All rows in version-enabled tables from <code>firstSP</code> to <code>secondSP</code> are checked to see if they need to be compressed in a workspace compression operation is performed. Savepoint names are case-sensitive.

### Usage Notes

You can (but do not need to) use this procedure before calling the [CompressWorkspace](#) or [CompressWorkspaceTree](#) procedure.

This procedure creates rows in the [WM\\_COMPRESSIBLE\\_TABLES](#) metadata view (described in [WM\\_COMPRESSIBLE\\_TABLES](#)) only for version-enabled tables that would need to be compressed during a workspace compression operation.

### Examples

The following example creates rows in the [WM\\_COMPRESSIBLE\\_TABLES](#) metadata view for any version-enabled tables that would need to be compressed during an operation that compressed the `B_focus_1` workspace.

```
EXECUTE DBMS_WM.SetCompressWorkspace ('B_focus_1');
```

## 4.84 SetConflictWorkspace

Determines whether or not conflicts exist between a workspace and its parent.

### Syntax

```
DBMS_WM.SetConflictWorkspace(  
    workspace IN VARCHAR2);
```

### Parameters

**Table 4-76 SetConflictWorkspace Procedure Parameters**

Parameter	Description
workspace	Name of the workspace. The name is case-sensitive.

### Usage Notes

This procedure checks for any conflicts between `workspace` and its parent workspace, and it modifies the content of the `<table_name>_CONF` views (explained in [xxx\\_CONF Views](#)) as needed.

A `SELECT` operation from the `<table_name>_CONF` views for all tables modified in a workspace displays all rows in the workspace that are in conflict with the parent workspace. (To obtain a list of tables that have conflicts for the current conflict workspace setting, use the SQL statement `SELECT * FROM ALL_WM_VERSIONED_TABLES WHERE conflict = 'YES'`;. The SQL statement `SELECT * FROM <table_name>_CONF` displays conflicts for `<table_name>` between the current workspace and its parent workspace.)

Any conflicts must be resolved before a workspace can be merged or refreshed. To resolve a conflict, you must use the [ResolveConflicts](#) procedure, and then merge the result of the resolution by using the [MergeWorkspace](#) procedure.

### Examples

The following example checks for any conflicts between `B_focus_2` and its parent workspace, and modifies the contents of the `<table_name>_CONF` views as needed.

```
EXECUTE DBMS_WM.SetConflictWorkspace ('B_focus_2');
```

## 4.85 SetDiffVersions

Finds differences in values in version-enabled tables for two savepoints and their common ancestor (base). It modifies the contents of the differences views that describe these differences.

### Syntax

```
DBMS_WM.SetDiffVersions(
  workspace1  IN VARCHAR2,
  workspace2  IN VARCHAR2,
  onlyModified IN BOOLEAN DEFAULT FALSE);
```

or

```
DBMS_WM.SetDiffVersions(
  workspace1  IN VARCHAR2,
  savepoint1  IN VARCHAR2,
  workspace2  IN VARCHAR2,
  savepoint2  IN VARCHAR2,
  onlyModified IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 4-77 SetDiffVersions Procedure Parameters**

Parameter	Description
workspace1	Name of the first workspace to be checked for differences in version-enabled tables. The name is case-sensitive.
savepoint1	Name of the savepoint in workspace1 for which values are to be checked. The name is case-sensitive.  If savepoint1 and savepoint2 are not specified, the rows in version-enabled tables for the LATEST savepoint in each workspace are checked. (The LATEST savepoint is explained in <a href="#">Using Savepoints</a> .)
workspace2	Name of the second workspace to be checked for differences in version-enabled tables. The name is case-sensitive.
savepoint2	Name of the savepoint in workspace2 for which values are to be checked. The name is case-sensitive.
onlyModified	A Boolean value (TRUE or FALSE).  TRUE removes from the _DIFF any rows that have a NC (no change) or NE (nonexistent) value for the WM_CODE column. This improves the performance of the view when these rows are not needed. As such, each primary key can have from one through three rows, instead of the usual three.  FALSE (the default) causes queries on the _DIFF view to always return three rows for each primary key value: one for the base row, and one for each of the specified savepoints.

### Usage Notes

This procedure modifies the contents of the differences views (xxx\_DIFF), which are described in [xxx\\_DIFF Views](#). Each call to the procedure populates one or more sets of three rows, each set consisting of:

- Values for the common ancestor
- Values for workspace1 (savepoint1 or LATEST savepoint values)
- Values for workspace2 (savepoint2 or LATEST savepoint values)

You can then select rows from the appropriate xxx\_DIFF view or views to check comparable table values in the two savepoints and their common ancestor. The common ancestor (or *base*) is identified as DiffBase in xxx\_DIFF view rows.

### Examples

The following example checks the differences in version-enabled tables for the B\_focus\_1 and B\_focus\_2 workspaces. (The output has been reformatted for readability.)

```
-- Add rows to difference view: COLA_MARKETING_BUDGET_DIFF
EXECUTE DBMS_WM.SetDiffVersions ('B_focus_1', 'B_focus_2');
```

```
-- View the rows that were just added.
SELECT * from COLA_MARKETING_BUDGET_DIFF;
```

PRODUCT_ID	PRODUCT_NAME	MANAGER	BUDGET	WM_DIFFVER	WMCODE
1	cola_a	Alvarez	2	DiffBase	NC
1	cola_a	Alvarez	1.5	B_focus_1, LATEST	U
1	cola_a	Alvarez	2	B_focus_2, LATEST	NC
2	cola_b	Burton	2	DiffBase	NC
2	cola_b	Beasley	3	B_focus_1, LATEST	U
2	cola_b	Burton	2.5	B_focus_2, LATEST	U
3	cola_c	Chen	1.5	DiffBase	NC
3	cola_c	Chen	1	B_focus_1, LATEST	U
3	cola_c	Chen	1.5	B_focus_2, LATEST	NC
4	cola_d	Davis	3.5	DiffBase	NC
4	cola_d	Davis	3	B_focus_1, LATEST	U
4	cola_d	Davis	2.5	B_focus_2, LATEST	U

12 rows selected.

[xxx\\_DIFF Views](#) explains how to interpret and use the information in the differences (xxx\_DIFF) views.

## 4.86 SetLockingOFF

Disables Workspace Manager locking for the current session.

### Syntax

```
DBMS_WM.SetLockingOFF();
```

### Parameters

None.

### Usage Notes

This procedure turns off Workspace Manager locking that was set on by the [SetLockingON](#) procedure. Existing locks applied by this session remain locked. All new changes by this session are not locked.

## Examples

The following example sets locking off for the session.

```
EXECUTE DBMS_WM.SetLockingOFF;
```

## 4.87 SetLockingON

Enables Workspace Manager locking for the current session.

### Syntax

```
DBMS_WM.SetLockingON(  
    lockmode IN VARCHAR2);
```

### Parameters

**Table 4-78 SetLockingON Procedure Parameters**

Parameter	Description
lockmode	<p>Locking mode. Must be E, WE, VE, S, or C.</p> <p>E (exclusive) mode locks the rows in the previous version and the corresponding rows in the current version; no other users in the workspace for either version can change any values.</p> <p>WE (workspace-exclusive) mode locks the rows in the previous version and the corresponding rows in the current version such that only the user that set the lock can change the values in the current workspace; however, other users in other workspaces can change the values.</p> <p>VE (version-exclusive) mode locks the rows in the previous version and the corresponding rows in the current version such that only the user that set the lock can change the values; no other users (in any workspace) can change the values.</p> <p>S (shared) mode locks the rows in the previous version and the corresponding rows in the current version; however, other users in the workspace for the current version (but no users in the workspace for the previous version) can change values in these rows.</p> <p>C (carry-forward) mode locks rows in the current workspace with the same locking mode as the corresponding rows in the previous version. (If a row is not locked in the previous version, its corresponding row in the current version is not locked.)</p>

### Usage Notes

This procedure affects Workspace Manager locking, which occurs in addition to any standard Oracle database locking. Workspace Manager locks can be used to prevent conflicts. When a user locks a row, the corresponding row in the parent workspace is also locked. Thus, when this workspace merges with the parent at merge time, it is guaranteed that this row will not have a conflict.

For information about Workspace Manager lock management, see [Lock Management with Workspace Manager](#).

Exclusive locking (lockmode value of E) prevents the use of *what-if* scenarios in which different values for one or more columns are tested. Thus, plan any testing of scenarios when exclusive locking is not in effect.

Locking is enabled at the user session level, and the locking mode stays in effect until any of the following occurs:

- The session goes to another workspace or connects to the database, in which case the locking mode is set to `C` (carry-forward) unless another locking mode has been specified using the [SetWorkspaceLockModeON](#) procedure.
- The session executes the [SetLockingOFF](#) procedure.

The locks remain in effect for the duration of the workspace, unless unlocked by the [UnlockRows](#) procedure. (Existing locks are not affected by the [SetLockingOFF](#) procedure.)

There are no specific privileges associated with locking. Any session that can go to a workspace can set locking on.

### Examples

The following example sets exclusive locking on for the session.

```
EXECUTE DBMS_WM.SetLockingON ('E');
```

All rows locked by this user remain locked until the workspace is merged or rolled back.

## 4.88 SetMultiWorkspaces

Makes the specified workspace or workspaces visible in the multiworkspace views for version-enabled tables.

### Syntax

```
DBMS_WM.SetMultiWorkspaces(  
    workspaces IN VARCHAR2);
```

### Parameters

**Table 4-79 SetMultiWorkspaces Procedure Parameters**

Parameter	Description
workspaces	The workspace or workspaces for which information is to be added to the multiworkspace views (described in <a href="#">xxx_MW Views</a> ). The workspace names are case-sensitive.  To specify more than one workspace (but no more than eight), use a comma to separate workspace names. For example: <code>'workspace1,workspace2'</code>

### Usage Notes

This procedure adds rows to the multiworkspace views (`xxx_MW`). See [xxx\\_MW Views](#) for information about the contents and uses of these views.

To see the names of workspaces visible in the multiworkspace views, use the [GetMultiWorkspaces](#) function.

An exception is raised if one or more of the following apply:



- The user does not have the privilege to go to one or more of the workspaces named in `workspaces`.
- A workspace named in `workspaces` is not valid.

### Examples

The following example adds information to the multiworkspace views for version-enabled tables in the `B_focus_1` workspace.

```
EXECUTE DBMS_WM.SetMultiWorkspaces ('B_focus_1');
```

The following example shows the use of the `SetMultiWorkspaces` procedure to view information without leaving the current workspace, and the use of the [GotoWorkspace](#) procedure to view the same information.

```
-- These two pairs of statements select the same information.
EXECUTE DBMS_WM.SetMultiWorkspaces ('myworkspace');
SELECT * from mytable_mw;

EXECUTE DBMS_WM.GotoWorkspace ('myworkspace');
SELECT * from mytable;
```

To select only the rows modified in `myworkspace`, change the first `SELECT` statement in the preceding example to the following:

```
SELECT * from mytable_mw WHERE wm_modified_by = 'myworkspace';
```

The following example shows the latest rows in the combined ancestor versions of the workspaces named `myworkspace` and `yourworkspace`. If the same row is selected from more than workspace, that row is shown only once. Note that there may be more than one row for a primary key because different workspaces might be selecting different versions of the primary key.

```
EXECUTE DBMS_WM.SetMultiWorkspaces ('myworkspace,yourworkspace');
SELECT * from mytable_mw;
```

## 4.89 SetSystemParameter

Sets the value of a Workspace Manager system parameter.

### Syntax

```
DBMS_WM.SetSystemParameter(
    name    IN VARCHAR2,
    value   IN VARCHAR2);
```

### Parameters

**Table 4-80 SetSystemParameter Procedure Parameters**

Parameter	Description
<code>name</code>	Name of the Workspace Manager system parameter for which to set the value. The name must be one of the parameter names listed in the table in <a href="#">System Parameters for Workspace Manager</a> .
<code>value</code>	Value for the specified Workspace Manager system parameter, as explained in the table in <a href="#">System Parameters for Workspace Manager</a> .

## Usage Notes

For information about Workspace Manager system parameters, see [System Parameters for Workspace Manager](#).

If this procedure completes successfully, it commits the caller's open database transaction.

An exception is raised if one or more of the following apply:

- The user does not have the `WM_ADMIN` system privilege.
- The system parameter name is not valid.
- The value is not valid for the system parameter.
- You tried to disallow capturing of events, and one or more types of events were being captured. You must first disable the capturing of all events (for example, by calling the [SetCaptureEvent](#) procedure and specifying `ALL_EVENTS` for `event_type` and `OFF` for `capture`).
- You tried to disallow multiparent workspaces, and one or more multiparent workspaces already existed. You must first ensure that all workspaces have no more than one parent workspace (for example, by calling the [RemoveAsParentWorkspace](#) procedure as needed).
- You tried to disallow nested table columns, and one or more tables with a nested table column were version-enabled. You must first disable versioning on all tables with nested table columns.
- You tried to change `CR_WORKSPACE_MODE` or `NONCR_WORKSPACE_MODE` to `PESSIMISTIC_LOCKING`, and data exists in a non-LIVE workspace for the corresponding type of workspace (continually refreshed or not continually refreshed).

## Examples

The following example allows multiparent workspaces (described in [Multiparent Workspaces](#)) to be created.

```
EXECUTE DBMS_WM.SetSystemParameter ('ALLOW_MULTI_PARENT_WORKSPACES', 'ON');
```

## 4.90 SetTriggerEvents

Enables the execution of a trigger for a specified set of triggering events. The trigger will not be executed for events not specified

### Syntax

```
DBMS_WM.SetTriggerEvents(  
    triggerName    IN VARCHAR2,  
    triggerEvents  IN VARCHAR2);
```

## Parameters

**Table 4-81 SetTriggerEvents Procedure Parameters**

Parameter	Description
triggerName	Name of the trigger for which to set one or more events.
triggerEvents	A comma-delimited list of trigger event names, where each trigger event name is one of the following string constants: DBMS_WM.DML: Only for DML operations. DBMS_WM.TABLE_IMPORT: Import table (using the <a href="#">Import</a> procedure). DBMS_WM.TABLE_MERGE_W_REMOVE_DATA: Merge table and remove data. DBMS_WM.TABLE_MERGE_WO_REMOVE_DATA: Merge table without removing data. DBMS_WM.WORKSPACE_MERGE_W_REMOVE: Merge workspace and remove the workspace DBMS_WM.WORKSPACE_MERGE_WO_REMOVE: Merge workspace without removing the workspace.

## Usage Notes

For information about using triggers with Workspace Manager, see [Triggers on Version-Enabled Tables](#).

By default, user-defined triggers are executed for both DML and workspace events, unless the default behavior is changed by using the Workspace Manager system parameter `FIRE_TRIGGERS_FOR_NONDML_EVENTS` (described in [System Parameters for Workspace Manager](#)). You can use the `SetTriggerEvents` procedure to override the current `FIRE_TRIGGERS_FOR_NONDML_EVENTS` setting for specific triggers; however, if you later change the value of the `FIRE_TRIGGERS_FOR_NONDML_EVENTS` system parameter, this new value overrides any setting previously specified using the `SetTriggerEvents` procedure.

If this procedure completes successfully, it commits the caller's open database transaction.

An exception is raised if one or more of the following apply:

- The user is not the trigger owner or does not have the `WM_ADMIN` system privilege.
- `triggerName` does not exist.
- one or more `triggerEvents` values are not valid.

## Examples

The following example enables the trigger `SCOTT.InsertTrigger` only for DML events.

```
EXECUTE DBMS_WM.setTriggerEvents('SCOTT.InsertTrigger', DBMS_WM.DML);
```

The following example enables the trigger `SCOTT.InsertTrigger` for DML events and table merge operations.

```
EXECUTE DBMS_WM.setTriggerEvents('SCOTT.InsertTrigger', dbms_wm.DML || ',' ||
    dbms_wm.TABLE_MERGE_WO_REMOVE_DATA || ',' ||
    dbms_wm.TABLE_MERGE_W_REMOVE_DATA);
```

## 4.91 SetValidTime

Sets the session valid time period. (Valid time support is described in [Workspace Manager Valid Time Support](#).)

### Syntax

```
DBMS_WM.SetValidTime(
    validFrom IN TIMESTAMP WITH TIME ZONE DEFAULT DBMS_WM.CURRENT_TIME,
    validTill IN TIMESTAMP WITH TIME ZONE DEFAULT DBMS_WM.UNTIL_CHANGED);
```

### Parameters

**Table 4-82 SetValidTime Procedure Parameters**

Parameter	Description
validFrom	The start of the session valid time period. The default value is the current timestamp value.
validTill	The end of the session valid time period. The default is that the time remains valid until the session valid time is changed.

### Usage Notes

For information about Workspace Manager valid time support, see [Workspace Manager Valid Time Support](#). [WM\\_PERIOD Data Type](#) explains how `validFrom` and `validTill` values are interpreted.

If this procedure is not invoked in the session or if it is invoked with no parameters, all rows that are valid at the current time are considered valid, and the valid time period is considered to be from the current time forward without limit.

### Examples

The following example sets the session valid time to include all of the year 2003.

```
EXECUTE DBMS_WM.SetValidTime(TO_DATE('01-01-2003', 'MM-DD-YYYY'),
    TO_DATE('01-01-2004', 'MM-DD-YYYY'));
```

## 4.92 SetValidTimeFilterOFF

Removes the valid time filter for the current session.

### Syntax

```
DBMS_WM.SetValidTimeFilterOFF();
```

### Parameters

None.

### Usage Notes

This procedure reverses the effect of the [SetValidTimeFilterON](#) procedure, and causes the previously defined valid time filter to be ignored for queries against tables with valid

time support. Workspace Manager valid time support is explained in [Workspace Manager Valid Time Support](#).

See also the Usage Notes for the [SetValidTimeFilterON](#) procedure.

### Examples

The following example removes the valid time filter for the current session.

```
EXECUTE DBMS_WM.SetValidTimeFilterOFF;
```

## 4.93 SetValidTimeFilterON

Sets a valid time filter for the current session (that is, a time to be applied to version-enabled tables).

### Syntax

```
DBMS_WM.SetValidTimeFilterON(
    filtertime IN TIMESTAMP WITH TIME ZONE DEFAULT NULL);
```

### Parameters

**Table 4-83 SetValidTimeFilterON Procedure Parameters**

Parameter	Description
filtertime	Date to be used as a filter when querying version-enabled tables that have valid time support. The default value is the current time; that is, each select operation on a version-enabled table with valid time support returns data that is valid as of the current time.

### Usage Notes

A **valid time filter** is a time that is applied to queries against version-enabled tables that have valid time support. When a valid time filter is set for the current session, only rows that are valid for the specified time are returned. Workspace Manager valid time support is explained in [Workspace Manager Valid Time Support](#).

The purpose for setting a valid time filter is usually to work with only one row for a given primary key value. For example, assume that for the current valid time period, the session has two rows for employee Adams: the first row is valid from 01-Mar-2004 to 30-Apr-2005, and the second row is valid from 01-May-2005 until it is changed. If you set the valid time filter to 01-Jan-2005 and select all rows for Adams, only the first row (the one valid from 01-Mar-2004 to 30-Apr-2005) is returned. If you remove the valid time filter and select all rows for Adams, both rows are returned.

The `filtertime` value must be in the valid time range for the session. You can set the valid time range using the [SetValidTime](#) procedure.

### Examples

The following example sets a valid time filter so that for queries against version-enabled tables with valid time support, only rows that are valid on January 1, 2005 are returned.

```
EXECUTE DBMS_WM.SetValidTimeFilterOn(TO_DATE('2005-01-01', 'yyyy-mm-dd'));
```

## 4.94 SetWMValidUpdateModeOFF

Disables sequenced and nonsequenced update operations and sequenced delete operations on tables that have valid time support.

### Syntax

```
DBMS_WM.SetWMValidUpdateModeOFF();
```

### Parameters

None.

### Usage Notes

This procedure disables sequenced and nonsequenced update operations and sequenced delete operations on tables that have valid time support. Workspace Manager valid time support is explained in [Workspace Manager Valid Time Support](#); sequenced and nonsequenced update operations and sequenced delete operations are explained in [Update Operations](#).

When sequenced update and delete operations are enabled, when an update or delete operation is performed on a table with valid time support, the session's current valid time period is used so that only rows valid during that period are updated or deleted. However, calling the SetWMValidUpdateModeOFF procedure enables all row data to be updated or deleted, regardless of the valid time period, and causes WM\_VALID column values in the table not to be updated. (This procedure does not affect insert or query operations on tables with valid time support.)

See also the Usage Notes for the [SetWMValidUpdateModeON](#) procedure.

### Examples

The following example disables sequenced and nonsequenced update operations and sequenced delete operations on tables that have valid time support.

```
EXECUTE DBMS_WM.SetWMValidUpdateModeOFF;
```

## 4.95 SetWMValidUpdateModeON

Enables sequenced and nonsequenced update operations and sequenced delete operations on tables that have valid time support.

### Syntax

```
DBMS_WM.SetWMValidUpdateModeON();
```

### Parameters

None.

### Usage Notes

This procedure enables sequenced and nonsequenced update operations and sequenced delete operations on tables that have valid time support. Sequenced update and delete operations are enabled when a table is version-enabled with valid

time support or when valid time support is added to a version-enabled table; however, sequenced update and delete operations can be disabled using the [SetWMValidUpdateModeOFF](#) procedure.

Workspace Manager valid time support is explained in [Workspace Manager Valid Time Support](#); sequenced and nonsequenced update operations and sequenced delete operations are explained in [Insert Operations](#).

### Examples

The following example enables sequenced and nonsequenced update operations and sequenced delete operations on tables that have valid time support. It reverses the effect of the [SetWMValidUpdateModeOFF](#) procedure.

```
EXECUTE DBMS_WM.SetWMValidUpdateModeON;
```

## 4.96 SetWoOverwriteOFF

Disables the `VIEW_WO_OVERWRITE` history option that was enabled by the [EnableVersioning](#) or [SetWoOverwriteON](#) procedure, changing the option to `VIEW_W_OVERWRITE` (*with overwrite*).

### Syntax

```
DBMS_WM.SetWoOverwriteOFF();
```

### Parameters

None.

### Usage Notes

This procedure affects the recording of history information in the views named `<table_name>_HIST` by changing the `VIEW_WO_OVERWRITE` option to `VIEW_W_OVERWRITE`. That is, from this point forward, the views show only the most recent modifications to the same version of the table. A history of modifications to the version is not maintained; that is, subsequent changes to a row in the same version overwrite earlier changes.

This procedure affects only tables that were version-enabled with the `hist` parameter set to `VIEW_WO_OVERWRITE` in the call to the [EnableVersioning](#) procedure.

The `<table_name>_HIST` views are described in [xxx\\_HIST Views](#). The `VIEW_WO_OVERWRITE` and `VIEW_W_OVERWRITE` options are further described in the description of the [EnableVersioning](#) procedure.

The history option affects the behavior of the [GotoDate](#) procedure. See the Usage Notes for that procedure.

The result of the `SetWoOverwriteOFF` procedure remains in effect only for the duration of the current session. To reverse the effect of this procedure, use the [SetWoOverwriteON](#) procedure.

### Examples

The following example disables the `VIEW_WO_OVERWRITE` history option.

```
EXECUTE DBMS_WM.SetWoOverwriteOFF;
```

## 4.97 SetWoOverwriteON

Enables the `VIEW_WO_OVERWRITE` history option that was disabled by the [SetWoOverwriteOFF](#) procedure.

### Syntax

```
DBMS_WM.SetWoOverwriteON();
```

### Parameters

None.

### Usage Notes

This procedure affects the recording of history information in the views named `<table_name>_HIST` by changing the `VIEW_W_OVERWRITE` option to `VIEW_WO_OVERWRITE` (*without overwrite*). That is, from this point forward, the views show all modifications to the same version of the table. A history of modifications to the version is maintained; that is, subsequent changes to a row in the same version do not overwrite earlier changes.

This procedure affects only tables that were affected by a previous call to the [SetWoOverwriteOFF](#) procedure.

The `<table_name>_HIST` views are described in [xxx\\_HIST Views](#). The `VIEW_WO_OVERWRITE` and `VIEW_W_OVERWRITE` options are further described in the description of the [EnableVersioning](#) procedure.

The `VIEW_WO_OVERWRITE` history option can be overridden when a workspace is compressed by specifying the `compress_view_wo_overwrite` parameter as `TRUE` with the [CompressWorkspace](#) or [CompressWorkspaceTree](#) procedure.

The history option affects the behavior of the [GotoDate](#) procedure. See the Usage Notes for that procedure.

To reverse the effect of this procedure, use the [SetWoOverwriteOFF](#) procedure.

### Examples

The following example enables the `VIEW_WO_OVERWRITE` history option.

```
EXECUTE DBMS_WM.SetWoOverwriteON;
```

## 4.98 SetWorkspaceLockModeOFF

Disables Workspace Manager locking for the specified workspace.

### Syntax

```
DBMS_WM.SetWorkspaceLockModeOFF(  
    workspace    IN VARCHAR2,  
    auto_commit  IN BOOLEAN DEFAULT TRUE);
```



## Parameters

**Table 4-84 SetWorkspaceLockModeOFF Procedure Parameters**

Parameter	Description
workspace	Name of the workspace for which to set the locking mode off. The name is case-sensitive.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a> .

## Usage Notes

This procedure turns off Workspace Manager locking that was set on by the [SetWorkspaceLockModeON](#) procedure. Existing locks applied by this session remain locked. All new changes by this session or a subsequent session are not locked, unless the session turns locking on by executing the [SetLockingON](#) procedure.

An exception is raised if any of the following occurs:

- The user does not have the `WM_ADMIN` system privilege or is not the owner of `workspace`.
- `auto_commit` is `TRUE` and an open transaction exists.

## Examples

The following example sets locking off for the workspace named `NEWWORKSPACE`.

```
EXECUTE DBMS_WM.SetWorkspaceLockModeOFF('NEWWORKSPACE');
```

## 4.99 SetWorkspaceLockModeON

Enables Workspace Manager locking for the specified workspace.

### Syntax

```
DBMS_WM.SetWorkspaceLockModeON(
  workspace    IN VARCHAR2,
  lockmode     IN VARCHAR2,
  override     IN BOOLEAN DEFAULT FALSE,
  auto_commit  IN BOOLEAN DEFAULT TRUE);
```

## Parameters

Table 4-85 SetWorkspaceLockModeON Procedure Parameters

Parameter	Description
workspace	Name of the workspace for which to enable Workspace Manager locking. The name is case-sensitive.
lockmode	<p>Default locking mode for row-level locking. Must be E, WE, VE, S, C, or D.</p> <p>E (exclusive) mode locks the rows in the parent workspace and the corresponding rows in the current workspace; no other users in either workspace can change any values.</p> <p>WE (workspace-exclusive) mode locks the rows in the previous version and the corresponding rows in the current version such that only the user that set the lock can change the values in the current workspace; however, other users in other workspaces can change the values.</p> <p>VE (version-exclusive) mode locks the rows in the previous version and the corresponding rows in the current version such that only the user that set the lock can change the values; no other users (in any workspace) can change the values.</p> <p>S (shared) mode locks the rows in the parent workspace and the corresponding rows in the current workspace; however, other users in the current workspace (but no users in the parent workspace) can change values in these rows.</p> <p>C (carry-forward) mode locks rows in the current workspace with the same locking mode as the corresponding rows in the parent workspace. (If a row is not locked in the parent workspace, its corresponding row in the child workspace is not locked.)</p> <p>D (disregard) mode disregards all locks. Any row can be updated in the workspace, even if it is locked by another workspace or the workspace is within a pessimistic setting. The consequences of using this mode are: (1) The workspace for which this mode is set cannot have its lockmode turned off (using <a href="#">SetLockingOFF</a>) or changed (using <a href="#">SetLockingON</a>) after rows have been modified. (2) The workspace cannot be merged. (3) The workspace cannot be refreshed. (4) The <code>override</code> option must be set to <code>false</code>. (5) When any child workspace is created from this workspace, the new workspace will inherit this same lockmode along with its restrictions.</p>
override	<p>A Boolean value (TRUE or FALSE)</p> <p>TRUE allows a session in the workspace to change the <code>lockmode</code> value by using the <a href="#">SetLockingON</a> and <a href="#">SetLockingOFF</a> procedures.</p> <p>FALSE (the default) prevents a session in the workspace from changing the <code>lockmode</code> value.</p>
auto_commit	<p>A Boolean value (TRUE or FALSE).</p> <p>TRUE (the default) causes the operation to be executed as an autonomous database transaction that will be committed when it finishes.</p> <p>FALSE causes the operation to be executed as part of the caller's open database transaction (if one exists). If there is no open database transaction, the operation is executed in a new database transaction. In either case, the caller is responsible for committing the transaction. For more information, see <a href="#">Autocommitting of Workspace Manager Operations</a>.</p>

## Usage Notes

This procedure affects Workspace Manager locking, which occurs in addition to any standard Oracle database locking. Workspace Manager locks can be used to prevent conflicts. When a user locks a row, the corresponding row in the parent workspace is also locked. Thus, when this workspace merges with the parent at merge time, it is guaranteed that this row will not have a conflict.

For information about Workspace Manager lock management, see [Lock Management with Workspace Manager](#).

The main use for the "Disregard" locking mode (`lockmode` value of `D`) is so that a workspace can be completely isolated from the rest of the workspaces in the system and is free to update any rows it wants. It turns the workspace into a test ("sandbox") workspace where anything can be tested, but because it cannot merge or refresh, the workspace is unable to propagate its changes to other workspaces. It is meant for testing only, after which the workspace can be removed.

Exclusive locking (`lockmode` value of `E`) prevents the use of *what-if* scenarios in which different values for one or more columns are tested. Thus, plan any testing of scenarios when exclusive locking is not in effect.

If the override parameter value is `TRUE`, locking can also be enabled and disabled at the user session level with the [SetLockingON](#) and [SetLockingOFF](#) procedures, respectively.

All new changes by this session or a subsequent session are locked, unless the session turns locking off by executing the [SetLockingOFF](#) procedure.

An exception is raised if any of the following occurs:

- The user does not have the `WM_ADMIN` system privilege or is not the owner of `workspace`.
- `auto_commit` is `TRUE` and an open transaction exists.
- `lockmode` is `D` and the workspace either is continually refreshed or is the `LIVE` workspace.

## Examples

The following example sets exclusive locking on for the workspace named `NEWWORKSPACE`.

```
EXECUTE DBMS_WM.SetWorkspaceLockModeON ('NEWWORKSPACE', 'E');
```

All locked rows remain locked until the workspace is merged or rolled back.

## 4.100 UnfreezeWorkspace

Enables access and changes to a workspace, reversing the effect of the [FreezeWorkspace](#) procedure.

### Syntax

```
DBMS_WM.UnfreezeWorkspace(  
    workspace IN VARCHAR2);
```

## Parameters

**Table 4-86 UnfreezeWorkspace Procedure Parameters**

Parameter	Description
workspace	Name of the workspace. The name is case-sensitive.

## Usage Notes

The operation fails if any sessions are in `workspace`.

You can unfreeze a workspace only if one or more of the following apply:

- You are the owner of the specified workspace.
- You have the `WM_ADMIN` system privilege, the `FREEZE_ANY_WORKSPACE` privilege, or the `FREEZE_WORKSPACE` privilege for the specified workspace.

## Examples

The following example unfreezes the `NEWWORKSPACE` workspace.

```
EXECUTE DBMS_WM.UnfreezeWorkspace ('NEWWORKSPACE');
```

## 4.101 UnlockRows

Enables access to versioned rows in a specified table and to corresponding rows in the parent workspace.

### Syntax

```
DBMS_WM.UnlockRows(
  workspace      IN VARCHAR2,
  table_name     IN VARCHAR2,
  where_clause   IN VARCHAR2 DEFAULT '',
  all_or_user    IN VARCHAR2 DEFAULT 'USER',
  lock_mode      IN VARCHAR2 DEFAULT 'ES',
  Xmin           IN NUMBER DEFAULT NULL,
  Ymin           IN NUMBER DEFAULT NULL,
  Xmax           IN NUMBER DEFAULT NULL,
  Ymax           IN NUMBER DEFAULT NULL);
```

## Parameters

**Table 4-87 UnlockRows Procedure Parameters**

Parameter	Description
workspace	Name of the workspace: locked rows in this workspace and corresponding rows in the parent workspace will be unlocked, as specified in the remaining parameters. The name is case-sensitive.  A value of <code>NONE</code> can be used if <code>lock_mode</code> is set to <code>VE</code> (version-exclusive). This causes rows locked by any workspace to be unlocked.

**Table 4-87 (Cont.) UnlockRows Procedure Parameters**

Parameter	Description
<code>table_name</code>	Name of the table or (if <code>Xmin</code> , <code>Ymin</code> , <code>Xmax</code> , and <code>Ymax</code> are specified) Spatial and Graph topology in which rows are to be unlocked. The name is not case-sensitive.
<code>where_clause</code>	<p>The <code>WHERE</code> clause (excluding the <code>WHERE</code> keyword) identifying the rows to be unlocked. Example: <code>'department_id = 20'</code></p> <p>Only primary key columns can be specified in the <code>WHERE</code> clause, except in a subquery. The subquery can refer to columns that are not primary keys, but it cannot refer to a version-enabled table.</p> <p>If the <code>where_clause</code> parameter is not specified, all rows in <code>table_name</code> are made accessible.</p> <p>Do not specify the <code>where_clause</code> parameter if <code>table_name</code> specifies a Spatial and Graph topology name.</p>
<code>all_or_user</code>	<p>Scope of the request: <code>ALL</code> or <code>USER</code>.</p> <p><code>ALL</code>: All locks accessible by the user in the specified workspace are considered.</p> <p><code>USER</code> (default): Only locks owned by the user in the specified workspace are considered.</p>
<code>lock_mode</code>	<p>Locking mode: <code>E</code>, <code>WE</code>, <code>VE</code>, <code>S</code>, or <code>ES</code> (default).</p> <p><code>E</code> (exclusive): Only exclusive mode locks are considered.</p> <p><code>WE</code> (workspace-exclusive): Only workspace-exclusive mode locks are considered.</p> <p><code>VE</code> (version-exclusive): Only version-exclusive mode locks are considered.</p> <p><code>S</code> (shared): Only shared mode locks are considered.</p> <p><code>ES</code> (exclusive and shared: the default): Both exclusive mode and shared mode locks are considered.</p>
<code>Xmin, Ymin</code>	For Oracle Spatial and Graph topologies only (see <a href="#">Locking Considerations with Topologies</a> ), the X and Y coordinate values, respectively, of the lower-left corner of the window containing the rows to be locked. You must specify these parameters if you specified a topology name for <code>table_name</code> ; otherwise, do not specify these parameters.
<code>Xmax, Ymax</code>	For Oracle Spatial and Graph topologies only (see <a href="#">Locking Considerations with Topologies</a> ), the X and Y coordinate values, respectively, of the upper-right corner of the window containing the rows to be locked. You must specify these parameters if you specified a topology name for <code>table_name</code> ; otherwise, do not specify these parameters.

### Usage Notes

This procedure affects Workspace Manager locking, which occurs in addition to any standard Oracle database locking. For an explanation of Workspace Manager locking, see [Lock Management with Workspace Manager](#).

This procedure unlocks rows that were previously locked (see the [LockRows](#) procedure). It does not affect whether Workspace Manager locking is set on or off (determined by the [SetLockingON](#) and [SetLockingOFF](#) procedures).

For information about Workspace Manager locking for tables in an Oracle Spatial and Graph topology, see [Locking Considerations with Topologies](#).

## Examples

The following example unlocks the `EMPLOYEES` table where `last_name = 'Smith'` in the `NEWWORKSPACE` workspace.

```
EXECUTE DBMS_WM.UnlockRows ('employees', 'NEWWORKSPACE', 'last_name =
'Smith');
```

## 4.102 UseDefaultValuesForNulls

Determines whether or not Workspace Manager, for the current session, uses the default value for a column when the user either specifies a null value or does not specify any value for the column in an insert operation on a version-enabled table.

### Syntax

```
DBMS_WM.UseDefaultValuesForNulls(
  mode_var IN VARCHAR2);
```

### Parameters

**Table 4-88 UseDefaultValuesForNulls Procedure Parameters**

Parameter	Description
<code>mode_var</code>	Mode for handling the insertion of null values: OFF or ON. OFF: A null value is inserted into the column. (This is the normal Oracle behavior.) ON: The default value for the column is inserted into the column.

### Usage Notes

This procedure affects what Workspace Manager does only if an `INSERT` statement into a version-enabled table explicitly specifies `NULL` for a column when the column has been defined as having a default value or leaves the column unspecified. For example, assume the following table definition:

```
CREATE TABLE players (name VARCHAR2(20) primary key, rating NUMBER DEFAULT 10);
```

If the `PLAYERS` table is version-enabled and if you have executed this procedure with a `mode_var` parameter value of `OFF`, either of the following statements would insert a row for `Smith` with a null `RATING` value:

```
INSERT INTO players VALUES ('Smith', NULL);
INSERT INTO players(name) VALUES ('Smith');
```

However, if you have executed the `UseDefaultValuesForNulls` procedure with a `mode_var` parameter value of `ON`, both statements insert a row for `Smith` with a `RATING` value of 10. If this procedure is not executed in a session, the default behavior is the same as if `mode_var` was specified as `ON`.

### Examples

The following example causes the column default value to be used during the rest of the current session whenever an `INSERT` statement into a version-enabled table

specifies a null value for a column that has a default value or the column is left unspecified.

```
EXECUTE DBMS_WM.UseDefaultValuesForNulls('ON');
```

# 5

## Workspace Manager Static Data Dictionary Views

Workspace Manager creates and maintains static data dictionary views to hold information about such things as version-enabled tables, workspaces, savepoints, users, privileges, locks, and conflicts.

These views are read-only to users. You can use the information in these views to help administer the Workspace Manager environment and diagnose problems.

There are also views created for each version-enabled table, as follows:

- Conflict view, each having a name in the form <table\_name>\_CONF. (See [xxx\\_CONF Views](#).)
- Difference view, each having a name in the form <table\_name>\_DIFF. (See [xxx\\_DIFF Views](#).)
- History view (if history tracking is enabled), each having a name in the form <table\_name>\_HIST. (See [xxx\\_HIST Views](#).)
- Lock view, each having a name in the form <table\_name>\_LOCK. (See [xxx\\_LOCK Views](#).)
- Multiworkspace view, each having a name in the form <table\_name>\_MW. (See [xxx\\_MW Views](#).)



### Note:

When an ALL\_xxx or USER\_xxx view is queried from a procedure, the results returned are based on whether the procedure has definer's rights or the rights of the database user whose privileges are currently active.

### CDB\_xxx Views

For every DBA\_xxx view, a CDB\_xxx view is defined. In the root of a multitenant container database (CDB), CDB\_xxx views can be used to obtain information about tables, tablespaces, users, privileges, parameters, and so on contained in the root and in pluggable databases (PDBs).

CDB\_xxx views are container data objects. When a user connected to the root queries a CDB\_xxx view, the query results will depend on the CONTAINER\_DATA attribute for users for the view. The CONTAINER\_DATA clause of the SQL ALTER USER statement is used to set and modify users' CONTAINER\_DATA attribute.

The CDB\_xxx views are owned by SYS, regardless of which user owns the underlying DBA\_xxx view.

By default, a user connected to the root will only see data pertaining to the root.



In a PDB, the CDB\_xxx views only show objects visible through a corresponding DBA\_xxx view.

In addition to all the columns found in a given DBA\_xxx view, the corresponding CDB\_xxx view also contains the CON\_ID column, which identifies a container whose data a given CDB\_\* row represents. In a non-CDB, the value of a CON\_ID column will be 0.

CDB views can return data from different containers in a CDB when queried from the root container. These objects will implicitly convert data to the character set of the root container (AL32UTF8) and then return the result to the user. Some character sets may have character expansion (more bytes needed to represent a character) when converted to AL32UTF8, so there may be data truncation if the view column width is not able to accommodate data from a given PDB.

Data returned by these views depends on whether a given PDB is open at the time the query is issued. In particular, in an Oracle RAC environment, data returned by these view may vary according to the instance to which a session is connected.

- [ALL\\_MP\\_GRAPH\\_WORKSPACES](#)
- [ALL\\_MP\\_PARENT\\_WORKSPACES](#)
- [ALL\\_REMOVED\\_WORKSPACES](#)
- [ALL\\_VERSION\\_HVIEW](#)
- [ALL\\_WM\\_CONS\\_COLUMNS](#)
- [ALL\\_WM\\_CONSTRAINT\\_VIOLATIONS](#)
- [ALL\\_WM\\_CONSTRAINTS](#)
- [ALL\\_WM\\_IND\\_COLUMNS](#)
- [ALL\\_WM\\_IND\\_EXPRESSIONS](#)
- [ALL\\_WM\\_LOCKED\\_TABLES](#)
- [ALL\\_WM\\_MODIFIED\\_TABLES](#)
- [ALL\\_WM\\_POLICIES](#)
- [ALL\\_WM\\_RIC\\_INFO](#)
- [ALL\\_WM\\_TAB\\_TRIGGERS](#)
- [ALL\\_WM\\_VERSIONED\\_TABLES](#)
- [ALL\\_WM\\_VT\\_ERRORS](#)
- [ALL\\_WORKSPACE\\_PRIVS](#)
- [ALL\\_WORKSPACE\\_SAVEPOINTS](#)
- [ALL\\_WORKSPACES](#)
- [DBA\\_REMOVED\\_WORKSPACES](#)
- [DBA\\_WM\\_SYS\\_PRIVS](#)
- [DBA\\_WM\\_VERSIONED\\_TABLES](#)
- [DBA\\_WM\\_VT\\_ERRORS](#)
- [DBA\\_WORKSPACE\\_PRIVS](#)
- [DBA\\_WORKSPACE\\_SAVEPOINTS](#)

- DBA\_WORKSPACE\_SESSIONS
- DBA\_WORKSPACES
- ROLE\_WM\_PRIVS
- USER\_MP\_GRAPH\_WORKSPACES
- USER\_MP\_PARENT\_WORKSPACES
- USER\_REMOVED\_WORKSPACES
- USER\_WM\_CONS\_COLUMNS
- USER\_WM\_CONSTRAINTS
- USER\_WM\_IND\_COLUMNS
- USER\_WM\_IND\_EXPRESSIONS
- USER\_WM\_LOCKED\_TABLES
- USER\_WM\_MODIFIED\_TABLES
- USER\_WM\_POLICIES
- USER\_WM\_PRIVS
- USER\_WM\_RIC\_INFO
- USER\_WM\_TAB\_TRIGGERS
- USER\_WM\_VERSIONED\_TABLES
- USER\_WM\_VT\_ERRORS
- USER\_WORKSPACE\_PRIVS
- USER\_WORKSPACE\_SAVEPOINTS
- USER\_WORKSPACES
- WM\_COMPRESS\_BATCH\_SIZES
- WM\_COMPRESSIBLE\_TABLES
- WM\_EVENTS\_INFO
- WM\_INSTALLATION
- xxx\_CONF Views
- xxx\_DIFF Views
- xxx\_HIST Views
- xxx\_LOCK Views
- xxx\_MW Views

## 5.1 ALL\_MP\_GRAPH\_WORKSPACES

ALL\_MP\_GRAPH\_WORKSPACES contains information about multiparent graph workspaces (explained in [Multiparent Workspaces](#)) for which the leaf workspace can be accessed by the current user.

### **Related View**

- [USER\\_MP\\_GRAPH\\_WORKSPACES](#) contains information about multiparent graph workspaces for which the leaf workspace is owned by the current user.

Column	Datatype	Null?	Description
MP_LEAF_WORKSPACE	VARCHAR2 (128)	NOT NULL	Name of the multiparent leaf workspace.
GRAPH_WORKSPACE	VARCHAR2 (128)	NOT NULL	Name of the multiparent graph workspace.
GRAPH_FLAG	VARCHAR2 (22)		L if the multiparent graph workspace is the leaf workspace in the multiparent graph; I if the multiparent graph workspace is an intermediate workspace in the multiparent graph; R if the multiparent graph workspace is the root workspace in the multiparent graph.

## 5.2 ALL\_MP\_PARENT\_WORKSPACES

ALL\_MP\_PARENT\_WORKSPACES contains information about parent workspaces of multiparent workspaces (explained in [Multiparent Workspaces](#)) that the current user can access.

### Related View

- [USER\\_MP\\_PARENT\\_WORKSPACES](#) contains information about parent workspaces of multiparent workspaces that the current user owns.

Column	Datatype	Null?	Description
MP_LEAF_WORKSPACE	VARCHAR2 (128)	NOT NULL	Name of the multiparent leaf workspace.
PARENT_WORKSPACE	VARCHAR2 (128)	NOT NULL	Name of the parent workspace.
CREATOR	VARCHAR2 (128)		Name of the user that made PARENT_WORKSPACE a parent workspace of MP_LEAF_WORKSPACE.
CREATETIME	TIMESTAMP WITH TIME_ZONE		Date and time when PARENT_WORKSPACE became a parent workspace of MP_LEAF_WORKSPACE.
ISREFRESHED	VARCHAR2 (3)		YES if the multiparent leaf workspace is a continually refreshed workspace; NO if the multiparent leaf workspace is not a continually refreshed workspace.
PARENT_FLAG	VARCHAR2 (17)		DP if PARENT_WORKSPACE is the default parent of MP_LEAF_WORKSPACE; MP if PARENT_WORKSPACE was added as a parent of MP_LEAF_WORKSPACE.

## 5.3 ALL\_REMOVED\_WORKSPACES

ALL\_REMOVED\_WORKSPACES contains information about workspaces, that the current user has access to, that have been removed during a [RemoveWorkspace](#) operation or a [MergeWorkspace](#) operation in which the `remove_workspace` parameter value was `true`, and while the value of the Workspace Manager system parameter `KEEP_REMOVED_WORKSPACES_INFO` was ON. (This system parameter is described in [System Parameters for Workspace Manager](#).)

### Related Views

- [USER\\_REMOVED\\_WORKSPACES](#) contains information about workspaces, that the current user owns, that have been removed during a [RemoveWorkspace](#) operation or a [MergeWorkspace](#) operation in which the `remove_workspace` parameter value was `true`, and while the value of the Workspace Manager system parameter `KEEP_REMOVED_WORKSPACES_INFO` was `ON`.
- [DBA\\_REMOVED\\_WORKSPACES](#) contains information about workspaces that have been removed during a [RemoveWorkspace](#) operation or a [MergeWorkspace](#) operation in which the `remove_workspace` parameter value was `true`, and while the value of the Workspace Manager system parameter `KEEP_REMOVED_WORKSPACES_INFO` was `ON`.

Column	Datatype	Null?	Description
OWNER	VARCHAR2 (128)		User name of the owner of the removed workspace.
WORKSPACE_NAME	VARCHAR2 (128)		Name of the removed workspace.
WORKSPACE_ID	NUMBER (38)	NOT NULL	ID of the removed workspace.
PARENT_WORKSPACE_NAME	VARCHAR2 (128)		Name of the parent workspace of the removed workspace.
PARENT_WORKSPACE_ID	NUMBER (38)		ID of the parent workspace of the removed workspace.
CREATETIME	TIMESTAMP WITH TIME ZONE		Date and time when the removed workspace was created.
RETIRETIME	TIMESTAMP WITH TIME ZONE		Date and time when the removed workspace was removed.
DESCRIPTION	VARCHAR2 (1000)		Description of the removed workspace.
MP_ROOT_WORKSPACE_ID	NUMBER (38)		ID of the root workspace of the multiparent graph; null if the workspace is not part of a multiparent graph. (Multiparent workspaces are explained in <a href="#">Multiparent Workspaces</a> .)
CONTINUALLY_REFRESHED	VARCHAR2 (3)		YES if the workspace is continually refreshed (see the description of the <code>isrefreshed</code> parameter of the <a href="#">CreateWorkspace</a> procedure); NO if the workspace is not continually refreshed.

## 5.4 ALL\_VERSION\_HVIEW

`ALL_VERSION_HVIEW` contains information about the version hierarchy. It is used by Workspace Manager to perform queries against the `xxx_HIST` views (described in [xxx\\_HIST Views](#)).

Column	Datatype	Null?	Description
VERSION	NUMBER (38)	NOT NULL	Version number of the workspace identified in the <code>WORKSPACE</code> column.
PARENT_VERSION	NUMBER (38)		Version number of the parent version of the version identified in the <code>VERSION</code> column.

Column	Datatype	Null?	Description
WORKSPACE	VARCHAR2 (128)		Name of the workspace associated with the version number in the VERSION column.
WORKSPACE_ID	NUMBER		ID of the workspace associated with the version number in the VERSION column.

## 5.5 ALL\_WM\_CONS\_COLUMNS

ALL\_WM\_CONS\_COLUMNS contains information about columns in unique constraints on version-enabled tables on which the current user has one or more of the following privileges: SELECT, INSERT, UPDATE, or DELETE.

### Related View

- [USER\\_WM\\_CONS\\_COLUMNS](#) contains information about columns in unique constraints on version-enabled tables that the current user owns.

Column	Datatype	Null?	Description
OWNER	VARCHAR2 (128)		User name of the constraint owner.
CONSTRAINT_NAME	VARCHAR2 (128)		Name of the constraint.
TABLE_NAME	VARCHAR2 (128)		Name of the version-enabled table on which the constraint is defined.
COLUMN_NAME	VARCHAR2 (4000)		Column in the constraint definition.
POSITION	NUMBER		Position of the column in the constraint.

## 5.6 ALL\_WM\_CONSTRAINT\_VIOLATIONS

ALL\_WM\_CONSTRAINT\_VIOLATIONS contains information related to unique, foreign key, and check constraint violation errors raised while executing the [MergeWorkspace](#), [MergeTable](#), [RefreshWorkspace](#), [RefreshTable](#), [CommitDDL](#), [AddAsParentWorkspace](#), [RemoveAsParentWorkspace](#), and [PurgeTable](#) procedures.

The view is only populated within a session after executing one of the these procedures and it fails due to a constraint violation. The view data is cleared any time one of these procedures is executed and a constraint check is successfully performed or the session disconnects.

Column	Datatype	Null?	Description
VIOLATION#	NUMBER	NOT NULL	An identifier representing the violation number.
TABLE_NAME1	VARCHAR2 (128)		Name of the first table involved in the constraint violation.
PREDICATE1	CLOB		A primary key predicate that can be used to identify the violating row of the first table.
VERSION1	NUMBER		The version of the row within the first table involved in the constraint violation.

Column	Datatype	Null?	Description
TABLE_NAME2	VARCHAR2 (128)		Name of the second table involved in the constraint violation.
PREDICATE2	CLOB		A primary key predicate that can be used to identify the violating row of the second table.
VERSION2	NUMBER		The version of the row within the second table involved in the constraint violation.

## 5.7 ALL\_WM\_CONSTRAINTS

ALL\_WM\_CONSTRAINTS contains information about constraints on version-enabled tables on which the current user has one or more of the following privileges: SELECT, INSERT, UPDATE, or DELETE. It provides information about the following kinds of constraints: UNIQUE constraint, unique index, PRIMARY KEY constraints, and CHECK constraints.

### Related View

- [USER\\_WM\\_CONSTRAINTS](#) contains information about constraints on version-enabled tables that the current user owns.

Column	Datatype	Null?	Description
OWNER	VARCHAR2 (128)	NOT NULL	User name of the constraint owner. (Same as the owner of TABLE_NAME.)
CONSTRAINT_NAME	VARCHAR2 (128)		Name of the constraint.
CONSTRAINT_TYPE	VARCHAR2 (2)		One of the following values: P = primary constraint, PU = primary constraint enforced using unique index, PN = primary constraint enforced using non-unique index, U = unique constraint, UU = unique constraint enforced using unique index, UN = unique constraint enforced using non-unique index, UI = unique index.
TABLE_NAME	VARCHAR2 (128)		Name of the table on which the constraint is defined.
SEARCH_CONDITION	CLOB		Condition for checking the constraint.
STATUS	VARCHAR2 (8)		ENABLED if the constraint is enabled; DISABLED if the constraint is disabled.
INDEX_OWNER	VARCHAR2 (128)		Owner of the index used for enforcing the constraint.
INDEX_NAME	VARCHAR2 (128)		Name of the index used for enforcing the constraint.
INDEX_TYPE	VARCHAR2 (26)		NORMAL if the index is not a function-based index; FUNCTION-BASED NORMAL for a function-based index.

## 5.8 ALL\_WM\_IND\_COLUMNS

ALL\_WM\_IND\_COLUMNS contains information about indexes used for enforcing unique constraints on version-enabled tables on which the current user has one or more of the following privileges: SELECT, INSERT, UPDATE, or DELETE.

### Related View

- [USER\\_WM\\_IND\\_COLUMNS](#) contains information about indexes used for enforcing unique constraints on version-enabled tables that the current user owns.

Column	Datatype	Null?	Description
INDEX_OWNER	VARCHAR2 (128)		User name of the index owner.
INDEX_NAME	VARCHAR2 (128)		Name of the index.
OWNER	VARCHAR2 (128)		User name of the owner of the version-enabled table on which the index is defined.
TABLE_NAME	VARCHAR2 (128)		Name of the version-enabled table on which the index is defined.
COLUMN_NAME	VARCHAR2 (4000)		Column on which the index is defined.
COLUMN_POSITION	NUMBER		Position of the column in the index.
COLUMN_LENGTH	NUMBER		Length of the column.
DESCEND	VARCHAR2 (4)		ASC if the column data in the index is in ascending order; DESC if the column data in the index is in descending order.

## 5.9 ALL\_WM\_IND\_EXPRESSIONS

ALL\_WM\_IND\_EXPRESSIONS contains information about functional expressions on functional unique indexes on version-enabled tables on which the current user has one or more of the following privileges: SELECT, INSERT, UPDATE, or DELETE.

### Related View

- [USER\\_WM\\_IND\\_EXPRESSIONS](#) contains information about functional expressions on functional unique indexes on version-enabled tables that the current user owns.

Column	Datatype	Null?	Description
INDEX_OWNER	VARCHAR2 (128)		User name of the index owner.
INDEX_NAME	VARCHAR2 (128)		Name of the index.
OWNER	VARCHAR2 (128)		User name of the owner of the version-enabled table on which the index is defined.
TABLE_NAME	VARCHAR2 (128)		Name of the version-enabled table on which the index is defined.
COLUMN_EXPRESSION	LONG		Text of the functional expression on which the index is defined.
COLUMN_POSITION	NUMBER		Position of the expression in the index.

## 5.10 ALL\_WM\_LOCKED\_TABLES

ALL\_WM\_LOCKED\_TABLES contains information about Workspace Manager locks on rows in version-enabled tables that the current user can access.

### Related View

- [USER\\_WM\\_LOCKED\\_TABLES](#) contains information about Workspace Manager locks on rows in version-enabled tables that the current user owns.

Column	Datatype	Null?	Description
TABLE_OWNER	VARCHAR2 (128)		User name of the table owner.
TABLE_NAME	VARCHAR2 (128)		Name of the table.
LOCK_MODE	VARCHAR2 (19)		Type of lock: EXCLUSIVE or SHARED.
LOCK_OWNER	VARCHAR2 (4000)		User name of the owner of the lock.
LOCKING_STATE	VARCHAR2 (4000)		Workspace in which the lock was placed.

## 5.11 ALL\_WM\_MODIFIED\_TABLES

ALL\_WM\_MODIFIED\_TABLES contains information about all version-enabled tables that have been modified and on which the current user has one or more of the following privileges: SELECT, INSERT, DELETE, UPDATE.

### Related View

- [USER\\_WM\\_MODIFIED\\_TABLES](#) contains information about version-enabled tables that have been modified and that the current user owns.

Column	Datatype	Null?	Description
TABLE_NAME	VARCHAR2 (257)		Name of a version-enabled table.
WORKSPACE	VARCHAR2 (128)	NOT NULL	Workspace in which the modification occurred.
SAVEPOINT	VARCHAR2 (128)		Name of the savepoint associated with the most recent modification, or LATEST if a savepoint does not yet exist is the workspace.

## 5.12 ALL\_WM\_POLICIES

ALL\_WM\_POLICIES contains information about Oracle Virtual Private Database (VPD) security policies defined on any version-enabled table or related view accessible to the current user. Its columns are the same as those in the ALL\_POLICIES view, described in *Oracle Database Reference*.

Workspace Manager uses this information to provide VPD support, which is described in [Virtual Private Database Considerations](#).

### Related View

- [USER\\_WM\\_POLICIES](#) contains information about Oracle Virtual Private Database (VPD) security policies defined on any version-enabled table or related view owned by the current user.

## 5.13 ALL\_WM\_RIC\_INFO

ALL\_WM\_RIC\_INFO contains information about referential integrity constraints in version-enabled tables that the current user can access. Workspace Manager uses this information to provide referential integrity support, which is described in [Referential Integrity Support](#).

### Related View



- [USER\\_WM\\_RIC\\_INFO](#) contains information about referential integrity constraints in version-enabled tables that the current user owns.

Column	Datatype	Null?	Description
CT_OWNER	VARCHAR2 (128)	NOT NULL	Owner of the child table in the referential integrity constraint.
CT_NAME	VARCHAR2 (128)		Name of the child table in the referential integrity constraint.
PT_OWNER	VARCHAR2 (128)		Owner of the parent table in the referential integrity constraint.
PT_NAME	VARCHAR2 (128)		Name of the parent table in the referential integrity constraint.
RIC_NAME	VARCHAR2 (128)	NOT NULL	Name of the referential integrity constraint.
CT_COLS	VARCHAR2 (4000)		List of foreign key columns in the child table in the referential integrity constraint.
PT_COLS	VARCHAR2 (4000)		List of foreign key columns in the parent table in the referential integrity constraint.
R_CONSTRAINT_NAME	VARCHAR2 (128)		Name of the unique constraint defined on the parent table in the referential integrity constraint.
DELETE_RULE	VARCHAR2 (1)		Rule to apply when deletion occurs in the parent table. C (Cascade) causes related child table rows to be deleted; N (Set Null) causes the foreign key of related child table rows to be set to null; R (Restrict) prevents the deletion if any related child table rows exist.
STATUS	VARCHAR2 (8)		ENABLED if the referential integrity constraint is enabled; DISABLED if the referential integrity constraint is disabled.

## 5.14 ALL\_WM\_TAB\_TRIGGERS

ALL\_WM\_TAB\_TRIGGERS contains information about triggers that the current user created and for version-enabled tables owned by the current user that have triggers defined on them. If the current user has the `CREATE ANY TRIGGER` privilege, trigger information is displayed for all version-enabled tables.

### Related View

- [USER\\_WM\\_TAB\\_TRIGGERS](#) contains information about triggers that are owned by the current user and that are on version-enabled tables.

Column	Datatype	Null?	Description
TRIGGER_OWNER	VARCHAR2 (128)	NOT NULL	Owner (schema) of the trigger.
TRIGGER_NAME	VARCHAR2 (128)	NOT NULL	Name of the trigger.
TABLE_OWNER	VARCHAR2 (128)		Owner (schema) of the table on which the trigger is defined.
TABLE_NAME	VARCHAR2 (128)		Name of the table on which the trigger is defined.

Column	Datatype	Null?	Description
TRIGGER_TYPE	VARCHAR2 (4000)		Trigger type: one of the codes described following this table.
STATUS	VARCHAR2 (8)		ENABLED if the trigger is enabled; DISABLED if the trigger is disabled.
WHEN_CLAUSE	VARCHAR2 (4000)		Clause that must evaluate to TRUE for the trigger body (TRIGGER_BODY) to execute.
DESCRIPTION	VARCHAR2 (4000)		Description of the trigger. Useful if the trigger must be re-created.
TRIGGER_BODY	CLOB		Statements executed by the trigger.
TAB_MERGE_WO_REMOVE	VARCHAR2 (3)		ON if DBMS_WM.TABLE_MERGE_WO_REMOVE_DATA has been set (see the <a href="#">SetTriggerEvents</a> procedure), or OFF if DBMS_WM.TABLE_MERGE_WO_REMOVE_DATA has not been set.
TAB_MERGE_W_REMOVE	VARCHAR2 (3)		ON if DBMS_WM.TABLE_MERGE_W_REMOVE_DATA has been set (see the <a href="#">SetTriggerEvents</a> procedure), or OFF if DBMS_WM.TABLE_MERGE_W_REMOVE_DATA has not been set.
WSPC_MERGE_WO_REMOVE	VARCHAR2 (3)		ON if DBMS_WM.WORKSPACE_MERGE_WO_REMOVE_DATA has been set (see the <a href="#">SetTriggerEvents</a> procedure), or OFF if DBMS_WM.WORKSPACE_MERGE_WO_REMOVE_DATA has not been set.
WSPC_MERGE_W_REMOVE	VARCHAR2 (3)		ON if DBMS_WM.WORKSPACE_MERGE_W_REMOVE_DATA has been set (see the <a href="#">SetTriggerEvents</a> procedure), or OFF if DBMS_WM.WORKSPACE_MERGE_W_REMOVE_DATA has not been set.
DML	VARCHAR2 (3)		ON if DBMS_WM.DML has been set (see the <a href="#">SetTriggerEvents</a> procedure), or OFF if DBMS_WM.DML has not been set.
TABLE_IMPORT	VARCHAR2 (3)		ON if DBMS_WM.TABLE_IMPORT has been set (see the <a href="#">SetTriggerEvents</a> procedure), or OFF if DBMS_WM.TABLE_IMPORT has not been set.

TRIGGER\_TYPE is one of the following values:

- BIR: before insert for each row
- AIR: after insert for each row
- BUR: before update for each row
- AUR: after update for each row
- BDR: before delete for each row
- ADR: after delete for each row
- BIS: before insert for each statement
- AIS: after insert for each statement

- BUS: before update for each statement
- AUS: after update for each statement
- BDS: before delete for each statement
- ADS: after delete for each statement

## 5.15 ALL\_WM\_VERSIONED\_TABLES

ALL\_WM\_VERSIONED\_TABLES contains information about all version-enabled tables on which the current user has one or more of the following privileges: SELECT, INSERT, DELETE, UPDATE.

### Related Views

- [USER\\_WM\\_VERSIONED\\_TABLES](#) contains information about version-enabled tables that the current user owns.
- [DBA\\_WM\\_VERSIONED\\_TABLES](#) contains information about all that the current user owns.

Column	Datatype	Null?	Description
TABLE_NAME	VARCHAR2 (128)	NOT NULL	Name of a version-enabled table.
OWNER	VARCHAR2 (128)	NOT NULL	Owner (schema) of the table.
STATE	VARCHAR2 (11)		State of the table: one of the values described following this table.
HISTORY	VARCHAR2 (22)		History option for the table: NONE, VIEW_W_OVERWRITE, or VIEW_WO_OVERWRITE. (For an explanation of the history option values, see the information about the <a href="#">EnableVersioning</a> procedure in <a href="#">DBMS_WM Package: Reference</a> .)
NOTIFICATION	VARCHAR2 (3)		(Not used for this release.)
CONFLICT	VARCHAR2 (4000)		YES if there are any conflicts on the table between the workspace that performed the <a href="#">SetConflictWorkspace</a> operation and its parent workspace; otherwise, NO.
DIFF	VARCHAR2 (4000)		YES if there are any differences for this table as a result of a <a href="#">SetDiffVersions</a> operation; otherwise, NO.
VALIDTIME	VARCHAR2 (3)		YES if valid time is enabled on the table; otherwise, NO.

STATE is one of the following values:

- ADD\_VT: The [AlterVersionedTable](#) procedure (with the `alter_option` parameter set to 'ADD\_VALID\_TIME') is being performed on this table.
- AVTDDL: The [AlterVersionedTable](#) procedure is being performed on this table.
- BDDL: The [BeginDDL](#) procedure is being performed on the table.
- BL\_F\_BEGIN: The [CommitBulkLoading](#) procedure is being performed on this table.
- BL\_P\_BEGIN: The [BeginBulkLoading](#) procedure is being performed on this table.
- BL\_P\_END: The table is active in a bulk loading session.

- **BL\_R\_BEGIN**: The [RollbackBulkLoading](#) procedure is being performed on this table.
- **CDDL**: The [CommitDDL](#) procedure is being performed on the table.
- **D\_HIST\_COLS**: The columns of the table version-enabled are being downgraded.
- **DDL**: The table is active in a DDL session.
- **DV**: The table is being version-disabled.
- **EV**: The table is being version-enabled.
- **IDDL**: The [Initialize\\_After\\_Import](#) procedure is being performed on the table.
- **LW\_DISABLED**: The table has been lightweight version-disabled (an internal operation).
- **LWDV**: The table is being lightweight version-disabled (an internal operation).
- **LWEV**: The table is being lightweight version-enabled (an internal operation).
- **ODDL**: The [GetOriginalDDL](#) procedure is being performed on this table.
- **RB\_IND**: The [AlterVersionedTable](#) procedure (with the `alter_option` parameter set to 'REBUILD\_INDEX') is being performed on this table.
- **RN\_CONS**: The [AlterVersionedTable](#) procedure (with the `alter_option` parameter set to 'RENAME\_CONSTRAINT') is being performed on this table.
- **RN\_IND**: The [AlterVersionedTable](#) procedure (with the `alter_option` parameter set to 'RENAME\_INDEX') is being performed on this table.
- **SYNCVTV1**: The [AlterVersionedTable](#) procedure (with the `alter_option` parameter set to 'USE\_WM\_PERIOD\_FOR\_VALIDTIME') is being performed on this table.
- **SYNCVTV2**: The [AlterVersionedTable](#) procedure (with the `alter_option` parameter set to 'USE\_SCALAR\_TYPES\_FOR\_VALIDTIME') is being performed on this table.
- **TDDEL**: The [Add\\_Topo\\_Geometry\\_Layer](#) or [Add\\_Topo\\_Geometry\\_Layer](#) procedure is being performed on this table.
- **U\_HIST\_COLS**: The columns of the table version-enabled are being upgraded.
- **VERSIONED**: The table has been version-enabled.

## 5.16 ALL\_WM\_VT\_ERRORS

ALL\_WM\_VT\_ERRORS contains information about the error that occurred during the last call to the [DisableVersioning](#) or [CommitDDL](#) procedure that specified a table on which the current user has one or more of the following privileges: SELECT, INSERT, DELETE, UPDATE.

### Related View

- **USER\_WM\_VT\_ERRORS** contains information about the error that occurred during the last call to the [DisableVersioning](#) or [CommitDDL](#) procedure that specified a table that the current user owns and on which the current user has one or more of the following privileges: SELECT, INSERT, DELETE, UPDATE.

Column	Datatype	Null?	Description
OWNER	VARCHAR2 (128)	NOT NULL	Owner (schema) of the table.

Column	Datatype	Null?	Description
TABLE_NAME	VARCHAR2 (128)	NOT NULL	Name of a version-enabled table.
STATE	VARCHAR2 (11)		State of the table. For example, <i>EV</i> means that the table is being version-enabled, and <i>DV</i> means that the table is being version-disabled.
SQL_STR	VARCHAR2 (4000)		The SQL statement that failed during the processing of the <a href="#">DisableVersioning</a> or <a href="#">CommitDDL</a> procedure.
STATUS	VARCHAR2 (49)		Information about the state of the SQL statement that failed during the processing of the <a href="#">DisableVersioning</a> or <a href="#">CommitDDL</a> procedure.
ERROR_MSG	VARCHAR2 (200)		Error message caused by the SQL statement that failed during the processing of the <a href="#">DisableVersioning</a> or <a href="#">CommitDDL</a> procedure.

## 5.17 ALL\_WORKSPACE\_PRIVS

ALL\_WORKSPACE\_PRIVS contains information about Workspace Manager privileges in all workspaces that the current user can access.

### Related Views

- [USER\\_WORKSPACE\\_PRIVS](#) contains information about Workspace Manager privileges in workspaces created by the current user.
- [DBA\\_WORKSPACE\\_PRIVS](#) contains information about Workspace Manager privileges in all workspaces.

Column	Datatype	Null?	Description
GRANTEE	VARCHAR2 (128)		User or role to which the privilege was granted.
WORKSPACE	VARCHAR2 (128)		Name of the workspace.
PRIVILEGE	VARCHAR2 (23)		Name of the Workspace Manager privilege.
GRANTOR	VARCHAR2 (128)		User or role that granted the privilege.
GRANTABLE	VARCHAR2 (3)		YES if grantee was given the grant option (that is, can grant the privilege to other users); NO if grantee was not given the grant option.

## 5.18 ALL\_WORKSPACE\_SAVEPOINTS

ALL\_WORKSPACE\_SAVEPOINTS contains information about savepoints in all workspaces that the current user can access.

### Related Views

- [USER\\_WORKSPACE\\_SAVEPOINTS](#) contains information about savepoints in workspaces created by the current user.
- [DBA\\_WORKSPACE\\_SAVEPOINTS](#) contains information about savepoints in all workspaces.

Column	Datatype	Null?	Description
SAVEPOINT	VARCHAR2 (128)	NOT NULL	Name of the savepoint. Explicit savepoints are named by users; implicit savepoints are named by Workspace Manager.
WORKSPACE	VARCHAR2 (128)	NOT NULL	Workspace in which the savepoint was created.
IMPLICIT	VARCHAR2 (3)		YES if the savepoint is implicit (that is, was created automatically by Workspace Manager); NO if the savepoint is explicit (that is, was created by a user).
POSITION	NUMBER (38)		Position of the savepoint in the sequence in which savepoints were created.
OWNER	VARCHAR2 (128)		Name of the user that created the savepoint.
CREATETIME	TIMESTAMP WITH TIME ZONE		Date and time that the savepoint was created.
DESCRIPTION	VARCHAR2 (1000)		Description of the savepoint.
CANROLLBACKTO	VARCHAR2 (3)		YES if the savepoint can be rolled back to; NO if the savepoint cannot be rolled back to. In a <a href="#">RollbackToSP</a> operation, if any implicit savepoints have greater POSITION values than the position of the savepoint to be rolled back to, you must first merge or remove the workspaces that caused these intervening implicit savepoints to be created.
REMOVABLE	VARCHAR2 (3)		YES if the savepoint can be removed; NO if the savepoint cannot be removed. An implicit savepoint cannot be removed if it has any child dependencies; all other implicit savepoints and all explicit savepoints can be removed.
VERSION	NUMBER		Version number of the specified savepoint within the workspace.

## 5.19 ALL\_WORKSPACES

ALL\_WORKSPACES contains information about all workspaces that the current user can access.

Its columns are the same as those for the [DBA\\_WORKSPACES](#) view, except for the following:

- ALL\_WORKSPACES includes the following columns that are not in [DBA\\_WORKSPACES](#): CONTINUALLY\_REFRESHED, WORKSPACE\_LOCKMODE, and WORKSPACE\_LOCKMODE\_OVERRIDE.
- [DBA\\_WORKSPACES](#) includes the following columns that are not in ALL\_WORKSPACES: SID and SERIAL#.

### Related Views

- [DBA\\_WORKSPACES](#) contains information about all workspaces. This view is only available to users with the WM\_ADMIN\_ROLE or SELECT\_CATALOG\_ROLE role.
- [USER\\_WORKSPACES](#) contains information about workspaces created by the current user.

Column	Datatype	Null?	Description
WORKSPACE	VARCHAR2 (128)		Name of the workspace.
WORKSPACE_ID	NUMBER (38)		ID of the workspace.
PARENT_WORKSPACE	VARCHAR2 (128)		Parent workspace of this workspace.
PARENT_SAVEPOINT	VARCHAR2 (128)		Implicit savepoint that was created in the parent workspace when this workspace was created.
OWNER	VARCHAR2 (128)		Name of the user that created the workspace.
CREATETIME	TIMESTAMP WITH TIME_ZONE		Date and time that the workspace was created.
DESCRIPTION	VARCHAR2 (1000)		Description of the workspace.
FREEZE_STATUS	VARCHAR2 (8)		FROZEN if the workspace is frozen (by a <a href="#">FreezeWorkspace</a> operation); UNFROZEN if the workspace is not frozen.
FREEZE_MODE	VARCHAR2 (16)		NO_ACCESS, READ_ONLY, 1WRITER, or 1WRITER_SESSION. See the <code>freezemode</code> parameter description for the <a href="#">FreezeWorkspace</a> procedure in <a href="#">DBMS_WM Package: Reference</a> .
FREEZE_WRITER	VARCHAR2 (128)		The user allowed to make changes in the workspace; or null if the workspace is not frozen or if it is frozen in NO_ACCESS or READ_ONLY mode. See the <code>freezewriter</code> parameter description for the <a href="#">FreezeWorkspace</a> procedure in <a href="#">DBMS_WM Package: Reference</a> .
FREEZE_OWNER	VARCHAR2 (128)		Name of the user that froze the workspace.
SESSION_DURATION	VARCHAR2 (3)		YES if the workspace is frozen only for the duration of the current session; NO if the workspace is frozen until an explicit <a href="#">UnfreezeWorkspace</a> procedure call is made; null if the workspace is not currently frozen.
CURRENT_SESSION	VARCHAR2 (3)		YES if the current session is allowed to make changes in the workspace; NO if the current session is not allowed to make changes in the workspace; null if the workspace is not currently frozen in <code>session_duration</code> mode.
RESOLVE_STATUS	VARCHAR2 (8)		ACTIVE if a conflict resolution session is in progress; INACTIVE if a conflict resolution session is not in progress.
RESOLVE_USER	VARCHAR2 (128)		Name of the user that started the conflict resolution session if <code>resolve_status</code> is ACTIVE; otherwise, null.
CONTINUALLY_REFRESHED	VARCHAR2 (3)		YES if the workspace is continually refreshed (see the description of the <code>isrefreshed</code> parameter of the <a href="#">CreateWorkspace</a> procedure); NO if the workspace is not continually refreshed.
WORKSPACE_LOCKMODE	VARCHAR2 (19)		EXCLUSIVE if the locking mode is exclusive; SHARED if the locking mode is shared; CARRY if the locking mode is carry-forward. See the <code>lockmode</code> parameter description for the <a href="#">SetWorkspaceLockModeON</a> procedure in <a href="#">DBMS_WM Package: Reference</a> .

Column	Datatype	Null?	Description
WORKSPACE_LOCKMODE_OVERRIDE	VARCHAR2 (3)		YES if the override option is TRUE; NO if the override option is FALSE; null if the workspace lock mode is not set. See the <code>override</code> parameter description for the <code>SetWorkspaceLockModeON</code> procedure in <a href="#">DBMS_WM Package: Reference</a> .
MP_ROOT_WORKSPACE	VARCHAR2 (128)		Name of the root workspace of the multiparent graph; null if the workspace is not part of a multiparent graph. (Multiparent workspaces are explained in <a href="#">Multiparent Workspaces</a> .)

## 5.20 DBA\_REMOVED\_WORKSPACES

DBA\_REMOVED\_WORKSPACES contains information about workspaces that have been removed during a [RemoveWorkspace](#) operation or a [MergeWorkspace](#) operation in which the `remove_workspace` parameter value was `true`, and while the value of the Workspace Manager system parameter `KEEP_REMOVED_WORKSPACES_INFO` was `ON`. (This system parameter is described in [System Parameters for Workspace Manager](#).) Its columns are the same as those in [ALL\\_REMOVED\\_WORKSPACES](#). This view is only available to users with the `WM_ADMIN_ROLE` or `SELECT_CATALOG_ROLE` role.

## 5.21 DBA\_WM\_SYS\_PRIVS

DBA\_WM\_SYS\_PRIVS contains information about all users that have Workspace Manager system-level privileges (that is, privilege names containing `_ANY_WORKSPACE`, as explained in [Privilege Management with Workspace Manager](#)). This view is only available to users with the `WM_ADMIN_ROLE` or `SELECT_CATALOG_ROLE` role.

Column	Datatype	Null?	Description
GRANTEE	VARCHAR2 (128)		User or role to which the system-level privilege was granted.
PRIVILEGE	VARCHAR2 (23)		Name of the Workspace Manager system-level privilege.
GRANTOR	VARCHAR2 (128)		User or role that granted the system-level privilege.
GRANTABLE	VARCHAR2 (3)		YES if <code>grantee</code> was given the grant option (that is, can grant the privilege to other users); NO if <code>grantee</code> was not given the grant option.

## 5.22 DBA\_WM\_VERSIONED\_TABLES

DBA\_WM\_VERSIONED\_TABLES contains information about all version-enabled tables. Its columns are the same as those in [ALL\\_WM\\_VERSIONED\\_TABLES](#). This view is only available to users with the `WM_ADMIN_ROLE` or `SELECT_CATALOG_ROLE` role.

## 5.23 DBA\_WM\_VT\_ERRORS

DBA\_WM\_VT\_ERRORS contains information about the error that occurred during the last call to the [DisableVersioning](#), [CommitDDL](#), or [RecoverFromDroppedUser](#) procedure. Its



columns are the same as those in [ALL\\_WM\\_VT\\_ERRORS](#). This view is only available to users with the `WM_ADMIN_ROLE` or `SELECT_CATALOG_ROLE` role.

## 5.24 DBA\_WORKSPACE\_PRIVS

`DBA_WORKSPACE_PRIVS` contains information about Workspace Manager privileges in all workspaces. Its columns are the same as those in [ALL\\_WORKSPACE\\_PRIVS](#). This view is only available to users with the `WM_ADMIN_ROLE` or `SELECT_CATALOG_ROLE` role.

## 5.25 DBA\_WORKSPACE\_SAVEPOINTS

`DBA_WORKSPACE_SAVEPOINTS` contains information about savepoints in all workspaces. Its columns are the same as those in [ALL\\_WORKSPACE\\_SAVEPOINTS](#). This view is only available to users with the `WM_ADMIN_ROLE` or `SELECT_CATALOG_ROLE` role.

## 5.26 DBA\_WORKSPACE\_SESSIONS

`DBA_WORKSPACE_SESSIONS` contains information about all users and workspaces (except for the `LIVE` workspace). This view is only available to users with the `WM_ADMIN_ROLE` or `SELECT_CATALOG_ROLE` role. It is useful for monitoring users in the different workspaces.

Column	Datatype	Null?	Description
USERNAME	VARCHAR2 (128)		User name.
WORKSPACE	VARCHAR2 (128)	NOT NULL	Workspace that the user is currently in.
SID	NUMBER		Session ID.
STATUS	VARCHAR2 (8)		ACTIVE if the user currently has an open transaction (that is, a database transaction); INACTIVE if the user does not have an open transaction.
ISIMPLICIT	VARCHAR2 (3)		NO if the session has executed <code>GotoWorkspace</code> or performed a DML operation on a version-enabled table. Otherwise, YES for any session in the <code>LIVE</code> workspace (the default for any session connecting to the database).

## 5.27 DBA\_WORKSPACES

`DBA_WORKSPACES` contains information about all workspaces, including those whose removal has been deferred. This view is only available to users with the `WM_ADMIN_ROLE` or `SELECT_CATALOG_ROLE` role.

Its columns are the same as those for the [ALL\\_WORKSPACES](#) view, except for the following:

- [ALL\\_WORKSPACES](#) includes the following columns that are not in `DBA_WORKSPACES`: `CONTINUALLY_REFRESHED`, `WORKSPACE_LOCKMODE`, and `WORKSPACE_LOCKMODE_OVERRIDE`.

- **DBA\_WORKSPACES** includes the following columns that are not in **ALL\_WORKSPACES**: **SID**, **SERIAL#**, and **INST\_ID**.

#### Related Views

- **ALL\_WORKSPACES** contains information about all workspaces.
- **USER\_WORKSPACES** contains information about workspaces created by the current user.

Column	Datatype	Null?	Description
WORKSPACE	VARCHAR2 (128)		Name of the workspace.
WORKSPACE_ID	NUMBER (38)		ID of the workspace.
PARENT_WORKSPACE	VARCHAR2 (128)		Parent workspace of this workspace.
PARENT_SAVEPOINT	VARCHAR2 (128)		Implicit savepoint that was created in the parent workspace when this workspace was created.
OWNER	VARCHAR2 (128)		Name of the user that created the workspace.
CREATETIME	TIMESTAMP WITH TIME ZONE		Date and time that the workspace was created.
DESCRIPTION	VARCHAR2 (1000)		Description of the workspace.
FREEZE_STATUS	VARCHAR2 (8)		FROZEN if the workspace is frozen (by a <a href="#">FreezeWorkspace</a> operation); UNFROZEN if the workspace is not frozen.
FREEZE_MODE	VARCHAR2 (16)		NO_ACCESS, READ_ONLY, 1WRITER, 1WRITER_SESSION, or DEFERRED_REMOVAL. See the <a href="#">freezemode</a> parameter description for the <a href="#">FreezeWorkspace</a> procedure in <a href="#">DBMS_WM Package: Reference</a> .  If the value is 1WRITER_SESSION, you can examine the <b>SID</b> , <b>SERIAL#</b> , and <b>INST_ID</b> column values to see which session has a lock on the workspace.
FREEZE_WRITER	VARCHAR2 (128)		The user allowed to make changes in the workspace; or null if the workspace is not frozen or if it is frozen in NO_ACCESS, READ_ONLY, or DEFERRED_REMOVAL mode. See the <a href="#">freezewriter</a> parameter description for the <a href="#">FreezeWorkspace</a> procedure in <a href="#">DBMS_WM Package: Reference</a> .
SID	NUMBER		Oracle session identifier (SID) of the associated session.
SERIAL#	NUMBER		Serial number associated with the session.
INST_ID	NUMBER		Instance ID associated with the session.
FREEZE_OWNER	VARCHAR2 (128)		Name of the user that froze the workspace.
SESSION_DURATION	VARCHAR2 (3)		YES if the workspace is frozen only for the duration of the current session; NO if the workspace is frozen until an explicit <a href="#">UnfreezeWorkspace</a> procedure call is made; null if the workspace is not currently frozen.
CURRENT_SESSION	VARCHAR2 (3)		YES if the current session is allowed to make changes in the workspace; NO if the current session is not allowed to make changes in the workspace; null if the workspace is not currently frozen in <a href="#">session_duration</a> mode.

Column	Datatype	Null?	Description
RESOLVE_STATUS	VARCHAR2 (8)		ACTIVE if a conflict resolution session is in progress; INACTIVE if a conflict resolution session is not in progress.
RESOLVE_USER	VARCHAR2 (128)		Name of the user that started the conflict resolution session if resolve_status is ACTIVE; otherwise, null.
MP_ROOT_WORKSPACE	VARCHAR2 (128)		Name of the root workspace of the multiparent graph; null if the workspace is not part of a multiparent graph. (Multiparent workspaces are explained in <a href="#">Multiparent Workspaces</a> .)

## 5.28 ROLE\_WM\_PRIVS

ROLE\_WM\_PRIVS contains information about privileges that all roles granted to the current user have in each workspace.

### Related View

- [USER\\_WM\\_PRIVS](#) contains information about privileges that the current user has in each workspace.

Column	Datatype	Null?	Description
ROLE	VARCHAR2 (128)		Name of the role.
WORKSPACE	VARCHAR2 (128)		Name of the workspace.
PRIVILEGE	VARCHAR2 (23)		Name of the Workspace Manager privilege.
GRANTABLE	VARCHAR2 (3)		YES if the role was given the grant option (that is, can grant the privilege to other users); NO if the role was not given the grant option.

## 5.29 USER\_MP\_GRAPH\_WORKSPACES

USER\_MP\_GRAPH\_WORKSPACES contains information about multiparent graph workspaces (explained in [Multiparent Workspaces](#)) for which the leaf workspace is owned by the current user. Its columns are the same as those in [ALL\\_MP\\_GRAPH\\_WORKSPACES](#).

## 5.30 USER\_MP\_PARENT\_WORKSPACES

USER\_MP\_PARENT\_WORKSPACES contains information about parent workspaces of multiparent workspaces (explained in [Multiparent Workspaces](#)) that the current user owns. Its columns are the same as those in [ALL\\_MP\\_PARENT\\_WORKSPACES](#).

## 5.31 USER\_REMOVED\_WORKSPACES

USER\_REMOVED\_WORKSPACES contains information about workspaces, that the current user owns, that have been removed during a [RemoveWorkspace](#) operation or a [MergeWorkspace](#) operation in which the remove\_workspace parameter value was

true, and while the value of the Workspace Manager system parameter `KEEP_REMOVED_WORKSPACES_INFO` was ON. (This system parameter is described in [System Parameters for Workspace Manager](#).) Its columns are the same as those in [ALL\\_REMOVED\\_WORKSPACES](#).

## 5.32 USER\_WM\_CONS\_COLUMNS

`USER_WM_CONS_COLUMNS` contains information about columns in unique constraints on version-enabled tables that the current user owns. Its columns are the same as those in [ALL\\_WM\\_CONS\\_COLUMNS](#), except it does not contain an `OWNER` column.

## 5.33 USER\_WM\_CONSTRAINTS

`USER_WM_CONSTRAINTS` contains information about constraints on version-enabled tables that the current user owns. It provides information about the following kinds of constraints: `UNIQUE` constraint, unique index, `PRIMARY KEY` constraints, and `CHECK` constraints. Its columns are the same as those in [ALL\\_WM\\_CONSTRAINTS](#), except it does not contain an `OWNER` or `INDEX_OWNER` column.

## 5.34 USER\_WM\_IND\_COLUMNS

`USER_WM_IND_COLUMNS` contains information about indexes used for enforcing unique constraints on version-enabled tables that the current user owns. Its columns are the same as those in [ALL\\_WM\\_IND\\_COLUMNS](#), except it does not contain an `OWNER` column.

## 5.35 USER\_WM\_IND\_EXPRESSIONS

`USER_WM_IND_EXPRESSIONS` contains information about indexes used for enforcing unique constraints on version-enabled tables that the current user owns. Its columns are the same as those in [ALL\\_WM\\_IND\\_EXPRESSIONS](#), except it does not contain an `OWNER` column.

## 5.36 USER\_WM\_LOCKED\_TABLES

`USER_WM_LOCKED_TABLES` contains information about Workspace Manager locks on rows in version-enabled tables that the current user owns. Its columns are the same as those in [ALL\\_WM\\_LOCKED\\_TABLES](#).

## 5.37 USER\_WM\_MODIFIED\_TABLES

`USER_WM_MODIFIED_TABLES` contains information about version-enabled tables that have been modified and that the current user owns. Its columns are the same as those in [ALL\\_WM\\_MODIFIED\\_TABLES](#).

## 5.38 USER\_WM\_POLICIES

`USER_WM_POLICIES` contains information about Oracle Virtual Private Database (VPD) security policies defined on any version-enabled table or related view owned by the current

user. Its columns are the same as those in the [ALL\\_WM\\_POLICIES](#) view, except it does not include an OWNER column.

Workspace Manager uses this information to provide VPD support, which is described in [Virtual Private Database Considerations](#).

## 5.39 USER\_WM\_PRIVS

USER\_WM\_PRIVS contains information about privileges that the current user has in each workspace.

### Related View

- [ROLE\\_WM\\_PRIVS](#) contains information about privileges that all roles granted to the current user have in each workspace.

Column	Datatype	Null?	Description
WORKSPACE	VARCHAR2 (128)		Name of the workspace.
PRIVILEGE	VARCHAR2 (23)		Name of the Workspace Manager privilege.
GRANTOR	VARCHAR2 (128)		Name of the user that granted the privilege to the current user.
GRANTABLE	VARCHAR2 (3)		YES if the user was given the grant option (that is, can grant the privilege to other users); NO if the user was not given the grant option.

## 5.40 USER\_WM\_RIC\_INFO

USER\_WM\_RIC\_INFO contains information about referential integrity constraints in version-enabled tables that the current user owns. Its columns are the same as those in [ALL\\_WM\\_RIC\\_INFO](#).

Workspace Manager uses this information to provide referential integrity support, which is described in [Referential Integrity Support](#).

## 5.41 USER\_WM\_TAB\_TRIGGERS

USER\_WM\_TAB\_TRIGGERS contains information about triggers that are owned by the current user and that are on version-enabled tables. Its columns are the same as those in [ALL\\_WM\\_TAB\\_TRIGGERS](#), except that it does not contain the TRIGGER\_OWNER column.

## 5.42 USER\_WM\_VERSIONED\_TABLES

USER\_WM\_VERSIONED\_TABLES contains information about version-enabled tables that the current user owns. Its columns are the same as those in [ALL\\_WM\\_VERSIONED\\_TABLES](#).

## 5.43 USER\_WM\_VT\_ERRORS

USER\_WM\_VT\_ERRORS contains information about the error that occurred during the last call to the [DisableVersioning](#) or [CommitDDL](#) procedure that specified a table that the current user owns and on which the current user has one or more of the following privileges: SELECT, INSERT, DELETE, UPDATE. Its columns are the same as those in [ALL\\_WM\\_VT\\_ERRORS](#).

## 5.44 USER\_WORKSPACE\_PRIVS

USER\_WORKSPACE\_PRIVS contains information about Workspace Manager privileges in workspaces created by the current user. Its columns are the same as those in [ALL\\_WORKSPACE\\_PRIVS](#).

## 5.45 USER\_WORKSPACE\_SAVEPOINTS

USER\_WORKSPACE\_SAVEPOINTS contains information about savepoints in workspaces created by the current user. Its columns are the same as those in [ALL\\_WORKSPACE\\_SAVEPOINTS](#).

## 5.46 USER\_WORKSPACES

USER\_WORKSPACES contains information about workspaces created by the current user. Its columns are the same as those in [ALL\\_WORKSPACES](#).

## 5.47 WM\_COMPRESS\_BATCH\_SIZES

WM\_COMPRESS\_BATCH\_SIZES contains information related to compression capabilities for version-enabled tables. This view is only available to users with the WM\_ADMIN\_ROLE or SELECT\_CATALOG\_ROLE role.

Column	Datatype	Null?	Description
OWNER	VARCHAR2 (128)	NOT NULL	User name of the table owner.
TABLE_NAME	VARCHAR2 (128)	NOT NULL	Name of the version-enabled table.
BATCH_SIZE	VARCHAR2 (23)		TABLE if the table can be compressed as a single batch only; TABLE/PRIMARY_KEY_RANGE if the table can be compressed as a single batch or in multiple batches.
NUM_BATCHES	NUMBER		1 if BATCH_SIZE is TABLE, or a number specifying the number of batches to be used for compression operations when a batch size of PRIMARY_KEY_RANGE is used.

## 5.48 WM\_COMPRESSIBLE\_TABLES

WM\_COMPRESSIBLE\_TABLES contains information about version-enabled tables that need to be compressed (if compression is to be performed) between two savepoints in a workspace. To create rows in this view, use the [SetCompressWorkspace](#) procedure.

Column	Datatype	Null?	Description
OWNER	VARCHAR2 (128)	NOT NULL	User name of the table owner.
TABLE_NAME	VARCHAR2 (128)	NOT NULL	Name of the version-enabled table.
WORKSPACE	VARCHAR2 (256)		Name of a workspace that was set as a result of a call to the <a href="#">SetCompressWorkspace</a> procedure.
BEGIN_SAVEPOINT	VARCHAR2(256)		Savepoint on the first version of the compression range. If the <code>firstSP</code> parameter was null in the call to the <a href="#">SetCompressWorkspace</a> procedure, this column contains <code>BEGINNING</code> .
END_SAVEPOINT	VARCHAR2(256)		Savepoint on the last version of the compression range. If both the <code>firstSP</code> and <code>secondSP</code> parameters were null in the call to the <a href="#">SetCompressWorkspace</a> procedure, this column contains <code>LATEST</code> .

## 5.49 WM\_EVENTS\_INFO

WM\_EVENTS\_INFO contains information about the capture of Workspace Manager events. For information about Workspace Manager events, see [Workspace Manager Events](#).

Column	Datatype	Null?	Description
EVENT_NAME	VARCHAR2 (128)	NOT NULL	Name indicating the type of event.
CAPTURE	VARCHAR2 (3)		ON if events of this type are being captured; OFF if events of this type are not being captured.

## 5.50 WM\_INSTALLATION

WM\_INSTALLATION contains information about the installed release of Workspace Manager. The information includes the Workspace Manager version number (`OWM_VERSION`) and the Workspace Manager system parameters.

Column	Datatype	Null?	Description
NAME	VARCHAR2 (128)		Name of an informational item or system parameter pertaining to the current release of Workspace Manager on the system. (System parameters are explained in <a href="#">System Parameters for Workspace Manager</a> .)

Column	Datatype	Null?	Description
VALUE	VARCHAR2 (4000)		Value associated with the informational item or system parameter identified in the NAME column.

## 5.51 xxx\_CONF Views

There is one conflict view for each version-enabled table. Each conflict view has a name in the form <table\_name>\_CONF. For example, if the EMPLOYEE table is version-enabled, the EMPLOYEE\_CONF metadata view exists.

Each conflict view contains the columns shown in [Table 5-1](#).

**Table 5-1 Columns in the xxx\_CONF Views**

Column	Datatype	Description
WM_WORKSPACE	VARCHAR2 (256)	Workspace in which the conflict exists.
(One column for each column in original table)	(Same as column in original table)	Value of the column in this workspace.
WM_VALID	WM_PERIOD	Time period during which the row is valid, if the table has valid time support (described in <a href="#">Workspace Manager Valid Time Support</a> ). If you set the USE_SCALAR_TYPES_FOR_VALIDTIME system parameter (described in <a href="#">System Parameters for Workspace Manager</a> ) to ON, this column is replaced by two columns, WM_VALIDFROM and WM_VALIDTILL, of type TIMEZONE WITH TIMESTAMP.
WM_DELETED	VARCHAR2 (3)	YES if the row has been deleted; NO if the row has not been deleted; NE if the row is nonexistent (does not exist).
WM_CONFLICTPERIOD	WM_PERIOD	Overlapping period of the rows for which conflicts were detected, if the table has valid time support (described in <a href="#">Workspace Manager Valid Time Support</a> ).

A SELECT operation from a conflict view uses the workspace conflict context established by the [GotoWorkspace](#) procedure, unless you have specified a workspace conflict context for the session by using the [SetConflictWorkspace](#) procedure. Selecting from the conflict view finds rows in that table that are changed in the current workspace context, and compares their values with corresponding rows in the parent workspace to identify conflicts. If the current workspace conflict context is the LIVE workspace, all rows in the table are selected and no conflicts are found.

The following example lists the key value and all column values of conflicting rows in the EMPLOYEE table in the current workspace and its parent workspace. The conflict view reflects the context established by a previous call to the [GetWorkspace](#) or [SetConflictWorkspace](#) procedure to set the workspace conflict context (the current workspace in this case).

```
SELECT * FROM EMPLOYEE_CONF;
```



If `ID`, `NAME`, and `CITY` are the columns in the `EMPLOYEE` table, then assume the following values:

<u>WM_WORKSPACE</u>	<u>ID</u>	<u>NAME</u>	<u>CITY</u>	<u>WM_DELETED</u>
NEWWORKSPACE	12	SMITH	NASHUA	NO
DiffBase	12	SMITH	NY	NO
LIVE	12	SMITH	BOSTON	NO

The database row identified by `ID = 12` was changed in `NEWWORKSPACE` and `LIVE` workspaces. In `NEWWORKSPACE` the city was changed to `NASHUA`, and in the `LIVE` workspace the city was changed to `BOSTON`. When `NEWWORKSPACE` is merged into `LIVE`, this row will show up as a conflict. The application must pick between the choices and resolve conflicts in favor of the workspace with the desired value.

Note that `DiffBase` refers to the common ancestor (or *base*), as explained in the Usage Notes for the [SetDiffVersions](#) procedure.

The following example begins a conflict resolution session, calls the [ResolveConflicts](#) procedure to delete the conflicting row from the `NEWWORKSPACE` workspace and to insert the value in the parent workspace (`LIVE`) into both workspaces, commits the transaction, and ends the conflict resolution session.

```
DBMS_WM.BeginResolve ('NEWWORKSPACE');
DBMS_WM.ResolveConflicts ('NEWWORKSPACE', 'EMPLOYEE', 'ID = 12', 'PARENT');
COMMIT;
DBMS_WM.CommitResolve ('NEWWORKSPACE');
```

For additional information about conflict resolution, see [Resolving Conflicts Before a Merge or Refresh Operation](#).

## 5.52 xxx\_DIFF Views

There is one difference view for each version-enabled table. Each difference view has a name in the form `<table_name>_DIFF`. For example, if the `EMPLOYEE` table is version-enabled, the `EMPLOYEE_DIFF` metadata view exists. Rows are added to one or more `xxx_DIFF` views each time the [SetDiffVersions](#) procedure is executed.

Each difference view contains the columns shown in [Table 5-2](#).

**Table 5-2 Columns in the xxx\_DIFF Views**

Column	Datatype	Description
(One column for each column in original table)	(Same as column in original table)	Value of the column in this workspace.

Table 5-2 (Cont.) Columns in the xxx\_DIFF Views

Column	Datatype	Description
WM_VALID	WM_PERIOD	Time period during which the row is valid, if the table has valid time support (described in <a href="#">Workspace Manager Valid Time Support</a> ).  If you set the <code>USE_SCALAR_TYPES_FOR_VALIDTIME</code> system parameter (described in <a href="#">System Parameters for Workspace Manager</a> ) to ON, this column is replaced by two columns, <code>WM_VALIDFROM</code> and <code>WM_VALIDTILL</code> , of type <code>TIMEZONE WITH TIMESTAMP</code> .
WM_DIFFVER	VARCHAR2 (256)	Branch from which the values in the preceding columns are taken. (See the explanation following this table.)
WM_CODE	VARCHAR2 (2)	One of the following codes describing the change: U (updated), D (deleted), I (inserted), NC (no change), NE (nonexistent).
WM_DIFFPERIOD	WM_PERIOD	Overlapping period of the rows for which a difference was detected, if the table has valid time support (described in <a href="#">Workspace Manager Valid Time Support</a> ).

The `WM_DIFFVER` value is in one of the following formats:

- '<workspace1>, <savepoint1>'
- '<workspace2>, <savepoint2>'
- 'DiffBase'

If the two-parameter version of the [SetDiffVersions](#) procedure was used, the value of `savepoint1` or `savepoint2` is `LATEST`.

Note the following about the possible values for `WM_CODE`:

- `NC` will appear for rows in workspaces that did not change the value when another workspace did change the value. For example, if '<workspace2>, <savepoint2>' updated the row, the code for that row is `U`, but the code for the '<workspace1>, <savepoint1>' and 'DiffBase' rows is `NC` if they did not modify the row.
- `NE` will appear for 'DiffBase' if a row is inserted in one or more branches, and `NE` will appear for 'DiffBase' and a branch if only one branch has had any insert operations.

For more information, including an example showing rows being added to a differences view, see the section on the [SetDiffVersions](#) procedure in [DBMS\\_WM Package: Reference](#).

## 5.53 xxx\_HIST Views

There is one history view for each version-enabled table, regardless of the value specified for the `hist` parameter in the call to the [EnableVersioning](#) procedure. Each history view has a

name in the form <table\_name>\_HIST. For example, if the `EMPLOYEE` table is version-enabled, the `EMPLOYEE_HIST` metadata view exists.

You can use the history views to log and audit modifications to version-enabled tables.

Each history view contains the columns shown in [Table 5-3](#). However, the `WM_CREATETIME` and `WM_RETIRETIME` columns are included only if the `hist` parameter was set to `VIEW_W_OVERWRITE` or `VIEW_WO_OVERWRITE` in the call to the [EnableVersioning](#) procedure.

**Table 5-3 Columns in the xxx\_HIST Views**

Column	Datatype	Description
(One column for each column in original table)	(Same as column in original table)	Value of the column in this workspace.
<code>WM_VALID</code>	<code>WM_PERIOD</code>	Time period during which the row is valid, if the table has valid time support (described in <a href="#">Workspace Manager Valid Time Support</a> ).  If you set the <code>USE_SCALAR_TYPES_FOR_VALIDTIME</code> system parameter (described in <a href="#">System Parameters for Workspace Manager</a> ) to <code>ON</code> , this column is replaced by two columns, <code>WM_VALIDFROM</code> and <code>WM_VALIDTILL</code> , of type <code>TIMEZONE WITH TIMESTAMP</code> .
<code>WM_WORKSPACE</code>	<code>VARCHAR2 (128)</code>	Name of the workspace containing the row.
<code>WM_VERSION</code>	<code>NUMBER (38)</code>	Version number of the row with which the data is associated.
<code>WM_USERNAME</code>	<code>VARCHAR2 (128)</code>	Name of the user that created the row.
<code>WM_OPTYPE</code>	<code>VARCHAR2 (1)</code>	Type of change operation that was performed on the row: D (delete), I (insert), or U (update).
<code>WM_CREATETIME</code>	<code>TIMESTAMP WITH TIME ZONE</code>	Time when the row was created or updated. (This column is present only if <code>VIEW_W_OVERWRITE</code> or <code>VIEW_WO_OVERWRITE</code> was specified for the <code>hist</code> parameter in the call to the <a href="#">EnableVersioning</a> procedure.)
<code>WM_RETIRETIME</code>	<code>TIMESTAMP WITH TIME ZONE</code>	Time when the row was deleted or modified. (This column is present only if <code>VIEW_W_OVERWRITE</code> or <code>VIEW_WO_OVERWRITE</code> was specified for the <code>hist</code> parameter in the call to the <a href="#">EnableVersioning</a> procedure.)

## 5.54 xxx\_LOCK Views

There is one lock view for each version-enabled table. Each lock view has a name in the form <table\_name>\_LOCK. For example, if the `EMPLOYEE` table is version-enabled, the `EMPLOYEE_LOCK` metadata view exists. (For an explanation of Workspace Manager locking, see [Lock Management with Workspace Manager](#).)

Each lock view contains the columns shown in [Table 5-4](#).

**Table 5-4 Columns in the xxx\_LOCK Views**

Column	Datatype	Description
(One column for each column in original table)	(Same as column in original table)	Value of the column in this workspace.
WM_VALID	WM_PERIOD	Time period during which the row is valid, if the table has valid time support (described in <a href="#">Workspace Manager Valid Time Support</a> ). If you set the <code>USE_SCALAR_TYPES_FOR_VALIDTIME</code> system parameter (described in <a href="#">System Parameters for Workspace Manager</a> ) to ON, this column is replaced by two columns, <code>WM_VALIDFROM</code> and <code>WM_VALIDTILL</code> , of type <code>TIMEZONE WITH TIMESTAMP</code> .
WM_LOCKMODE	VARCHAR2 (19)	Type of lock: EXCLUSIVE, WORKSPACE EXCLUSIVE, VERSION EXCLUSIVE, or SHARED.
WM_USERNAME	VARCHAR2 (128)	User name of the owner of the lock.
WM_LOCKINGWORKSPACE	VARCHAR2 (128)	Name of the workspace in which the lock was placed.
WM_INCURWORKSPACE	VARCHAR2 (3)	YES if the row is contained in the current workspace; NO if the row is not contained in the current workspace.

## 5.55 xxx\_MW Views

There is one multiworkspace view for each version-enabled table. Each multiworkspace view has a name in the form `<table_name>_MW`. For example, if the `EMPLOYEE` table is version-enabled, the `EMPLOYEE_MW` metadata view exists. Rows are added to one or more `xxx_MW` views each time the [SetMultiWorkspaces](#) procedure (described in [DBMS\\_WM Package: Reference](#)) is executed.

Each multiworkspace view contains the columns shown in [Table 5-5](#).

**Table 5-5 Columns in the xxx\_MW Views**

Column	Datatype	Description
(One column for each column in original table)	(Same as column in original table)	Value of the column in this workspace.

Table 5-5 (Cont.) Columns in the xxx\_MW Views

Column	Datatype	Description
WM_VALID	WM_PERIOD	Time period during which the row is valid, if the table has valid time support (described in <a href="#">Workspace Manager Valid Time Support</a> ).  If you set the <code>USE_SCALAR_TYPES_FOR_VALIDTIME</code> system parameter (described in <a href="#">System Parameters for Workspace Manager</a> ) to ON, this column is replaced by two columns, <code>WM_VALIDFROM</code> and <code>WM_VALIDTILL</code> , of type <code>TIMEZONE WITH TIMESTAMP</code> .
WM_MODIFIED_BY	VARCHAR2(128)	Workspace containing the row that was modified.
WM_SEEN_BY	CLOB	Comma-delimited list of workspaces from which the row is visible.
WM_OPTYPE	VARCHAR2(1)	One of the following codes describing the change: U (updated), I (inserted).

You can use the `<table_name>_MW` view to see changes in another workspace without leaving the current workspace (for example, to check if there is a conflict with the other workspace). Each row in the view shows the data as it would be in that workspace if the workspace had been merged when the row was inserted in the view.

You can also use the `<table_name>_DIFF` view (see [xxx\\_DIFF Views](#)) to see changes in another workspace without leaving the current workspace; however, the `<table_name>_DIFF` view can be used for only two workspaces, whereas the `<table_name>_MW` view can be used for any number of workspaces. In addition, the `<table_name>_DIFF` view shows deleted rows, whereas the `<table_name>_MW` view does not show deleted rows.

For more information and several examples, see the information about the [SetMultiWorkspaces](#) procedure in [DBMS\\_WM Package: Reference](#).

# Part III

## Supplementary Information

This document has three parts:

- [Conceptual and Usage Information](#) provides conceptual and usage information about Workspace Manager.
- [Reference Information](#) provides reference information about the Workspace Manager PL/SQL API (DBMS\_WM package) and static data dictionary views.
- Part III provides supplementary information (appendixes and a glossary).

Part III contains the following:

- [Installing Workspace Manager with Custom Databases](#)
- [Workspace Manager Error Messages](#)
- [Glossary](#)

# A

## Installing Workspace Manager with Custom Databases

Workspace Manager is installed by default in the seed database and in all databases created by the Database Configuration Assistant (DBCA). However, in all other Oracle databases, such as those you create with a customized procedure, you must install Workspace Manager before you can use its features.

To install Workspace Manager in a custom database, do the following:

1. At the system command prompt, change the current directory to the directory that contains Workspace Manager installation script and packages, as shown in the following example:

```
cd <ORACLE_HOME>/rdbms/admin
```

2. Connect as `SYS` to the Oracle instance with a command in the following format:

```
sqlplus sys
```

Enter the password for the `SYS` account when you are prompted.

3. Run the `owminst.plb` script:

```
SQL> @owminst.plb
```

4. Verify the installation of Workspace Manager by entering the following command while connected as any valid database user, and ensure that the output is as shown here:

```
SQL> select dbms_wm.getWorkspace from dual;
```

```
GETWORKSPACE
```

```
-----  
LIVE
```

# B

## Workspace Manager Error Messages

This appendix lists the Workspace Manager error messages, including the cause and recommended user action for each.

**WM\_ERROR\_1 name of column '*string*' has more than 28 characters**

Cause: An attempt was made to version-enable a table that had a column with a name that has more than 28 characters.

Action: Ensure that all column names for the table are 28 characters or less.

**WM\_ERROR\_2 '*string*' is not allowed for workspace: '*string*' frozen in '*string*' mode**

Cause: An operation was executed on a workspace that was frozen.

Action: Unfreeze the workspace before retrying the operation.

**WM\_ERROR\_3 cannot modify primary key values for version-enabled table**

Cause: A DML operation that modifies one or more values in columns in the primary key constraint was performed on a version-enabled table.

Action: Do not perform DML operations on columns in the primary key constraints of version-enabled tables.

**WM\_ERROR\_4 There are open short transactions on this table.**

Cause: DisableVersioning failed because there were open database transactions on the table to be version-disabled.

Action: The user with the open database transaction should issue a standard database commit or rollback.

**WM\_ERROR\_5 integrity constraint ('*string*.'*string*') violated - child record found**

Cause: An attempt was made to delete or update a record in a parent table of a referential integrity constraint with the RESTRICT option, and there was a matching record in the child table of the integrity constraint. RESTRICT is a default property of a referential integrity constraint, the other being ON DELETE CASCADE, where the dependent rows in the child tables are deleted if corresponding rows in the parent table are deleted. The CASCADE option applies only to a deletion from the parent table. An update of the parent table always follows the RESTRICT option.

Action: Delete all matching records from the child table first.

**WM\_ERROR\_6 integrity constraint ('*string*.'*string*') violated - parent key not found**

Cause: An attempt was made to insert or update a record in a child table of a referential integrity constraint, and there was no matching record in the parent table of the integrity constraint.

Action: Insert a matching record in the parent table first.



**WM\_ERROR\_7 WM not found on the import platform**

Cause: Import of a version-enabled database failed because the import platform did not have Workspace Manager installed.

Action: Install Workspace Manager on the import platform and retry.

**WM\_ERROR\_8 the import platform cannot have any versioned tables**

Cause: Import of a version-enabled database failed because the import platform already had one or more version-enabled tables.

Action: The import platform may not have any version-enabled tables. A clean install of Workspace Manager is needed on the import platform.

**WM\_ERROR\_9 the import platform has non-"LIVE" workspaces or explicit savepoints**

Cause: Import of a version-enabled database failed because the import platform had either non-LIVE workspaces in the workspace hierarchy or explicit savepoints in the LIVE workspace.

Action: The import platform may have only the LIVE workspace and there may be no explicit savepoints. A clean install of Workspace Manager is needed on the import platform.

**WM\_ERROR\_10 unique key violation**

Cause: An insert operation failed because it violated the table's primary key constraint.

Action: Ensure that the primary key is not violated by the insert operation in the current workspace.

**WM\_ERROR\_11 need to be on the latest version to delete**

Cause: A delete operation failed because the delete was being made in a non-LATEST version of a workspace.

Action: Ensure that the current session is on the LATEST version in the workspace by using the GotoWorkspace or GotoSavepoint procedures.

**WM\_ERROR\_12 need to be on the latest version to insert**

Cause: An insert operation failed because the insert was being made in a non-LATEST version of a workspace.

Action: Ensure that the current session is on the LATEST version in the workspace by using the GotoWorkspace or GotoSavepoint procedures.

**WM\_ERROR\_13 need to be on the latest version to update**

Cause: An update operation failed because the update was made in a non-LATEST version of a workspace.

Action: Ensure that the current session is on the LATEST version in the workspace by using the GotoWorkspace or GotoSavepoint procedures.

**WM\_ERROR\_14 'string'. 'string' has not been version enabled**

Cause: This operation failed because it can only be invoked on a version-enabled table.

Action: Verify that the table is version-enabled.

**WM\_ERROR\_15 "/" is not allowed in a workspace name**

Cause: CreateWorkspace failed because the workspace name contained a "/".

Action: Choose another workspace name that does not contain a "/".

**WM\_ERROR\_16 "WM\_ADMIN" system privilege is required to version disable a table in another schema**

Cause: DisableVersioning failed because only a user with the WM\_ADMIN system privilege can version-disable a table in another schema.

Action: Ensure that the invoking user has the required privileges before attempting to version-disable this table. Otherwise, have the owner of the table version-disable it.

**WM\_ERROR\_17 "WM\_ADMIN" system privilege is required to version enable a table in another schema**

Cause: EnableVersioning failed because only a user with the WM\_ADMIN system privilege can version-enable a table in another schema.

Action: Ensure that the invoking user has the required privileges before attempting to version-enable this table. Otherwise, have the owner of the table version-enable it.

**WM\_ERROR\_18 "WM\_ADMIN" system privilege or ownership is required to alter workspace attributes**

Cause: AlterWorkspace failed because only a user with the WM\_ADMIN system privilege or the owner of the workspace can alter workspace attributes.

Action: Ensure that the invoking user has the required privileges before attempting to alter the workspace attributes. Otherwise, have the owner of the workspace alter the workspace attributes.

**WM\_ERROR\_19 "WM\_ADMIN" system privilege or ownership is required to freeze a workspace**

Cause: FreezeWorkspace failed because only a user with the WM\_ADMIN system privilege or the owner of the workspace can freeze a workspace.

Action: Ensure that the invoking user has the required privileges before attempting to freeze the workspace. Otherwise, have the owner of the workspace freeze it.

**WM\_ERROR\_20 "WM\_ADMIN" system privilege or ownership is required to set workspace lock mode**

Cause: SetWorkspaceLockModeOn failed because only a user with the WM\_ADMIN system privilege or the owner of the workspace can set the workspace lock mode.

Action: Ensure that the invoking user has the required privileges before attempting to set the workspace lock mode. Otherwise, have the owner of the workspace set the workspace lock mode.

**WM\_ERROR\_21 insufficient privileges to change savepoint attributes**

Cause: AlterSavepoint failed because only a user with the WM\_ADMIN system privilege or the owner of the workspace or savepoint can alter the savepoint attributes.

Action: Ensure that the invoking user has the required privileges before attempting to alter the savepoint attributes. Otherwise, have the workspace owner or the savepoint owner alter the savepoint attributes.

**WM\_ERROR\_22 insufficient privileges to delete savepoint**

Cause: DeleteSavepoint failed because only a user with the WM\_ADMIN system privilege or the owner of the workspace or savepoint can delete the savepoint.

Action: Ensure that the invoking user has the required privileges before attempting to delete the savepoint. Otherwise, have the workspace owner or the savepoint owner delete the savepoint.

**WM\_ERROR\_23 a workspace already exists with the name: 'string'**

Cause: CreateWorkspace failed because a workspace with the same name already existed in the system. Workspace Manager requires that workspace names be unique across the database.

Action: Choose another workspace name and retry.

**WM\_ERROR\_24 a workspace cannot be rolled back over an implicit savepoint**

Cause: A RollbackWorkspace operation was invoked on a non-leaf workspace across an implicit savepoint.

Action: Do not rollback over an implicit savepoint. To remove the implicit savepoint, merge or remove the descendant workspace.

**WM\_ERROR\_25 a table cannot be merged from the "LIVE" workspace**

Cause: MergeTable was invoked with the input workspace specified as the LIVE workspace. The LIVE workspace is the root workspace in the workspace hierarchy tree.

Action: Do not invoke MergeTable with the workspace parameter LIVE.

**WM\_ERROR\_27 a table cannot be refreshed to the "LIVE" workspace**

Cause: RefreshTable was invoked with the input workspace specified as the LIVE workspace. The LIVE workspace is the root workspace in the workspace hierarchy tree.

Action: Do not invoke RefreshTable with the workspace parameter LIVE.

**WM\_ERROR\_28 a table cannot be rolled back over an implicit savepoint**

Cause: A RollbackTable operation was invoked on a non-leaf workspace across an implicit savepoint.

Action: Do not rollback over an implicit savepoint. To remove the implicit savepoint, merge or remove the descendant workspace.

**WM\_ERROR\_29 cannot rollback this table using RollbackTable**

Cause: RollbackTable failed because the table to be rolled back is part of a referential integrity constraint.

Action: Use RollbackWorkspace or RollbackToSP instead.

**WM\_ERROR\_30 cannot merge this table using MergeTable**

Cause: MergeTable failed because the table to be merged is part of a referential integrity constraint.

Action: Use MergeWorkspace instead.

**WM\_ERROR\_31 All version enabled tables owned by 'string' must be disabled first.**

Cause: An attempt was made to drop a database user who owns one or more version-enabled tables.

Action: Ensure that all the version-enabled tables owned by the user have been explicitly disabled before attempting to drop the database user.

**WM\_ERROR\_32 An index-organized table cannot be version enabled.**

Cause: Workspace Manager does not support index-organized tables.

Action: Ensure the table to be version-enabled is not index-organized.

**WM\_ERROR\_33 attempt to 'string' a row locked by: 'string' in workspace 'string'**

Cause: A DML operation failed because the row was previously locked.

Action: Wait for the lock on the row to be released or have the lock owner use the UnlockRows procedure to unlock the row. Consult the table's `_LOCK` view to see which rows in this table are locked.

**WM\_ERROR\_34 attempt to 'string' a row locked by 'string' in workspace: 'string'**

Cause: A DML operation failed because the row was previously locked.

Action: Wait for the lock on the row to be released or have the lock owner use the UnlockRows procedure to unlock the row. Consult the table's `_LOCK` view to see which rows in this table are locked.

**WM\_ERROR\_35 attempt to lock a row locked in workspace: 'string'**

Cause: The operation failed because a lock could not be obtained on the row, since it was already locked.

Action: Wait for the lock on the row to be released or have the lock owner use the UnlockRows procedure to unlock the row. Consult the table's `_LOCK` view to see which rows in this table are locked.

**WM\_ERROR\_36 attempt to lock a row locked by 'string'**

Cause: The operation failed because a lock could not be obtained on the row, since it was already locked.

Action: Wait for the lock on the row to be released or have the lock owner use the UnlockRows procedure to unlock the row. Consult the table's `_LOCK` view to see which rows in this table are locked.

**WM\_ERROR\_37 attempt to modify a WM generated procedure**

Cause: An attempt to drop or re-create a database procedure failed because that procedure was created by Workspace Manager.

Action: Do not drop or re-create this procedure.

**WM\_ERROR\_38 cannot disable version a table modified in non-LIVE workspaces**

Cause: DisableVersioning failed because the table had been modified in non-LIVE workspaces.

Action: Remove or merge all workspaces that have modified this table. Otherwise, use the `FORCE` option of DisableVersioning.

**WM\_ERROR\_39 cannot drop tables involved in foreign key relationships**

Cause: An attempt to drop a database table failed because it was involved in a foreign key relationship with a version-enabled table.

Action: Consult the WM\_RIC\_INFO view and version-disable the table that is involved in the foreign key relationship before attempting to drop the table.

**WM\_ERROR\_40 only grantor of a privilege may revoke it**

Cause: An attempt was made to revoke a privilege that was not granted by the current user.

Action: Do not attempt to revoke this privilege.

**WM\_ERROR\_41 unable to set workspace lock mode**

Cause: SetWorkspaceLockModeOn failed because the workspace contained modifications from one or more version-enabled tables.

Action: Use SetLockingOn to set the session's lock mode. Use SetWorkspaceLockModeOn only for those workspaces that have not yet modified any version-enabled tables.

**WM\_ERROR\_42 cannot version enable tables owned by 'string'**

Cause: EnableVersioning failed because Workspace Manager cannot version-enable tables owned by the specified user.

Action: Do not invoke EnableVersioning on tables owned by that user.

**WM\_ERROR\_43 A continually refreshed workspace must be a leaf workspace.**

Cause: CreateWorkspace failed because the workspace to be created was to be a child of a continually refreshed workspace. Continually refreshed workspaces carry with them the restriction that they must be leaf workspaces.

Action: Do not create a workspace off of a continually refreshed workspace.

**WM\_ERROR\_44 insufficient privileges to merge data**

Cause: The merge operation failed because the user does not have both `ACCESS` and `MERGE` privileges on the workspace on which it was invoked; or, in a multiparent workspace environment, the user does not have both `ACCESS` and `MERGE` privileges on the non-root workspaces and `ACCESS` privilege on the root workspace of the multiparent workspace graph.

Action: Use the function `GetPrivs` to ensure that the user invoking this operation has the required privileges.

**WM\_ERROR\_45 merge operation requires ACCESS privileges on the parent workspace**

Cause: The operation invoked failed because it required `ACCESS` privileges on the parent workspace of the workspace it was invoked on.

Action: Use the function `GetPrivs` to ensure that the user invoking this operation has the required privileges on the parent workspace.

**WM\_ERROR\_46 commit/rollback open short transactions before calling CommitResolve**

Cause: `CommitResolve` failed because open database transactions existed.

Action: The user with the open database transaction should issue a standard database commit or rollback.

**WM\_ERROR\_47 commit/rollback open short transactions before calling CompressWorkspace**

Cause: CompressWorkspace failed because open database transactions existed.

Action: The user with the open database transaction should issue a standard database commit or rollback.

**WM\_ERROR\_48 commit/rollback open short transactions before calling CompressWorkspaceTree**

Cause: CompressWorkspaceTree failed because open database transactions existed.

Action: The user with the open database transaction should issue a standard database commit or rollback.

**WM\_ERROR\_49 commit/rollback open short transactions before calling DeleteSavepoint**

Cause: DeleteSavepoint failed because open database transactions existed.

Action: The user with the open database transaction should issue a standard data

**WM\_ERROR\_50 commit/rollback open short transactions before calling GotoWorkspace**

Cause: GotoWorkspace failed because open database transactions existed.

Action: The user with the open database transaction should issue a standard database commit or rollback.

**WM\_ERROR\_51 commit/rollback open short transactions before calling RollbackResolve**

Cause: RollbackResolve failed because open database transactions existed.

Action: The user with the open database transaction should issue a standard database commit or rollback.

**WM\_ERROR\_52 CommitResolve can be called only after BeginResolve has been invoked**

Cause: CommitResolve failed because BeginResolve was not previously invoked.

Action: To resolve conflicts, first issue a BeginResolve, then issue ResolveConflicts, and finally issue CommitResolve.

**WM\_ERROR\_53 CompressWorkspace operation requires ACCESS and MERGE privileges on the workspace**

Cause: The operation invoked failed because it required both `ACCESS` and `MERGE` privileges on the workspace on which it was invoked.

Action: Use the function `GetPrivs` to ensure that the user invoking this operation has the required privileges on the workspace.

**WM\_ERROR\_54 CompressWorkspace operation requires ACCESS privilege on the workspace**

Cause: The operation invoked failed because it required `ACCESS` privileges on the workspace on which it was invoked.

Action: Use the function GetPrivs to ensure that the user invoking this operation has the required privileges on the workspace.

**WM\_ERROR\_55 conflicts detected for workspace: 'string' in table: 'string'**

Cause: An operation failed because there were conflicts detected for the table.

Action: To resolve conflicts, first issue a BeginResolve, then issue ResolveConflicts, and finally issue CommitResolve. Otherwise, refrain from calling this operation.

**WM\_ERROR\_56 conflicts detected for workspace: 'string' in table: 'string'. 'string'**

Cause: An operation failed because there were conflicts detected for the table.

Action: To resolve conflicts, first issue a BeginResolve, then issue ResolveConflicts, and finally issue CommitResolve. Otherwise, refrain from calling this operation.

**WM\_ERROR\_57 CreateSavepoint operation requires ACCESS privileges on the workspace**

Cause: The operation invoked failed because it required ACCESS privileges on the workspace on which it was invoked.

Action: Use the function GetPrivs to ensure that the user invoking this operation has the required privileges on the workspace.

**WM\_ERROR\_58 RemoveWorkspace operation requires ACCESS and REMOVE privileges on the workspace**

Cause: The operation invoked failed because it required both ACCESS and REMOVE privileges on the workspace on which it was invoked.

Action: Use the function GetPrivs to ensure that the user invoking this operation has the required privileges on the workspace.

**WM\_ERROR\_59 entry already exists in spatial metadata table for 'string'\_WM**

Cause: EnableVersioning of the spatial table failed because the spatial metadata table already contained an entry for the table.

Action: Contact Oracle Support Services.

**WM\_ERROR\_60 user must call BeginResolve or have "WM\_ADMIN" system privilege to invoke RollbackResolve**

Cause: RollbackResolve can be successful only if the user invoking it also invoked BeginResolve, or if the user invoking it had the WM\_ADMIN system privilege.

Action: Ensure that the invoking user has the required privileges before attempting to invoke RollbackResolve. Otherwise, have the user that issued the BeginResolve operation invoke RollbackResolve.

**WM\_ERROR\_61 versioned objects have to be version disabled before being dropped**

Cause: An attempt to drop a database table or view failed because it was associated with a version-enabled table.

Action: version-disable the table first. In the case of a view, version-disable the table associated with the view.

**WM\_ERROR\_62 versioned table: 'string' does not exist**

Cause: The operation failed because the table passed in as input did not exist or was not version-enabled.

Action: Pass in an existing, version-enabled table as input.

**WM\_ERROR\_63 need to be on the latest version to create a continually refreshed workspace.**

Cause: CreateWorkspace failed because the session was in a non-LATEST version of the workspace.

Action: Ensure that the current session is on the LATEST version in the workspace by using the GotoWorkspace or GotoSavepoint procedures.

**WM\_ERROR\_64 need to be on the latest version to create a savepoint**

Cause: CreateSavepoint failed because the session was in a non-LATEST version of the workspace.

Action: Ensure that the current session is on the LATEST version in the workspace by using the GotoWorkspace or GotoSavepoint procedures.

**WM\_ERROR\_65 grantor and grantee may not be the same user**

Cause: An attempt was made to grant or revoke a privilege from or to the same user.

Action: Do not attempt to grant or revoke privileges from or to the same user. Privileges can only be granted or revoked between different users.

**WM\_ERROR\_66 unable to version enable this table with history option**

Cause: An attempt was made to version-enable a table with VIEW\_WO\_OVERWRITE or VIEW\_W\_OVERWRITE option and the cumulative length of the names of the primary key columns was greater than 600.

Action: Rename the primary key columns.

**WM\_ERROR\_67 grantee must be an existing user, an existing role or PUBLIC**

Cause: A grant operation was attempted with an invalid grantee parameter.

Action: The grantee may only be an existing user, role, or PUBLIC. Verify correct spelling of the grantee parameter.

**WM\_ERROR\_68 input parameter grant\_option must be "YES" or "NO"**

Cause: An attempt was made to invoke the GrantWorkspacePriv or GrantSystemPriv procedure with an invalid input parameter.

Action: Ensure that the valid parameters are passed to the GrantWorkspacePriv or GrantSystemPriv procedure. The grant\_option parameter may only be YES or NO.

**WM\_ERROR\_69 invalid in\_date time for GotoDate**

Cause: GotoDate was invoked with an in\_date time less than the create time of the current workspace.

Action: The in\_date parameter for GotoDate must be greater than or equal to the create time for the current workspace.



**WM\_ERROR\_70 insufficient privileges on 'string' to lock rows**

Cause: An attempt was made to invoke the LockRows procedure on a versioned table without the required privileges on the table.

Action: Ensure that the invoking user has the required privileges before invoking the operation. The lockRows procedure requires the invoking user to have SELECT, INSERT, UPDATE and DELETE privileges on the versioned table.

**WM\_ERROR\_71 insufficient privileges on 'string' to unlock rows**

Cause: An attempt was made to invoke the UnlockRows procedure on a versioned table without the required privileges on the table.

Action: Ensure that the invoking user has the required privileges before invoking the operation. The UnlockRows procedure requires the invoking user to have SELECT, INSERT, UPDATE and DELETE privileges on the versioned table.

**WM\_ERROR\_72 insufficient privileges on 'string'. 'string'**

Cause: An attempt was made to invoke the ResolveConflicts procedure on a versioned table without the required privileges on the table.

Action: Ensure that the invoking user has the required privileges before invoking the operation. The ResolveConflicts procedure requires the invoking user to have SELECT, INSERT, UPDATE and DELETE privileges on the versioned table being conflict resolved.

**WM\_ERROR\_73 insufficient privileges to ACCESS the workspace: 'string'**

Cause: An attempt was made to invoke an operation that required the specified privileges on the input workspace.

Action: Ensure that the invoking user has the required privileges before invoking the operation. Privileges can be granted using the GrantWorkspacePriv or the GrantSystemPriv procedures. Use the function GetPrivs to see which privileges you have on a workspace.

**WM\_ERROR\_74 insufficient privileges to ACCESS the parent workspace: 'string'**

Cause: An attempt was made to invoke an operation that required the specified privileges on the parent workspace of the input workspace.

Action: Ensure that the invoking user has the required privileges before invoking the operation. Privileges can be granted using the grantWorkspacePriv or the grantSystemPriv procedures. Use the function GetPrivs to see which privileges you have on a workspace.

**WM\_ERROR\_75 insufficient privileges to create a child workspace of: 'string'**

Cause: An attempt was made to invoke the CreateWorkspace procedure from a workspace without the required privileges on the workspace.

Action: Ensure that the invoking user has the required privileges before invoking the operation. The invoking user must have CREATE privileges on a workspace to be allowed to create a workspace off of it. Privileges can be granted using the grantWorkspacePriv or the grantSystemPriv procedures. Use the function GetPrivs to see which privileges you have on a workspace.

**WM\_ERROR\_76 insufficient privileges to grant 'string'**

Cause: An attempt was made to invoke the GrantWorkspacePriv or GrantSystemPriv procedure without the required privileges to do so.

Action: Ensure that the invoking user has the required privileges to grant the privilege. A user needs to have been granted a privilege with the GRANT option to be able to grant it to others.

**WM\_ERROR\_77 insufficient privileges on the versioned table 'string'**

Cause: An attempt was made to invoke a Workspace Manager procedure without the required privileges on the versioned table.

Action: Ensure that the invoking user has the required privileges before invoking the operation. All Workspace Manager workspace wide operations require the invoking user to have SELECT, INSERT, UPDATE and DELETE privileges on all versioned tables that were modified in the input workspace.

**WM\_ERROR\_78 insufficient privileges on the versioned table: 'string'. 'string'**

Cause: An attempt was made to invoke a Workspace Manager procedure without the required privileges on the versioned table.

Action: Ensure that the invoking user has the required privileges before invoking the operation. All Workspace Manager workspace wide operations require the invoking user to have SELECT, INSERT, UPDATE and DELETE privileges on all versioned tables that were modified in the input workspace.

**WM\_ERROR\_79 WM internal error ['string']**

Cause: A Workspace Manager operation resulted in an internal error.

Action: Contact Oracle Support Services to resolve the issue.

**WM\_ERROR\_80 invalid "hist" parameter for EnableVersioning**

Cause: An invalid value was specified for the hist parameter of procedure EnableVersioning.

Action: Valid values for the hist parameter are NONE, VIEW\_W\_OVERWRITE, and VIEW\_WO\_OVERWRITE.

**WM\_ERROR\_81 invalid column name specified in the where-clause**

Cause: An attempt was made to invoke a Workspace Manager procedure with an invalid where\_clause parameter as input.

Action: Ensure that the input where\_clause parameter contains only valid column names and has proper syntax.

**WM\_ERROR\_82 invalid privilege type: 'string' was specified as input**

Cause: An attempt was made to invoke a Grant or Revoke Privilege procedure with an invalid priv\_type parameter.

Action: Ensure that the valid parameters are passed to the Grant or Revoke Privilege operation. The valid privilege types are: ACCESS\_WORKSPACE, MERGE\_WORKSPACE, ROLLBACK\_WORKSPACE, REVOKE\_WORKSPACE, and CREATE\_WORKSPACE.

**WM\_ERROR\_83 invalid user specified for the freezewriter parameter**

Cause: The FreezeWorkspace procedure was called with an invalid freezewriter parameter.

Action: Ensure that the freezewriter parameter passed in as input to the FreezeWorkspace procedure is an existing database user.

**WM\_ERROR\_84 invalid value for lock\_mode - "E" or "S" expected**

Cause: An invalid value was specified for the `lock_mode` parameter of procedure LockRows.

Action: Specify a valid value for `lock_mode`. The valid values for `lock_mode` are `E` and `S` (default is `E`).

**WM\_ERROR\_85 invalid value for the lock\_mode argument - "E", "S" or "ES" expected**

Cause: An invalid value has been specified for the `lock_mode` parameter (fifth parameter) of procedure UnlockRows.

Action: Specify a valid value for `lock_mode`. The valid values for `lock_mode` are `E`, `S`, and `ES` (default is `ES`).

**WM\_ERROR\_86 invalid value for the all\_or\_user argument - "ALL" or "USER" expected**

Cause: An invalid value has been specified for the `all_or_user` parameter (fourth parameter) of procedure UnlockRows.

Action: Specify a valid value for `all_or_user`. The valid values for `all_or_user` are `ALL` and `USER` (default is `USER`).

**WM\_ERROR\_87 IsWorkspaceOccupied cannot be used for the 'string' workspace**

Cause: A user attempted to invoke `IsWorkspaceOccupied` on the default workspace.

Action: Workspace Manager allows `IsWorkspaceOccupied` to be invoked only on workspaces that are not the default workspace. The default workspace, as defined by the `DEFAULT_WORKSPACE` system parameter, is not monitored by Workspace Manager for users in the workspace. Do not invoke this method on the specified workspace.

**WM\_ERROR\_88 IsWorkspaceOccupied requires ACCESS privilege on the workspace**

Cause: `IsWorkspaceOccupied` was invoked for a workspace on which the user did not have `ACCESS` privilege.

Action: `IsWorkspaceOccupied` can only be invoked for a workspace on which the user has `ACCESS` privilege.

**WM\_ERROR\_89 "LIVE" workspace can be frozen only in (READ\_ONLY, 1WRITER, 1WRITER\_SESSION, WM\_ONLY) modes**

Cause: An attempt was made to Freeze the `LIVE` workspace in `NO_ACCESS` mode. Workspace Manager does not support this mode for the `LIVE` workspace.

Action: Use one of (`READ_ONLY`, `1WRITER`, `1WRITER_SESSION`, `WM_ONLY`) modes to freeze the `LIVE` workspace.

**WM\_ERROR\_90 lock operation requires ACCESS privilege on the parent workspace**

Cause: `LockRows` was invoked for a workspace whose parent workspace was not accessible to the user.

Action: The user requires `ACCESS` privilege on the parent workspace of the workspace for which `lockRows` is invoked.

**WM\_ERROR\_91 lock operation requires ACCESS privilege on the workspace**

Cause: `LockRows` was invoked for a workspace on which the user did not have `ACCESS` privilege.

Action: The user requires `ACCESS` privilege on the workspace for which `LockRows` is invoked.

**WM\_ERROR\_92 cannot 'string' because locking is on and row is already versioned**

Cause: An attempt to place a shared or exclusive lock on a row in a versioned table failed because the row was already versioned in some other workspace.

Action: To update, delete, or insert a row that was already versioned in some other workspace, the current session must turn locking off. Consult the table's `_LOCK` view to see which rows in this table are locked.

**WM\_ERROR\_93 The multi-workspace view requires ACCESS privilege on the workspace : 'string'**

Cause: `SetMultiWorkspaces` was invoked with the name of a workspace on which the user did not have `ACCESS` privilege.

Action: Names of only those workspaces for which the user has `ACCESS` privilege can be passed to `SetMultiWorkspaces`.

**WM\_ERROR\_94 non-existent versioned table: 'string'. 'string'**

Cause: This operation was invoked on a non-version-enabled table.

Action: This operation can only be invoked on a version-enabled table. Verify that the table is version-enabled. The `xxx_VERSIONED_TABLES` views show all the versioned tables in the database.

**WM\_ERROR\_95 null savepoint name passed in as input**

Cause: An attempt was made to invoke a Workspace Manager procedure with a null savepoint name parameter.

Action: Pass in a non-null savepoint parameter for this procedure to succeed

**WM\_ERROR\_96 null workspace name passed in as input**

Cause: A null value was passed in as input to a Workspace Manager operation

Action: Pass in a non-null workspace parameter for this operation to succeed

**WM\_ERROR\_97 null table name parameter passed in**

Cause: `MergeTable` was invoked with a null table name.

Action: Specify the name of the version-enabled table to be merged.

**WM\_ERROR\_98 Number of workspaces in the multi-workspace view cannot be greater than 8.**

Cause: `SetMultiWorkspaces` was invoked with more than 8 workspace names.

Action: Invoke `SetMultiWorkspaces` with 8 or fewer workspace names.

**WM\_ERROR\_99 WM failed to install - system triggers not properly created**

Cause: One of the Workspace Manager generated database triggers was not created properly.

Action: Contact Oracle Support Services to resolve the issue.

**WM\_ERROR\_100 '*string*' is both parent and child tables of referential integrity constraints**

Cause: An attempt was made to version-enable a table that was both parent and child tables of referential integrity constraints.

Action: Version-enable both tables (specifying a comma-delimited list of table names) in the same call to the [EnableVersioning](#) procedure.

**WM\_ERROR\_101 child table must be version enabled**

Cause: An attempt was made to version-enable the parent table of a referential integrity constraint whose child table was not version-enabled.

Action: Before version-enabling a table, all tables that are child tables of referential integrity constraints (excluding self referential integrity constraints) that have this table as the parent table must be version-enabled.

**WM\_ERROR\_102 cannot version enable this table**

Cause: An attempt was made to version-enable a table that was the child table of a non-self referential integrity constraint with the CASCADE option and that had a self referential integrity constraint defined on it.

Action: If application semantics permit, change the CASCADE option to the RESTRICT option.

**WM\_ERROR\_103 cannot version disable this table with force option**

Cause: The FORCE option was specified while version-disabling a table that was the parent table of a referential integrity constraint.

Action: The FORCE option cannot be specified while version-disabling a table that is the parent table of a referential integrity constraint. Commit or roll back all changes done on this table in non-LIVE workspaces and then version-disable the table without the FORCE option.

**WM\_ERROR\_104 cannot version disable this table**

Cause: An attempt has been made to version-disable the child table of a referential integrity constraint whose parent table was still version-enabled.

Action: Version-disable the parent table before version-disabling this table.

**WM\_ERROR\_105 owner of constraint ('*string*','*string*') must have select privilege on the parent**

Cause: An attempt was made to version-enable a table that was the child table of a referential integrity constraint with another table, and the owner of the table to be version-enabled did not have SELECT privilege on the parent table.

Action: Workspace Manager requires that before version-enabling the child table of a integrity constraint, the child table owner must have SELECT privilege on the parent table. Grant the required privilege before version-enabling.

**WM\_ERROR\_106 select and delete privileges needed on the child of constraint ('string'. 'string')**

Cause: An attempt was made to version-enable a table that was the parent table of a referential integrity constraint with another table and the owner of the table to be version-enabled did not have the specified privilege on the child table.

Action: Workspace Manager requires that before version-enabling the parent table of a referential integrity constraint, the parent table owner must have the specified privilege on the child table. Grant the privilege on the child table to the owner of the table being version-enabled.

**WM\_ERROR\_107 select privilege needed on the child of constraint ('string'. 'string')**

Cause: An attempt was made to version-enable a table that was the parent table of a referential integrity constraint with another table and the owner of the table to be version-enabled did not have the specified privilege on the child table.

Action: Workspace Manager requires that before version-enabling the parent table of a referential integrity constraint, the parent table owner must have the specified privilege on the child table. Grant the privilege on the child table to the owner of the table being version-enabled.

**WM\_ERROR\_108 triggering event 'string' not allowed**

Cause: A triggering event of the form "insert OR update OR delete" was specified.

Action: Drop the trigger and re-create separate triggers (with identical bodies) for insert, update, and delete.

**WM\_ERROR\_109 a table with unique constraints cannot be version enabled**

Cause: An attempt was made to version-enable a table that had unique constraints defined on it.

Action: Drop the unique constraint on this table before version-enabling it. If the table needs to have a index for performance reasons, create a non-unique index on the relevant set of columns. Oracle will use the created index to optimize queries on the version-enabled table whenever appropriate.

**WM\_ERROR\_112 insufficient privileges to refresh data**

Cause: The refresh operation failed because the user does not have both `ACCESS` and `MERGE` privileges on the child workspace on which it was invoked; or, in a multiparent workspace environment, the user does not have both `ACCESS` and `MERGE` privileges on the non-root workspaces and `ACCESS` privilege on the root workspace of the multiparent workspace graph.

Action: Use the `GetPrivs` function to ensure that the user invoking this operation has the required privileges.

**WM\_ERROR\_113 refresh operation requires ACCESS privileges on the parent workspace**

Cause: An attempt was made to invoke `RefreshTable` or `RefreshWorkspace` and the user did not have `ACCESS` privilege on the parent workspace.

Action: Ensure that the invoking user has `ACCESS` privilege on the parent workspace before invoking `RefreshTable` or `RefreshWorkspace`. Privileges can be granted using the `GrantWorkspacePriv` or the `GrantSystemPriv` procedures. Use the `GetPrivs` function to see which privileges the current user has on a workspace.

**WM\_ERROR\_114 Continually refreshed workspaces can be created only off of the "LIVE" workspace**

Cause: An attempt was made to create a continually refreshed workspace off a non-LIVE workspace.

Action: Workspace Manager only supports creation of continually refreshed workspaces off the LIVE workspace. The user needs to be in the LIVE workspace before invoking CreateWorkspace for creating a continually refreshed workspace.

**WM\_ERROR\_115 ResolveConflicts can be called only after BeginResolve is invoked**

Cause: The ResolveConflicts procedure was invoked without calling the BeginResolve procedure first.

Action: Ensure that BeginResolve is invoked by the current user on a workspace before invoking ResolveConflicts for a version-enabled table in that workspace. (See the Resolving Conflicts section of the User Guide for details on the process of resolving conflicts for version-enabled tables.)

**WM\_ERROR\_116 rollback operation requires ACCESS and ROLLBACK privileges on the workspace**

Cause: An attempt was made to invoke RollbackTable or RollbackWorkspace and the user did not have ACCESS or ROLLBACK privilege on the workspace.

Action: Ensure that the invoking user has ACCESS and ROLLBACK privileges on the workspace before invoking RollbackTable or RollbackWorkspace. Privileges can be granted using the GrantWorkspacePriv or the GrantSystemPriv procedures. Use the function GetPrivs to see which privileges the current user has on a workspace.

**WM\_ERROR\_117 RollbackResolve can be called only after BeginResolve has been invoked**

Cause: RollbackResolve procedure was invoked without calling the BeginResolve procedure first.

Action: Ensure that BeginResolve is invoked before you invoke RollbackResolve. (See the Resolving Conflicts section of the User Guide for details on the process of resolving conflicts for version-enabled tables.)

**WM\_ERROR\_118 savepoint names may not be longer than 30 characters**

Cause: An attempt was made to create a savepoint whose name had more than 30 characters.

Action: Choose a shorter savepoint name.

**WM\_ERROR\_119 savepoint names may not begin with "ICP-"**

Cause: An attempt was made to create a savepoint whose name began with the string "ICP-".

Action: Choose a savepoint name that does not begin with the string "ICP-". Workspace Manager reserves names starting with "ICP-" for naming implicit savepoints.

**WM\_ERROR\_120 savepoint: 'string' already exists in workspace: 'string'**

Cause: An attempt was made to create a savepoint with the same name as an existing savepoint. Workspace Manager savepoint names must be unique within a workspace.



Action: Choose another savepoint name.

**WM\_ERROR\_121 savepoint: 'string' does not exist in workspace: 'string'**

Cause: An attempt was made to invoke a Workspace Manager operation on a savepoint that did not exist in the specified workspace.

Action: Verify that the savepoint name is spelled correctly and that it exists in the specified workspace. Workspace names and savepoint names are case-sensitive.

**WM\_ERROR\_122 workspace 'string' does not exist**

Cause: An attempt was made to invoke a Workspace Manager operation on a workspace that did not exist.

Action: Pass in an existing workspace name as input. Workspace names and savepoint names are case-sensitive.

**WM\_ERROR\_123 workspace 'string' is currently frozen in 'string' mode**

Cause: The user invoked a Workspace Manager operation that cannot proceed because the specified workspace has been frozen in the specified mode.

Action: Wait for the database session that holds the lock to release the lock. See the Workspace Manager documentation for a description of the Workspace Manager operations allowed for different workspace freeze modes. Consult the `xxx_WM_WORKSPACES` view to see which workspaces are currently frozen.

**WM\_ERROR\_124 workspace name may not be "BASE"**

Cause: A user attempted to create a workspace with the name `BASE`.

Action: Workspace Manager considers "BASE" to be a reserved keyword. Therefore, Workspace Manager does not allow the workspace to be named `BASE`. Choose another workspace name.

**WM\_ERROR\_125 workspace name may not be "LIVE"**

Cause: A user attempted to create a workspace with the name `LIVE`.

Action: Workspace Manager considers "LIVE" to be a reserved keyword. Therefore, Workspace Manager does not allow new workspaces to be named `LIVE`. Choose another workspace name.

**WM\_ERROR\_126 workspace name may not exceed 30 characters**

Cause: A user attempted to create a workspace with the workspace name length greater than 30 characters.

Action: Workspace Manager limits workspace names to 30 characters. Choose a shorter workspace name.

**WM\_ERROR\_127 workspace: 'string' is already being conflict resolved by user: 'string'**

Cause: A user attempted to invoke `BeginResolve` on a workspace that was already being conflict resolved by some other user.

Action: Workspace Manager allows only one user to resolve conflicts for a workspace at the same time. Wait until the user is finished resolving conflicts in the workspace and verify that the conflicts you are attempting to resolve still exist. Use the `xxx_WORKSPACES` views to check on the current resolve status of the workspace.



**WM\_ERROR\_128 workspace: 'string' is temporarily frozen in an internal mode for a 'string' operation**

Cause: A user attempted to invoke a Workspace Manager operation on a workspace that was frozen internally for another Workspace Manager operation.

Action: Workspace Manager acquires internal freezes on workspaces for the duration of various Workspace Manager operations. Wait until Workspace Manager releases the internal freeze on the workspace. See the User Guide for details on the freezes that Workspace Manager acquires for various workspace-wide operations. Use the xxx\_WORKSPACES views to check on the current freeze status of the workspace.

**WM\_ERROR\_129 table 'string' does not exist**

Cause: An attempt was made to invoke a Workspace Manager operation on a table that did not exist.

Action: Verify that the table exists.

**WM\_ERROR\_130 table 'string' has been modified in an open transaction**

Cause: An attempt was made to execute a Workspace Manager operation that required that there be no open database transactions on the table.

Action: Ensure that all open database transactions on the specified table have completed before invoking the Workspace Manager operation.

**WM\_ERROR\_131 table 'string' is already version enabled**

Cause: The specified table is already version-enabled.

Action: To version-disable it, execute the DisableVersioning procedure. The xxx\_VERSIONED\_TABLES views show all the versioned tables in the database.

**WM\_ERROR\_132 table 'string' is not version enabled**

Cause: This operation can only be invoked on a version-enabled table.

Action: Verify that the specified table is version-enabled. The xxx\_VERSIONED\_TABLES views show all the versioned tables in the database.

**WM\_ERROR\_133 table 'string' needs to have a primary key**

Cause: An attempt was made to version-enable a table that did not have a primary key defined on it. Workspace Manager requires that a primary key exist on a version-enabled table.

Action: Add a primary key constraint on this table before version-enabling it.

**WM\_ERROR\_134 table 'string' is already being version disabled**

Cause: An attempt was made to version-disable a table which another transaction was in the process of version-disabling.

Action: Wait until the other transaction finishes version-disabling the specified table. The xxx\_VERSIONED\_TABLES views show all the versioned tables in the database.

**WM\_ERROR\_135 table 'string' is being version enabled**

Cause: An attempt was made to version-enable a table which another transaction was in the process of version-enabling.

Action: Wait until the other transaction finishes version-enabling the specified table. The xxx\_VERSIONED\_TABLES views show all the versioned tables in the database.

**WM\_ERROR\_136 table names are limited to 25 characters**

Cause: An attempt was made to version-enable a table whose name was longer than 25 characters, or to perform another action (such as an export from a staging table) where the table name was longer than 25 characters.

Action: Rename the table to a shorter table name.

**WM\_ERROR\_138 table: 'string' is in use in other sessions**

Cause: An attempt to version-disable a table has failed due to the existence of database transaction locks on the table.

Action: To successfully version-disable this table, verify that there are no database transaction locks on the table.

**WM\_ERROR\_140 invalid value for FreezeMode parameter**

Cause: An attempt was made to invoke the FreezeWorkspace procedure with an invalid `freezemode` parameter.

Action: The `freezemode` parameter for the FreezeWorkspace procedure must be one of (NO\_ACCESS, READ\_ONLY, 1WRITER, 1WRITER\_SESSION, WM\_ONLY). Ensure that FreezeWorkspace is invoked with the correct parameters.

**WM\_ERROR\_141 the parameter freezewriter can be non-null only for the 1WRITER mode**

Cause: An attempt was made to invoke the FreezeWorkspace procedure with an invalid `freezewriter` parameter.

Action: The `freezewriter` parameter for the FreezeWorkspace procedure can be non-null only when the `freezemode` parameter is 1WRITER. Ensure that FreezeWorkspace is invoked with the correct parameters.

**WM\_ERROR\_142 the keep parameter must be one of ("PARENT","CHILD","BASE")**

Cause: The ResolveConflicts procedure was called with an invalid `keep` parameter.

Action: Ensure that the `keep` parameter to the ResolveConflicts procedure is one of (CHILD,PARENT,BASE). This parameter is not case-sensitive. See the Resolving Conflicts section of the Workspace Manager documentation for details on the process of conflict resolution.

**WM\_ERROR\_143 the "LIVE" workspace can only be rolled back to a savepoint**

Cause: An attempt was made to rollback the entire LIVE workspace. Workspace Manager only supports the RollbackToSP operation for the LIVE workspace.

Action: Use RollbackToSP to achieve the desired result.

**WM\_ERROR\_144 the "LIVE" workspace cannot be merged**

Cause: A user attempted to invoke MergeWorkspace on the LIVE workspace.

Action: Workspace Manager disallows commit of the LIVE workspace. Do not invoke MergeWorkspace on the LIVE workspace.

**WM\_ERROR\_145 the "LIVE" workspace cannot be removed**

Cause: A user attempted to invoke RemoveWorkspace on the LIVE workspace.

Action: To rollback changes in the `LIVE` workspace, use the `RollbackToSP` operation. To remove descendants to the `LIVE` workspace, use the `RemoveWorkspace` operation on the child workspaces.

**WM\_ERROR\_147 the "LIVE" workspace cannot be refreshed**

Cause: A user attempted to invoke `RefreshWorkspace` on the `LIVE` workspace.

Action: Workspace Manager disallows the `Refresh` operation on the `LIVE` workspace. Do not invoke `RefreshWorkspace` on the `LIVE` workspace.

**WM\_ERROR\_148 the lock mode is currently not set for this session**

Cause: The user invoked a `SetLockingOFF` operation without having called `SetLockingON` earlier in the current session.

Action: A user can only execute `SetLockingOff` if the user had called `SetLockingOn` in the session. To see what the current lock mode is, use the `GetLockMode` function.

**WM\_ERROR\_149 the lock mode must be one of ("C","D","E","S","VE","WE")**

Cause: The user invoked a `SetLockingON` operation with an invalid `lockmode` parameter.

Action: Use a lock mode that Workspace Manager currently supports: `C`(carry-forward), `D` (disregard), `E` (exclusive), `S` (Shared), `VE` (version exclusive), or `WE` (workspace exclusive). For a discussion of the differences and similarities among these modes, see the Workspace Manager documentation.

**WM\_ERROR\_150 the lock mode is already set for workspace: 'string'**

Cause: An attempt was made to invoke the `SetWorkspaceLockModeON` operation for a workspace whose lock mode has already been set.

Action: To change the lock mode for a workspace, use the `SetWorkspaceLockModeOFF` procedure to first unset the lock mode.

**WM\_ERROR\_151 the parent workspace 'string' is currently frozen in 'string'mode**

Cause: An attempt was made to invoke a Workspace Manager operation that required the specified parent workspace to be unfrozen.

Action: Wait for the workspace to be unfrozen before invoking the Workspace Manager operation. The workspace can be unfrozen by the owner of the workspace, by a user with the `WM_ADMIN_ROLE`, or by a user with the `WM_ADMIN` system privilege using the `UnfreezeWorkspace` procedure.

**WM\_ERROR\_152 the workspace 'string' is not a leaf workspace**

Cause: A workspace wide operation was invoked on an intermediate workspace. Workspace Manager supports this operation only on leaf workspaces. A leaf workspace is one that does not have any descendants.

Action: Invoke the operation only on leaf workspaces.

**WM\_ERROR\_153 the workspace: 'string' has savepoints in the branch specified**

Cause: A `CompressWorkspace` or `CompressWorkspaceTree` operation resulted in this internal error.

Action: Contact Oracle Support Services to resolve the issue.

**WM\_ERROR\_154 the workspaceLockMode for 'string' has been set to 'string' without override**

Cause: An attempt was made to invoke the SetLockingON or the SetLockingOFF procedure while the current session was in a workspace whose lock mode was set without override.

Action: The lock mode can be changed by the current session only if the session is in a workspace whose lock mode has not been set or if the session is in a workspace whose lock mode has been set with the override option. Privileged users can change the lock mode for a workspace using the SetWorkspaceLockModeON and the SetWorkspaceLockModeOFF procedures.

**WM\_ERROR\_155 the where-clause can involve only primary key columns**

Cause: An attempt was made to invoke a Workspace Manager operation with an invalid `where_clause` parameter as input.

Action: Ensure that the input `where_clause` parameter contains only valid column names and has proper syntax. The `where_clause` parameter for this Workspace Manager operation can contain only columns that are part of the primary key.

**WM\_ERROR\_156 there are active sessions using the workspace: 'string'**

Cause: An attempt was made to invoke a Workspace Manager operation that required that there be no sessions in the specified workspace (or, in a multiparent workspace environment, in any non-root workspaces in the multiparent workspace graph).

Action: To successfully invoke the Workspace Manager operation on the specified workspace, ensure that there are no sessions in the workspace or workspaces involved. Privileged users can view all the sessions in a workspace using the `DBA_WORKSPACE_SESSIONS` view.

**WM\_ERROR\_157 there are sessions on non-latest versions in the workspace: 'string'**

Cause: An attempt was made to invoke `CompressWorkspace` with some sessions in the workspace being on non-LATEST savepoints in the workspace. `CompressWorkspace` requires that all sessions in the specified workspace be on the LATEST version of the workspace.

Action: All sessions in the specified workspace must either go to another workspace using `GotoWorkspace` or must go to the LATEST savepoint using `GotoSavepoint`. Privileged users can view all the sessions in a workspace using the `DBA_WORKSPACE_USERS` view.

**WM\_ERROR\_158 this procedure cannot be invoked on the "LIVE" workspace**

Cause: An attempt was made to invoke a Workspace Manager procedure on the LIVE workspace.

Action: Invoke this Workspace Manager procedure only on non-LIVE workspaces.

**WM\_ERROR\_159 unable to exclusively lock table: 'string'. 'string'**

Cause: An attempt to version-disable a table failed due to the existence of database transaction locks on the table.

Action: To successfully version-disable this table, verify that there are no database transaction locks on the table.

**WM\_ERROR\_160 unable to grant/revoke appropriate privileges**

Cause: An attempt to version-disable a table failed due to an internal error in granting or revoking appropriate privileges on the table being version-enabled.

Action: Contact Oracle Support Services to resolve the issue.

**WM\_ERROR\_161 unable to lock 'string': 'string' in 'string' mode**

Cause: An attempt was made to invoke a Workspace Manager operation that failed because Workspace Manager was unable to acquire an exclusive lock on the specified resource.

Action: The specified resource may have been locked by some other database session performing a Workspace Manager operation. Wait for the lock on the resource to be released before proceeding with the Workspace Manager operation.

**WM\_ERROR\_162 unlock operation requires ACCESS privilege on the workspace**

Cause: The user attempted to invoke the UnlockRows operation on a workspace without `ACCESS` privileges on the workspace.

Action: The UnlockRows operation requires `ACCESS` privileges on the workspace. Invoke the UnlockRows operation only on workspaces that you have `ACCESS` privileges for.

**WM\_ERROR\_163 use Commit/Rollback Resolve to unfreeze workspaces being conflict resolved**

Cause: A user attempted to invoke UnfreezeWorkspace on a workspace undergoing conflict resolution. This workspace was frozen due to a user having issued a BeginResolve operation on it.

Action: To unfreeze the workspace, call the CommitResolve or a RollbackResolve procedure. Only a user with the `WM_ADMIN_ROLE` role or the `WM_ADMIN` system privilege, or the user who initiated the BeginResolve operation on the workspace, can issue a CommitResolve or RollbackResolve call for that workspace.

**WM\_ERROR\_164 use the RemoveWorkspaceTree procedure to drop non-leaf workspaces**

Cause: A user attempted to invoke RemoveWorkspace on an intermediate workspace. To prevent the occurrence of orphaned workspaces, RemoveWorkspace can only be invoked on leaf workspaces.

Action: Execute the RemoveWorkspaceTree procedure to remove the workspace and all its descendants.

**WM\_ERROR\_165 use the force parameter to freeze a currently frozen workspace**

Cause: An attempt was made to invoke the FreezeWorkspace procedure for a workspace that was already frozen.

Action: To freeze workspaces that are already frozen, use the FreezeWorkspace procedure with the `force` parameter.

**WM\_ERROR\_166 only a BeginResolve invoker or a user with the "WM\_ADMIN" system privilege can call CommitResolve**

Cause: A user attempted to invoke CommitResolve without having initiated the BeginResolve operation earlier and without having the `WM_ADMIN` system privilege.

Action: CommitResolve can be invoked only by the user who initiated the BeginResolve operation or by a user who has the `WM_ADMIN` system privilege.

**WM\_ERROR\_167 null lockMode parameter passed in**

Cause: A user called a procedure that requires that the `lockMode` parameter have a non-null value.

Action: The user must pass in a non-null `lockmode` parameter for this operation to succeed.

**WM\_ERROR\_168 Cannot disable workspace lockmode**

Cause: An attempt was made to disable the workspace lock mode, but this is not supported.

Action: Do not attempt to turn off locking for this workspace.

**WM\_ERROR\_169 "WM\_ADMIN" system privilege or ownership is required to UnFreeze a workspace**

Cause: `UnfreezeWorkspace` failed because only a user with the `WM_ADMIN_ROLE` role or `WM_ADMIN` system privilege or the owner of the workspace can unfreeze a frozen workspace.

Action: Ensure that the invoking user has the required privileges before attempting to unfreeze the workspace. Otherwise, have the owner of the workspace unfreeze it.

**WM\_ERROR\_170 The row to be locked has already been versioned**

Cause: `LockRows` failed because the row specified to be locked was already versioned.

Action: Do not attempt to lock rows that have already been versioned. Use the `where_clause` parameter of `LockRows` to specify those rows that have not already been versioned.

**WM\_ERROR\_171 WM error: 'string'**

Cause: A Workspace Manager error occurred.

Action: See the Workspace Manager documentation.

**WM\_ERROR\_172 all version enabled tables have to be disabled before uninstalling**

Cause: An attempt was made to uninstall Workspace Manager with existing version-enabled tables.

Action: Version-disable all version-enabled tables before attempting to uninstall Workspace Manager. Version-enabled tables can be disabled using the `DisableVersioning` procedure.

**WM\_ERROR\_173 cannot create workspaces that are more than 30 levels deep**

Cause: An attempt was made to create a workspace that is more than 30 levels in depth from the `LIVE` workspace.

Action: Do not create workspaces that are more than 30 levels in depth from the `LIVE` workspace.

**WM\_ERROR\_174 table: 'string' contains columns with unsupported data types**

Cause: An attempt was made to version-enable a table with one or more columns with an unsupported data type.

Action: Ensure that all the columns in the table being version-enabled are of supported data types. The currently unsupported data types for version-enabled tables are: `LONG` and `LONG RAW`.

**WM\_ERROR\_175 cannot delete implicit savepoints with dependent child workspaces**

Cause: An attempt was made to invoke the `DeleteSavepoint` procedure on an implicit savepoint with dependent child workspaces.

Action: Ensure that the savepoint being deleted is not implicit or it does not have any child workspaces created off of it. The `xxx_WORKSPACES` views show the parent savepoints for all the workspaces in the system. Ensure that the savepoint being deleted is not a parent savepoint for some workspace.

**WM\_ERROR\_176 A user-defined trigger defined on '*string*.'*string*' has compilation errors.**

Cause: An attempt was made to version-enable a table that has a user-defined trigger with compilation errors defined on it.

Action: Ensure that all user-defined triggers on the table to be version-enabled have no compilation errors.

**WM\_ERROR\_177 sum of length of all column names of '*string*.'*string*' exceeds 8250 characters**

Cause: An attempt was made to version-enable a table where the sum of the column name lengths exceeded 8250 characters.

Action: Rename some of the table's columns to reduce the sum of the column name lengths.

**WM\_ERROR\_178 user-defined trigger body defined on '*string*.'*string*' exceeds 28000 characters**

Cause: An attempt was made to version-enable a table that has a user-defined trigger with a trigger body length of more than 28,000 characters defined on it.

Action: Ensure that all user-defined triggers on the table to be version-enabled have trigger body lengths that are less than 28,000 characters.

**WM\_ERROR\_179 combination of column name sizes and user-defined trigger lengths too large**

Cause: An attempt was made to version-enable a table where the length of all of the column names combined with the length of the largest trigger body defined on the table was too large.

Action: Reduce the length of the largest trigger body defined on this table, rename some of the table's columns to reduce the sum of the column name lengths, or do both.

**WM\_ERROR\_180 table '*string*.'*string*' has too many primary key columns**

Cause: An attempt was made to version-enable a table that has more than 31 primary key columns.

Action: Decrease the number of primary key columns on the table to 31, at most.

**WM\_ERROR\_181 attempt to modify a WM generated trigger**

Cause: An attempt to drop or re-create a database trigger failed because that trigger was created by Workspace Manager.

Action: Do not drop or re-create this trigger.

**WM\_ERROR\_182 attempt to modify a WM generated view**

Cause: An attempt to re-create a database view failed because it was associated with a version-enabled table.

Action: Do not re-create this view. The view will automatically be dropped when the table associated with it is version-disabled.



**WM\_ERROR\_183 reserved column name found**

Cause: An attempt to version-enable the table failed because it had a column whose name started with WM\$ or WM\_.

Action: Rename the column to a different name.

**WM\_ERROR\_184 reserved index name found**

Cause: An attempt to version-enable the table failed because it had an index on it with the index name being the name of the table (to version-enabled) with the prefix \_PKI\$ or \_TI\$.

Action: Re-create the index using a different name.

**WM\_ERROR\_185 operation disallowed on workspace '*string*' involved in a conflict resolution session**

Cause: An attempt was made to execute an operation on a Workspace undergoing conflict resolution. A Workspace is under conflict resolution if BeginResolve method has been called on the workspace but CommitResolve or RollbackResolve has not been called yet.

Action: Wait for conflict resolution to either commit or rollback before trying the operation on the workspace.

**WM\_ERROR\_186 the parameter freezewriter must be null when session\_duration is true',**

Cause: An attempt was made to invoke the FreezeWorkspace procedure with an invalid `freezewriter` parameter.

Action: The `freezewriter` parameter of the FreezeWorkspace procedure must be null whenever the `session_duration` parameter is TRUE. The `freezewriter` is implicitly assumed to be the currently connected session. Ensure that FreezeWorkspace is invoked with the correct parameters.

**WM\_ERROR\_187 the parameter session\_duration must be true for the 1WRITER\_SESSION mod**

Cause: An attempt was made to invoke the FreezeWorkspace procedure with an invalid `session_duration` parameter.

Action: The `session_duration` parameter of the FreezeWorkspace must be TRUE when attempting to freeze a workspace in 1WRITER\_SESSION mode. Ensure that FreezeWorkspace is invoked with the correct parameters.

**WM\_ERROR\_188 At least one table failed during lwDisableVersioning. Please query all\_wm\_vt\_errors view to get the errors '*string*'**

Cause: If lightweight disable-versioning fails for some reason during the upgrade or downgrade.

Action: Contact Oracle Support Services with the upgrade or downgrade log.

**WM\_ERROR\_189 workspaces, savepoints, or versioned tables cannot be present on the IMPORT platform**

Cause: The instance you are importing Workspace Manager into has some savepoints, workspaces, or version-enabled tables.

Action: Clean up savepoints, workspaces and version-enabled tables, or reinstall Workspace Manager before importing other Workspace Manager data.



**WM\_ERROR\_190 table '*string*' is in mutating state, no structural operations can be performed**

Cause: When a structural operation (for example, DisableVersioning) is in progress on a table, another structural operation (for example, BeginDDL) was invoked.

Action: Complete the ongoing operation before calling a new one.

**WM\_ERROR\_191 LWDisableVersioning not called on the table '*string*'**

Cause: Internal error during the upgrade or downgrade.

Action: Contact Oracle Support Services with the upgrade or downgrade output log.

**WM\_ERROR\_192 At least one table failed during temporary disable-versioning '*string*'**

Cause: Internal error during downgrade.

Action: Contact Oracle Support Services with downgrade output log.

**WM\_ERROR\_194 At least one table failed during lwEnableVersioning. Please query all\_wm\_vt\_errors view to get the errors '*string*'**

Cause: Lightweight enable-versioning failed for some reason during the upgrade or downgrade.

Action: Internal error: contact Oracle Support Services with the upgrade or downgrade output log.

**WM\_ERROR\_195 Following tables with VIEW\_WO\_OVERWRITE failed during recreation of PRIMARY KEY constraint '*string*'**

Cause: The primary key constraint could not be re-created during the upgrade or downgrade.

Action: Internal error: contact Oracle Support Services with the upgrade or downgrade log.

**WM\_ERROR\_196 '*string*' operation requires "FREEZE\_WORKSPACE" privilege on the workspace or "FREEZE\_ANY\_WORKSPACE" or "WM\_ADMIN" system privilege**

Cause: Insufficient privilege for freezing or unfreezing a workspace.

Action: Grant `FREEZE_WORKSPACE` privilege on the workspace, or `FREEZE_ANY_WORKSPACE`, the `WM_ADMIN_ROLE` role, or the `WM_ADMIN` system privilege to the user trying the operation.

**WM\_ERROR\_197 a ddl operation is being committed on '*string*'**

Cause: A DDL operation is in the process of being committed on the table.

Action: Wait until the DDL operation is complete and then retry the current operation.

**WM\_ERROR\_198 primary key constraint of a version enabled table cannot be renamed**

Cause: An attempt was made to rename the primary key constraint of the skeleton table associated with a version-enabled table.

Action: Rename the primary key constraint of the skeleton table to its original name and call the `CommitDDL` procedure again.

**WM\_ERROR\_199 primary key columns cannot be added/dropped/modified/reordered for version enabled tables**

Cause: An attempt was made to add, drop, modify, or reorder the primary key columns of the skeleton table associated with a version-enabled table.

Action: Restore the primary key columns to their original state and call the [CommitDDL](#) procedure again.

**WM\_ERROR\_200 unsupported constraint '*string*' detected**

Cause: A check or unique constraint was detected on the skeleton table associated with a version-enabled table.

Action: Check or unique constraints cannot be defined on a version-enabled table. Remove the constraint from the skeleton table and call the [CommitDDL](#) procedure again.

**WM\_ERROR\_201 creation of partitioned/join indexes on version enabled tables is not supported**

Cause: A partitioned or join index was detected on the skeleton table associated with a version-enabled table.

Action: Drop all partitioned or join indexes on the skeleton table and call the [CommitDDL](#) procedure again.

**WM\_ERROR\_202 index name '*string*' is longer than 26 characters**

Cause: An index name with more than 26 characters was detected on the skeleton table associated with a version-enabled table.

Action: Rename the index and call the [CommitDDL](#) procedure again.

**WM\_ERROR\_203 enable/disable versioning or begin/commitDDL is being executed on '*string*'**

Cause: Versioning is being enabled or disabled, or BeginDDL or CommitDDL is being executed, on this table.

Action: Wait until the version-enabling, version-disabling, or BeginDDL or CommitDDL operation is complete, and then retry the current operation. The operation getting executed on the table can be found by querying the ALL\_WM\_VERSIONED\_TABLES view.

**WM\_ERROR\_204 beginDDL not called on '*string*'**

Cause: BeginDDL needs to be executed on the table before the current operation can be performed.

Action: Call BeginDDL on the table and then perform the current operation again.

**WM\_ERROR\_205 '*string*' contains data - cannot be modified**

Cause: A column of the skeleton table associated with a version-enabled table was modified, and the versioned table contains non-null data in this column.

Action: Restore the column to its original state and call the [CommitDDL](#) procedure again.

**WM\_ERROR\_206 column reordering is not supported**

Cause: Columns of the skeleton table associated with a version-enabled table were reordered.

Action: Restore the columns to their original state and call the [CommitDDL](#) procedure again. Reordering of columns can be achieved by first dropping columns in a DDL session and then adding columns in a subsequent DDL session.

**WM\_ERROR\_207 referential integrity constraint exists with a table not contained in the list of specified tables**

Cause: A referential integrity constraint exists with a table not contained in the list of tables passed to the `EnableVersioning` or `DisableVersioning` procedure.

Action: Add the table to the list passed to enable or disable versioning. If you do not want to version-enable this table not contained in the list, you need to version-enable the tables one at a time.

**WM\_ERROR\_208 cycle detected in referential integrity constraints on specified tables**

Cause: A cycle exists in the referential integrity constraints between tables passed to enable or disable versioning or a new referential constraint added between two skeleton tables caused a cycle in the referential constraints.

Action: Drop one of the referential constraints in the cycle and implement it using user-defined triggers.

**WM\_ERROR\_209 table '*string*' has been modified in non-LIVE workspaces**

Cause: `DisableVersioning` failed because the table had been modified in non-LIVE workspaces.

Action: Remove or merge all workspaces that have modified this table. Otherwise, use the `FORCE` option of `DisableVersioning`.

**WM\_ERROR\_210 multi-level referential integrity constraint with cascade option detected**

Cause: `DisableVersioning` failed because the table has a cascade referential constraint with a version-enabled child table that, in turn, is the parent table of another referential constraint.

Action: Version-disable the child and parent tables together.

**WM\_ERROR\_211 DDL is being done on '*string*'**

Cause: A DDL session has already been started on the table.

Action: Wait until the previous DDL session has been committed or rolled back.

**WM\_ERROR\_212 deferrable option not supported for integrity constraints**

Cause: Deferrable option is not supported for referential integrity constraints defined on version-enabled tables.

Action: Re-create any referential constraints that have the deferrable option so that they do not have the deferrable option.

**WM\_ERROR\_213 unsupported referential constraint with '*string*' detected**

Cause: The skeleton table associated with a version-enabled table has a referential constraint with a table that is not a skeleton table.

Action: Drop this referential constraint. You can only define referential constraints between two skeleton tables.

**WM\_ERROR\_214 'string' has a cascade referential constraint with a non-version enabled table**

Cause: A new referential integrity constraint was added between the skeleton tables of two version-enabled tables, but the parent table already had a cascading referential constraint with a table that is not version-enabled.

Action: Drop the new referential integrity constraint between the skeleton tables and perform the current operation again.

**WM\_ERROR\_215 A savepoint cannot be created with the name "LATEST"**

Cause: A savepoint cannot be named "LATEST".

Action: Choose another name for the savepoint.

**WM\_ERROR\_220 Following tables failed during sentinel row adjustment 'string'**

Cause: An error occurred when Workspace Manager was being migrated from one version to another.

Action: Examine the spool file to find the Oracle error that caused this error to occur. Correct the error and enter the following SQL statement while connected AS SYSDBA: `SQL> EXECUTE WMSYS.OWM_MIG_PKG.AllFixSentinelVersion;`

**WM\_ERROR\_221 'string' could not be recovered from Migration Error: ['string']**

Cause: An error occurred when Workspace Manager was being migrated from one version to another.

Action: The ALL\_WM\_VT\_ERRORS view can be queried for more detailed information about the error. The RecoverMigratingTable or RecoverAllMigratingTables procedures can be used to recover one or more tables that were left in an inconsistent state.

**WM\_ERROR\_222 Following tables could not be recovered from Migration Error: 'string'**

Cause: An error occurred when Workspace Manager was being migrated from one version to another.

Action: The ALL\_WM\_VT\_ERRORS view can be queried for more detailed information about the error. The RecoverMigratingTable or RecoverAllMigratingTables procedures can be used to recover one or more tables that were left in an inconsistent state.

**WM\_ERROR\_223 "WM\_ADMIN" system privilege is required to invoke this procedure**

Cause: A Workspace Manager operation was invoked without the requisite privileges.

Action: The WM\_ADMIN system privilege or WM\_ADMIN\_ROLE role is required to invoke this specific operation. Ensure that the current user has the required privileges to invoke this operation.

**WM\_ERROR\_226 Nested Tables must have a primary key constraint**

Cause: An attempt was made to execute EnableVersioning on a table that contained a nested table column that did not have a primary key.

Action: Add a primary key constraint to any nested table column contained in the table to be version-enabled.

**WM\_ERROR\_228 this operation is not allowed for table '*string*' with version state '*string*'**

Cause: An attempt was made to invoke a workspace operation on a table with a version state that is invalid.

Action: The table on which the operation was invoked has a version state that disallows the operation from being performed. Query the ALL\_WM\_VERSIONED\_TABLES view to look up the version state for the specified table, and see the documentation for the ALL\_WM\_VERSIONED\_TABLES view (in [ALL\\_WM\\_VERSIONED\\_TABLES](#)) for a definition of the possible version state values.

**WM\_ERROR\_229 statement '*string*' failed during EnableVersioning. Error: *string*'**

Cause: Version-enabling of the table failed due to some error. This may occur due to insufficient resources or some unexpected Oracle error.

Action: Retry the operation after fixing the cause of the error.

**WM\_ERROR\_230 table '*string*' failed during UndoEnableVersioning/DisableVersioning. Error: *string*'**

Cause: If EnableVersioning fails for some reason, an attempt is made to bring back the table to original state. This error occurs when this undo attempt fails on the partially versioned tables.

Action: Check the ALL\_WM\_VT\_ERRORS view to see the statement that failed and the error that occurred. After fixing the cause of the error, you can version-enable the table using [EnableVersioning](#) or disable versioning on the table using [DisableVersioning](#). (Be careful if you specify 'ignore\_last\_error => TRUE' with [DisableVersioning](#).)

**WM\_ERROR\_231 table '*string*' failed during DisableVersioning. Error: *string*'**

Cause: DisableVersioning of the table failed due to some error. This may occur due to insufficient resources or some unexpected Oracle error.

Action: See the Usage Notes for the [DisableVersioning](#) procedure for information about handling the error.

**WM\_ERROR\_232 unique constraint '*string*.'*string*' violated'**

Cause: The DML operation or workspace operation violated the unique constraint '*string*.'*string*' on a version-enabled table.

Action: Find the row that violates the constraint, and attempt the operation without the row.

**WM\_ERROR\_233 deadlock detected when trying to acquire lock for '*string*': '*string*', session may have open database transactions'**

Cause: The workspace operation with an `auto_commit` value of `TRUE` is invalid if the current session has an open database transaction on that workspace.

Action: Commit or roll back the current database transaction before invoking the procedure, or invoke the procedure with an `auto_commit` value of `FALSE`.

**WM\_ERROR\_234 continually-refreshed workspaces may have only continually-refreshed workspaces as children**

Cause: An attempt was made to create a workspace that is not continually refreshed as a child of a continually refreshed workspace

Action: Continually refreshed workspaces can have only continually refreshed workspaces as child workspaces.

**WM\_ERROR\_235 invalid system parameter name or value**

Cause: An invalid string was passed as a name or value for a parameter for GetSystemParameter or SetSystemParameter.

Action: Check the documentation for valid names and values of Workspace Manager system parameters.

**WM\_ERROR\_236 system setting does not allow invocation of this procedure'**

Cause: UnlockRows cannot be called if the Workspace Manager system parameter NONCR\_WORKSPACE\_MODE is set to PESSIMISTIC\_LOCKING.

Action: If no data exists in workspaces that are not continually refreshed, you can set NONCR\_WORKSPACE\_MODE is set to OPTIMISTIC\_LOCKING. To see the current Workspace Manager system parameter settings, use the WM\_INSTALLATION metadata view.

**WM\_ERROR\_237 integrity constraint ('string'.*string*) violated in workspace 'string' or one of its descendants - child record found**

Cause: An attempt was made to delete or update a record in a parent table of a referential integrity constraint with the RESTRICT option, and there was a matching record in the child table of the integrity constraint in the identified workspace or one of its continually refreshed descendant workspaces.

Action: Delete or roll back matching child table records first.

**WM\_ERROR\_238 integrity constraint ('string'.*string*) violated in workspace 'string' or one of its descendants - parent key not found**

Cause: An attempt was made to insert or update a record in a child table of a referential integrity constraint, and there was no matching record in the parent table of the integrity constraint in the identified workspace or one its continually refreshed descendant workspaces.

Action: Insert a matching record in the parent table or roll back deleted matching parent table records first.

**WM\_ERROR\_239 integrity constraint ('string'.*string*) violated in a descendant workspace - parent key not found**

Cause: An attempt was made to insert or update a record in a child table of a referential integrity constraint, and there was no matching record in the parent table of the integrity constraint in a continually refreshed descendant workspace of the current workspace.

Action: Insert a matching record in the parent table or roll back deleted matching parent table records first.

**WM\_ERROR\_240 reserved character found in workspace name**

Cause: The name of a workspace contains one or more of these characters: "/", "\*", ",", "\$", "#"

Action: Remove these characters or replace them with valid characters.

**WM\_ERROR\_241 system parameter 'string' should be set to 'string' for multiparent functionality'**

Cause: The Workspace Manager system parameter is not set correctly to allow multiparent workspaces.

Action: Check the documentation about Workspace Manager system parameters, and be sure that any values required for multiparent workspace support are set correctly.

**WM\_ERROR\_242 '*string*' already in ancestor hierarchy of '*string*'**

Cause: The workspace that is being added as a parent workspace is already an ancestor of the (child) workspace.

Action: Ensure that a workspace is not already an ancestor of a workspace to which it is to be added as a parent workspace.

**WM\_ERROR\_243 all workspaces under the root of multiparent graph must be same type'**

Cause: In a multiparent workspace graph, all workspaces must be either continually refreshed or not continually refreshed.

Action: Ensure that the workspaces under the root of a multiparent graph are either all continually refreshed or all not continually refreshed. You can use the ChangeWorkspaceType procedure to change the workspace type between continually refreshed and not continually refreshed.

**WM\_ERROR\_244 AddAsParentWorkspace operation requires ACCESS privilege on all nodes except root in the graph and CREATE privilege on the new parent workspace'**

Cause: The user in the multiparent workspace environment does not have the specified privileges.

Action: Use the function GetPrivs to ensure that the user invoking this operation has the required privileges.

**WM\_ERROR\_245 '*string*' is not multi-parent of '*string*'**

Cause: For a RemoveWorkspaceAsParent operation, the workspace to be removed was not previously added as a parent workspace.

Action: Ensure that you have specified the correct workspace.

**WM\_ERROR\_246 '*string*' cannot be removed because data has been versioned from the workspace branch being removed**

Cause: In a multiparent workspace environment, if data has been versioned in the multiparent leaf workspace from any of the workspaces that will be removed as ancestors by this operation, the operation is not allowed.

Action: Roll back the leaf workspace to remove the versioned data from the branch being removed.

**WM\_ERROR\_247 the multi-parent graph formed by '*string*' is not a leaf graph**

Cause: In a multiparent workspace environment, there is a workspace that is a child of some non-root workspace of the multi-parent graph on which this operation was invoked.

Action: Remove all the workspaces that are children of non-root workspaces of the graph before performing this operation.

**WM\_ERROR\_248 intermediate workspaces of a multiparent graph cannot be refreshed**

Cause: In a multiparent workspace environment, only the leaf workspace of a multiparent graph can be refreshed.



Action: Ensure that you are refreshing the correct workspace.

**WM\_ERROR\_249 primary key constraint violated for '*string.string*'**

Cause: In a multiparent workspace environment, the primary key constraint for the table is violated, as viewed from the leaf workspace of the multiparent graph.

Action: Delete or roll back one of the rows that is shown as a duplicate.

**WM\_ERROR\_250 workspace name may not be "NULL"**

Cause: An attempt was made to name a workspace "NULL".

Action: Choose another name for the workspace.

**WM\_ERROR\_251 attempt to '*string*' a row locked by '*string*' in workspace '*string*' in mode "WE"**

Cause: Only the user that locked the row in WE mode can further edit the row in the same workspace.

Action: The row cannot be edited by the current user until the locking user removes the lock on the row.

**WM\_ERROR\_252 attempt to '*string*' a row locked by '*string*' in mode "VE"**

Cause: Only the user who locked the row in VE mode can further edit the row.

Action: The row cannot be edited by the current user until the locking user removes the lock on the row.

**WM\_ERROR\_253 lock\_mode of only "VE" is allowed when workspace is "NONE"**

Cause: A value of NONE for the workspace parameter is permitted only with VE as the value for lock\_mode.

Action: Specify the name of an existing workspace when specifying a lock\_mode value other than VE.

**WM\_ERROR\_254 cannot '*string*' because PESSIMISTIC\_LOCKING is on and row is already versioned'**

Cause: The DML operation cannot be executed because system parameter CR\_WORKSPACE\_MODE or NONCR\_WORKSPACE\_MODE is set to PESSIMISTIC\_LOCKING and the DML operation violates the system setting.

Action: If data has not been versioned in non-LIVE workspaces, you can change the PESSIMISTIC\_LOCKING setting to OPTIMISTIC\_LOCKING. To see the current Workspace Manager system parameter settings, use the WM\_INSTALLATION metadata view.

**WM\_ERROR\_255 insufficient privileges [*string*]**

Cause: An attempt was made to invoke an import or export operation without the required privileges.

Action: Ensure that the user has the required privileges before invoking the operation. To import from or export to a staging table, the user must have privileges to select from and perform DML operations on the staging table.

**WM\_ERROR\_256 '*string*' cannot be invoked with a null '*string*' parameter**

Cause: The specified parameter cannot be null.

Action: Reissue the operation using a non-null value for the specified parameter.



**WM\_ERROR\_257 savepoint '*string*' does not exist in '*string*'s hierarchy**

Cause: The ancestor savepoint for an import operation does not exist in the hierarchy of the workspace.

Action: Specify a savepoint that is contained in the workspace's hierarchy.

**WM\_ERROR\_258 specified system where clause is invalid [*string*"]**

Cause: An import or export operation was invoked with an invalid system `WHERE` clause.

Action: Ensure the compatibility of the system `WHERE` clause in conjunction with the parameters for the operation.

**WM\_ERROR\_259 table '*string*' is invalid [*string*]**

Cause: The staging table has been modified from its original state required for an import or export operation.

Action: Restore the staging table to its original state.

**WM\_ERROR\_260 Export error [*string*]**

Cause: Unable to perform the export operation due to the specified error.

Action: Fix the error and retry the appropriate operation.

**WM\_ERROR\_261 Import error [*string*]**

Cause: Unable to perform the import operation due to the specified error.

Action: Fix the error and retry the appropriate operation.

**WM\_ERROR\_262 this parameter cannot be set to 'PESSIMISTIC\_LOCKING' if data has been versioned in Non-LIVE workspaces'**

Cause: This setting is not permitted if data exists in non-LIVE workspaces.

Action: If you want to use the `PESSIMISTIC_LOCKING` setting, ensure that there is no data versioned in non-LIVE workspaces for the workspace type (continually refreshed or not continually refreshed) for which the parameter is being set.

**WM\_ERROR\_272 invalid event name: '*string*'**

Cause: An invalid Workspace Manager event name was passed as an argument to the function.

Action: Pass a valid event name. See the `WM_EVENTS_INFO` view for a list of all valid events.

**WM\_ERROR\_273 set system parameter 'ALLOW\_CAPTURE\_EVENTS' to 'ON' for capturing events**

Cause: An attempt was made to capture an event even though the Workspace Manager system parameter `ALLOW_CAPTURE_EVENTS` was set to `OFF`.

Action: Call `SetSystemParameter` to set `ALLOW_CAPTURE_EVENTS` to `ON`, and retry the operation.

**WM\_ERROR\_274 this parameter cannot be set to 'OFF' when some events are set to be captured**

Cause: An attempt was made to disallow the capture of Workspace Manager events while one or more types of events were set to be captured.

Action: Turn off event capture by calling `SetCaptureEvents('ALL_EVENTS', 'OFF')`, and retry the operation.

**WM\_ERROR\_275 invalid value for capture - 'ON' or 'OFF' expected**

Cause: The `SetCaptureEvent` procedure was called with an invalid value for the capture parameter.

Action: Specify either `ON` or `OFF` for the capture parameter.

**WM\_ERROR\_276 this parameter cannot be set to 'OFF' when some multiparent workspaces exist**

Cause: An attempt was made to set `ALLOW_MULTI_PARENT_WORKSPACES` to `OFF` when one or more multiparent workspaces existed in the system.

Action: Remove all multiparent workspaces by using any combination of the `RemoveAsParentWorkspace`, `MergeWorkspace`, and `RemoveWorkspace` procedures.

**WM\_ERROR\_277 system parameter 'ALLOW\_NESTED\_TABLE\_COLUMNS' cannot be set to 'OFF' when a version enabled table exists containing a nested table column**

Cause: An attempt was made to set `ALLOW_NESTED_TABLE_COLUMNS` to `OFF` when one or more version-enabled tables contained a nested table column.

Action: Disable versioning on all tables that contain a nested table column.

**WM\_ERROR\_278 'string' cannot be version enabled because system parameter 'ALLOW\_NESTED\_TABLE\_COLUMNS' has been set to 'OFF'**

Cause: An attempt was made to version-enable a table containing a nested table column and the Workspace Manager system parameter `ALLOW_NESTED_TABLE_COLUMNS` was set to `OFF`.

Action: Call `DBMS_WM.SetSystemParameter` to set `ALLOW_NESTED_TABLE_COLUMNS` to `ON`, and retry the operation.

**WM\_ERROR\_279 histogram stats not found for table 'string.string' on column 'string'**

Cause: Required histogram statistics have not been collected on the specified column.

Action: Use the `DBMS_STATS.GATHER_TABLE_STATS` procedure to collect the histogram statistics; then try the operation again.

**WM\_ERROR\_280 datatype of column 'string' in table 'string.string' not supported for batch updates**

Cause: The specified data type cannot be used for batches of `PRIMARY_KEY_RANGE`.

Action: Specify the batch size as `TABLE`.

**WM\_ERROR\_281 batch\_size parameter must be 'TABLE' or 'PRIMARY\_KEY\_RANGE'**

Cause: The `batch_size` parameter value was invalid.

Action: Specify the `batch_size` parameter value as `TABLE` or `PRIMARY_KEY_RANGE`.

**WM\_ERROR\_282 number of batches must be between 1 and 1000**

Cause: The value specified for the Workspace Manager system parameter `NUMBER_OF_COMPRESS_BATCHES` was invalid.

Action: Specify a number from 1 to 1000 (inclusive).

**WM\_ERROR\_283 invalid 'string' parameter specified**

Cause: The value specified for the parameter was invalid.

Action: Specify a valid value for that parameter.

**WM\_ERROR\_284 the length of the 'string' parameter cannot exceed 'string' characters**

Cause: The value of a specified parameter exceeded the allowable length.

Action: Shorten the value to an allowable length.

**WM\_ERROR\_285 the 'string' parameter must be set to 'string'**

Cause: The value specified for the parameter was invalid.

Action: Set the parameter to the specified value.

**WM\_ERROR\_286 quoted identifiers are not supported**

Cause: A quoted identifier was used.

Action: Remove the quote from the table or dependent object and retry the operation.

**WM\_ERROR\_287 the 'string' parameter must be specified**

Cause: The specified parameter was omitted.

Action: Include the parameter when executing the operation.

**WM\_ERROR\_288 'string' privilege(s) needed**

Cause: The executing user did not have the specified privileges.

Action: Grant the necessary privilege or privileges to the user.

**WM\_ERROR\_289 insufficient privileges on 'string'**

Cause: The executing user did not have the necessary privileges on the object.

Action: Grant the necessary privilege or privileges on the object to the user.

**WM\_ERROR\_290 only the table owner or a user with the "WM\_ADMIN" system privilege can invoke this procedure**

Cause: The procedure invoker had insufficient privileges.

Action: Either grant the `WM_ADMIN` system privilege or `WM_ADMIN_ROLE` role to the executing user, or execute the procedure as the owner of the table.

**WM\_ERROR\_291 deadlock detected while trying to acquire lock on 'string'**

Cause: A deadlock was encountered when acquiring a lock using the `DBMS_LOCK` package.

Action: Wait for the sessions that caused the deadlock to either roll back or commit their transactions.

**WM\_ERROR\_292 error while trying to acquire lock on 'string', status='string'**

Cause: A required lock could not be obtained when using the `DBMS_LOCK` package.

Action: The required action is dependent on the specified status: status=1: retry the operation; status=2: wait for the sessions that caused the deadlock to either roll back or commit their transactions; status=3 or status=5: contact Oracle Support Services to resolve the issue.

**WM\_ERROR\_293 error while trying to release lock on '*string*', status='*string*'**

Cause: The required lock could not be released when using the DBMS\_LOCK package.

Action: Contact Oracle Support Services to resolve the issue.

**WM\_ERROR\_294 open short transaction exists on '*string*'**

Cause: An open transaction had a lock on the specified object.

Action: Commit or roll back any open transactions on the object.

**WM\_ERROR\_295 the '*string*' schema cannot contain dependent objects**

Cause: The specified schema contained a dependent object when trying to version-enable a table.

Action: Drop the objects from the schema, and re-create them in a different schema.

**WM\_ERROR\_296 topology '*string*' does not exist**

Cause: The specified topology did not exist.

Action: Specify the correct topology.

**WM\_ERROR\_297 object '*string*' already exists**

Cause: A required object already exists.

Action: Drop the object.

**WM\_ERROR\_298 object '*string*' does not exist**

Cause: The specified object did not exist.

Action: Create the necessary object if it does not exist; otherwise, check the spelling of the string.

**WM\_ERROR\_299 tablespace '*string*' does not exist**

Cause: The specified tablespace did not exist.

Action: Execute the procedure with a valid tablespace name.

**WM\_ERROR\_300 the '*string*' workspace cannot be renamed**

Cause: The specified workspace cannot be renamed.

Action: Do not attempt to rename the workspace.

**WM\_ERROR\_301 workspace names cannot include the character '*string*'**

Cause: A workspace name contained the invalid specified character.

Action: Choose a different workspace name that does not contain the character.

**WM\_ERROR\_302 the new name must be distinct from the old name**

Cause: When renaming an object, the old name and the new name must be distinct.

Action: Choose a different new name for the object.

**WM\_ERROR\_303 a workspace cannot be added as a parent workspace of itself**

Cause: An attempt was made to add a workspace as a parent workspace of itself.

Action: Do not try to add a workspace as a parent of itself.

**WM\_ERROR\_304 cannot add a parent workspace to LIVE**

Cause: An attempt was made to add a workspace as a parent workspace of the `LIVE` workspace.

Action: Do not attempt to add a parent workspace to the `LIVE` workspace.

**WM\_ERROR\_305 'string' cannot be called on a topology feature table**

Cause: An invalid operation was attempted on a topology feature table.

Action: Do not execute the operation on the table.

**WM\_ERROR\_306 MergeTable with remove\_data on a parent table of a referential constraint is not supported for multi-parent workspaces**

Cause: The MergeTable operation was executed on an unsupported table.

Action: Set the `remove_data` parameter to `false`, or do not execute the operation on the table for the specified workspace.

**WM\_ERROR\_307 workspace 'string' has a crstatus that is not supported for this operation**

Cause: The operation attempted to use a workspace that is not currently supported.

Action: Do not execute the operation for the specified workspace.

**WM\_ERROR\_308 a 'string' with more than 'string' characters in its name was found - rename to a shorter name**

Cause: The object's name exceeded the specified number of characters.

Action: Rename the object to have a shorter name.

**WM\_ERROR\_309 some tables have already been version enabled with a different validtime option**

Cause: An attempt was made to modify an topology feature table in an optimistic mode.

Action: Use the `validtime` parameter option that was originally specified.

**WM\_ERROR\_310 a topology feature table can be modified only in pessimistic mode - set locking ON**

Cause: An attempt was made to modify an topology feature table in an optimistic mode.

Action: Use a pessimistic setting when modifying this table.

**WM\_ERROR\_311 feature tables cannot be deleted from this topology**

Cause: An attempt was made to remove the last feature table from the topology.

Action: Do not remove the last feature table from a topology.

**WM\_ERROR\_312 topology 'string' does not have a feature table**

Cause: EnableVersioning was being executed on a topology without a feature table.

Action: Add a feature table to the topology.

**WM\_ERROR\_313 the 'string' parameter is incompatible with the other options**

Cause: The specified parameter was not compatible with the other parameters.

Action: Specify a valid value for the parameter. In the case of the `PurgeTable` procedure, do not specify non-null values for both the `instant` and `savepoint` parameters.

**WM\_ERROR\_314 'string' is not a version-enabled topology**

Cause: The topology specified has not been version-enabled.

Action: Only specify topologies that have been version-enabled.

**WM\_ERROR\_315 this operation is not allowed during bulk loading**

Cause: An unsupported operation was executed during bulk loading.

Action: Wait for the bulk loading of the workspace to complete before executing the operation.

**WM\_ERROR\_316 wm\_version column should not be specified when bulk loading**

Cause: The `wm_version` column was specified in the control file.

Action: Remove the `wm_version` column.

**WM\_ERROR\_317 cannot bulk load into a table with a self-referential integrity constraint**

Cause: An attempt was made to bulk load into a table that contained a foreign key constraint that referenced a column in the same table.

Action: Either disable or remove the constraint from the table.

**WM\_ERROR\_318 bulk loading into only the latest version of a workspace is supported**

Cause: An attempt was made to bulk load into a version that is not supported.

Action: When bulk loading into the `LIVE` workspace, the `savepoint_name` parameter must be either `'LATEST'` or `'ROOT_VERSION'`. When bulk loading into any workspace other than `LIVE`, the `savepoint_name` parameter must be `'LATEST'`.

**WM\_ERROR\_319 this operation is not supported while 'string' is in 'string' state**

Cause: An operation was executed while the specified table was in an unsupported state.

Action: Revert the table to a supported state before retrying the operation.

**WM\_ERROR\_320 Workspace Manager must first be uninstalled from CDB\$ROOT**

Cause: An attempt was made to uninstall Workspace Manager from a PDB (pluggable database) before uninstalling it from the `CDB$ROOT`.

Action: Uninstall Workspace Manager from `CDB$ROOT` before the PDB.

**WM\_ERROR\_321 nested tables cannot themselves contain a nested table**

Cause: An attempt was made to version-enable a table that contained a nested table that also contained a nested table.

Action: Remove one of the nested table columns.

**WM\_ERROR\_322 nested tables contained in objects are not supported**

Cause: An attempt was made to version-enable a table that contained a column whose data type contained a nested table.

Action: Remove the nested table from the data type, or the column from the table.

**WM\_ERROR\_323 system parameter 'string' must be set to 'string'**

Cause: An operation was executed that required a system parameter to match a specific value.

Action: Modify the system parameter to the appropriate value.

**WM\_ERROR\_324 the following index tables were unable to be DisableVersioned: 'string'**

Cause: During a downgrade operation, there were errors when attempting to disable version the specified index table.

Action: Contact Oracle Support Services to resolve the issue.

**WM\_ERROR\_325 the following index tables were unable to be EnableVersioned: 'string'**

Cause: During an upgrade operation, there were errors when attempting to version-enable the specified index table.

Action: Contact Oracle Support Services to resolve the issue.

**WM\_ERROR\_326 null value not allowed for a primary key column**

Cause: A DML operation resulted in the primary key value being set to null.

Action: Do not use null values for primary key values.

**WM\_ERROR\_327 the child table 'string' contains a record outside the specified valid time range**

Cause: A constraint violation occurred as a result of adding the `validtime` option to a parent table in a foreign key relationship.

Action: Set the `validtime` range to encompass all child table rows.

**WM\_ERROR\_328 all child tables must be valid-time enabled**

Cause: An attempt was made to add the `validtime` option to a table that is the parent table in a foreign key constraint, without first adding the option to all child tables.

Action: Add the `validtime` option to the necessary child tables.

**WM\_ERROR\_329 this index operation is not currently supported while 'string' is version enabled**

Cause: An attempt was made to perform an unsupported DDL operation on an index due to the table being version-enabled

Action: Avoid issuing the DDL statement directly on the index. Use `CommitDDL`.

**WM\_ERROR\_330 multi-transaction update not supported for a clustered table - specify 'UNLIMITED' undo space**

Cause: An invalid `undo_space` parameter value was specified for a clustered table.

Action: Set the `undo_space` parameter to 'UNLIMITED'.

**WM\_ERROR\_331 object tables cannot be version enabled**

Cause: An attempt was made to version-enable an object table.

Action: Do not execute `EnableVersioning` on an object table.

**WM\_ERROR\_332 One of the rows in the lineage to be removed has a createtime greater than the specified instant**

Cause: The PurgeTable procedure was unable to complete due to existing data.

Action: Specify an instant that is greater than all rows that are going to be purged.

**WM\_ERROR\_333 referential constraints on object attributes not supported**

Cause: An attempt was made to version-enable a table that contained a referential integrity constraint on an object attribute.

Action: Drop the constraint before version-enabling the table.

**WM\_ERROR\_334 savepoint names cannot include the character 'string'**

Cause: A savepoint name contained the invalid specified character.

Action: Choose a different savepoint name that does not contain the character.

**WM\_ERROR\_335 this operation is not currently supported while 'string' is version enabled**

Cause: An unsupported DDL statement was executed while the table is version-enabled.

Action: Avoid issuing the DDL statement directly on the object. Use CommitDDL.

**WM\_ERROR\_336 the index 'string' cannot be renamed**

Cause: An attempt was made to rename an index that cannot be renamed.

Action: Do not rename the specified index.

**WM\_ERROR\_337 this table can only be dropped by dbms\_wm.rollbackDDL/commitDDL**

Cause: An attempt was made to drop the skeleton table created by the BeginDDL procedure.

Action: Execute RollbackDDL or CommitDDL.

**WM\_ERROR\_338 skeleton tables used by beginDDL/commitDDL cannot be version enabled**

Cause: An attempt was made to version-enable the skeleton table created by the BeginDDL procedure.

Action: Do not execute the EnableVersioning procedure on this table.

**WM\_ERROR\_339 this table is not available for queries and DML operations**

Cause: An attempt was made to issue a DML statement on a table that is currently unavailable.

Action: Wait for the operation that caused the table to become unavailable to complete.

**WM\_ERROR\_340 this table is used by Workspace Manager for referential integrity checking**

Cause: An attempt was made to drop a table required by Workspace Manager for referential integrity checking.

Action: Do not attempt to drop the table.

**WM\_ERROR\_341 check constraint 'string' violated**

Cause: When executing CommitDDL, a check constraint was violated.



Action: Disable the constraint or remove the data that is violating the constraint.

**WM\_ERROR\_342 trigger creation on this object is not supported**

Cause: An attempt was made to create a trigger directly on the base table used by Workspace Manager.

Action: Avoid creating the trigger directly on the base table. Use CommitDDL.

**WM\_ERROR\_343 types contained in versioned tables cannot be dropped**

Cause: An attempt was made to drop a type required by a version-enabled table.

Action: Version-disable the table before dropping the type.

**WM\_ERROR\_344 unable to create \_LCK table**

Cause: An index necessary for the creation on the \_LCK table was unable to be created due to an existing object.

Action: Rename the index preventing the creation of the \_LCK table.

**WM\_ERROR\_345 unable to support unique index due to naming conflict**

Cause: An attempt was made to version-enable a table containing an index name that has a naming conflict.

Action: Rename the necessary index.

**WM\_ERROR\_346 insufficient privileges to revoke 'string'**

Cause: An attempt was made to invoke the RevokeWorkspacePriv or RevokeSystemPriv procedure without the required privileges to do so.

Action: Ensure that the invoking user has the required privileges to revoke the privilege. A user needs to have been granted a privilege with the GRANT option to be able to revoke it from others.

**WM\_ERROR\_347 the object definition exceeds the maximum allowed length**

Cause: The search\_condition for a check constraints exceeded the supported length.

Action: Redefine the constraint to use a shorter constraint definition.

**WM\_ERROR\_348 constraint 'string' exists on another table**

Cause: A foreign key constraint was unable to be created due to the existence of another constraint with the same name.

Action: Choose a new name for the constraint.

**WM\_ERROR\_349 constraint enforcement on existing versioned data not supported in the presence of multi-parent workspaces**

Cause: Enforcement of a foreign key constraint was not possible due to the presence of a multiparent workspace.

Action: Drop or disable the constraint, or set the `enforce_ric` parameter to `false`.

**WM\_ERROR\_350 hint cannot contain 'string'**

Cause: An unsupported hint was specified when executing the AdduserDefinedHint procedure.

Action: Do not include the specified string within the hint.

**WM\_ERROR\_351 implicit savepoints cannot be renamed**

Cause: An attempt was made to rename an implicit savepoint.

Action: Do not rename an implicit savepoint.

**WM\_ERROR\_352 'string' not granted to 'string'**

Cause: An attempt was made to revoke privileges from a user that did not have them.

Action: Only revoke privileges from a user that has already had the privileges granted to them.

**WM\_ERROR\_353 instant can be specified only for a table with history option**

Cause: The `instant` parameter was not null for a table that did not have the `history` option.

Action: Set the `instant` parameter to be null.

**WM\_ERROR\_354 only LATEST data is currently supported**

Cause: An attempt was made to import or export non-LATEST data.

Action: Ensure the compatibility of the parameters specified for the operation.

**WM\_ERROR\_355 overlapping periods found for column `wm_valid` of table 'string'**

Cause: When version-enabling a table with `validtime`, a constraint violation was found within the table.

Action: Make sure that the primary key is unique for all time periods.

**WM\_ERROR\_356 referential constraint 'string' exists with another parent table**

Cause: A foreign key constraint was unable to be created due to the existence of another constraint with the same name.

Action: Choose a new name for the constraint.

**WM\_ERROR\_357 referential constraint with a non-primary-key constraint 'string' detected**

Cause: A referential integrity constraint was found within the table that referenced a column that was not part of the primary key.

Action: Disable the constraints or only include primary key columns.

**WM\_ERROR\_359 incompatible options specified**

Cause: Options were specified that are incompatible with each other.

Action: Remove one of the incompatible options.

**WM\_ERROR\_360 this release only supports changing to the workspace type: `DBMS_WM.CR_WORKSPACE_TYPE`**

Cause: An attempt was made to change a workspace into a non-continually refreshed workspace.

Action: Do not attempt to modify continually refreshed workspaces with the `ChangeWorkspaceType` procedure.

**WM\_ERROR\_361 unable to import this table**

Cause: An attempt was made to import a table that required unsupported operations.

Action: Do not specify either the `ancestor_savepoint_workspace` or `ancestor_savepoint_name` parameters when importing this table.

**WM\_ERROR\_362 unsupported options specified while multi-parent workspaces exist**

Cause: The parameters specified for the `PurgeTable` procedure are not supported when a multiparent workspace exists.

Action: When a multiparent workspace exists, the following parameters must be set when executing `Purgetable`: `workspace=>'LIVE'`, `PurgeAfter=>>true`, `savepoint_name=>null`, `instant=>null`.

**WM\_ERROR\_363 unable to create workspace with the same name as a workspace whose removal has been deferred**

Cause: A workspace was removed using by specifying either 'FAST' or 'REMOVE\_LOCKS' for the `defer_option` parameter. Until the workspace has been completely cleaned up, a new workspace with the same name cannot be created.

Action: Execute `dbms_wm.RemoveDeferredWorkspaces` before creating the workspace.

**WM\_ERROR\_364 'string' not supported on a table with valid time**

Cause: The operation was not supported for table with the `validtime` option.

Action: Do not execute the operation on a table that has the `validtime` option enabled.

**WM\_ERROR\_365 'string' and 'string' have an identical column name**

Cause: A nested table had a column with the same name as a column that is contained in its parent table.

Action: Rename either the column in the nested table or the column in the parent table.

**WM\_ERROR\_366 'string' is already version enabled with valid time option**

Cause: An attempt was made to add the `validtime` option to a table that already has the option enabled.

Action: Do not use the `ADD_VALID_TIME` option of `alterVersionedTable` for a version-enabled table that already has the `validtime` option enabled.

**WM\_ERROR\_367 invalid active date for DML**

Cause: An invalid time was specified when modifying a table.

Action: Contact Oracle Support Services to resolve the issue.

**WM\_ERROR\_368 a referential integrity constraint will be violated between 'string' and 'string' in the workspace 'string' or one of its descendants - child record found**

Cause: The operation could not be completed due to a constraint violation of a foreign key constraint in the specified workspace (or one of its descendants).

Action: Fix the constraint violation before executing the operation.

**WM\_ERROR\_369 a referential integrity constraint will be violated between 'string' and 'string' in the workspace 'string' or one of its descendants - parent key not found**

Cause: The operation could not be completed due to a constraint violation of a foreign key constraint in the specified workspace (or one of its descendants).

Action: Fix the constraint violation before executing the operation.

**WM\_ERROR\_370 invalid values contained in the wm\_valid column**

Cause: When executing EnableVersioning, a row with an invalid `wm_valid` column was found.

Action: Fix all rows containing the `wm_valid` column so that the `validfrom` column is non-null and the `validtill` column is not less than the `validfrom` column.

**WM\_ERROR\_371 privileges on 'string' policy must be granted using `dbms_wm.grantPrivsOnPolicy`**

Cause: A table with an OLS policy was being version-enabled before executing the `grantPrivsOnPolicy` procedure for the table.

Action: Execute the `grantPrivsOnPolicy` procedure on the table before executing `EnableVersioning`.

**WM\_ERROR\_372 Workspace Manager is not properly installed**

Cause: An import operation was being performed with an invalid installation of Workspace Manager.

Action: Make sure that Workspace Manager is in a valid state before importing the data.

**WM\_ERROR\_373 unsupported version of Workspace manager detected**

Cause: An import operation was being performed with an incompatible version of Workspace Manager installed.

Action: Install a compatible version of Workspace Manager before importing the data.

**WM\_ERROR\_374 Datapump job did not successfully complete**

Cause: An error was encountered during the Oracle Data Pump job.

Action: See the Data Pump log files for more details on the error.

**WM\_ERROR\_375 invalid datapump job specified**

Cause: The specified Oracle Data Pump job could not be used for the operation due to incompatible options.

Action: Check the options specified when executing the `dbms_datapump.open` procedure.

**WM\_ERROR\_376 the 'string' schema does not contain the necessary data**

Cause: The schema contained in the dump file did not contain the necessary data to complete the `import_schemas` procedure.

Action: Only use dump files that have been generated by the `export_schemas` procedure.

**WM\_ERROR\_377 Export and Import operations cannot be run simultaneously**

Cause: An attempt was made to run simultaneous export and import operations.

Action: Only execute the `export_schemas` and `import_schemas` procedures so that they do not overlap.

**WM\_ERROR\_378 Removing a workspace that was a parent of a workspace whose removal has been referred is not supported**

Cause: An attempt was made to remove a workspace that has a child workspace that had its removal deferred.

Action: Execute the `dbms_wm.RemoveDeferredWorkspaces` procedure to finish the removal of the child workspace before removing its parent, or defer the removal of the parent workspace as well.

**WM\_ERROR\_379 a workspace with a 'string' lockmode cannot be merged or refreshed**

Cause: An attempt was made to merge or refresh a workspace with the specified lockmode.

Action: Do not attempt to merge or refresh this workspace with that lockmode.

**WM\_ERROR\_380 the 'string' lockmode cannot be set for a continually refreshed workspace or the 'LIVE' workspace**

Cause: An attempt was made to set the lockmode of a continually refreshed workspace or the 'LIVE' workspace to an unsupported mode.

Action: Do not set the lockmode of this workspace to an unsupported mode. Instead, specify one that is supported ('E', 'S', 'VE', 'WE').

**WM\_ERROR\_381 all child workspaces must have their lockmodes set to 'D'**

Cause: An attempt was made to set the lockmode of the specified workspace to 'D', even though a child workspace of the specified workspace had a different lockmode.

Action: For any workspace with the lockmode set to 'D', all child workspaces must also have the same lockmode. So, either update the lockmode of all of the child workspaces of the specified workspace to 'D' before changing the lockmode of the specified workspace, or use a different lockmode for this workspace.

**WM\_ERROR\_382 range partitioning is not supported for version enabled tables**

Cause: An attempt was made to version-enable a table that uses range partitioning.

Action: Either do not attempt to version-enable this table, or use a different partitioning scheme.

**WM\_ERROR\_383 Workspace Manager must first be uninstalled from each PDB prior to uninstalling from CDB\$ROOT**

Cause: An attempt was made to uninstall Workspace Manager from CDB\$ROOT while it still exists in one or more PDBs.

Action: If Workspace Manager needs to be uninstalled, be sure to uninstall it in each PDB before attempting to uninstall it from CDB\$ROOT.

**WM\_ERROR\_384 This operation cannot be executed while the DEFAULT\_WORKSPACE system parameter is set to 'string'**

Cause: An attempt was made to modify the workspace defined by the DEFAULT\_WORKSPACE system parameter in an unsupported manner.

Action: Either change the DEFAULT\_WORKSPACE system parameter to a different workspace, or avoid executing the procedure on this workspace.

**WM\_ERROR\_385 The workspace '*string*' must have the ACCESS\_WORKSPACE privilege granted to PUBLIC**

Cause: An attempt was made to change the DEFAULT\_WORKSPACE system parameter to a workspace that lacked access by PUBLIC..

Action: Grant the ACCESS\_WORKSPACE privilege to PUBLIC for the specified workspace, or choose a different workspace to be the default.

**WM\_ERROR\_386 Invisible columns are not supported**

Cause: An attempt was made to version-enable a table that contained an invisible column.

Action: Either drop the invisible column from the table, or do not attempt to version-enable the table.

**WM\_ERROR\_387 This procedure has been deprecated**

Cause: This procedure is no longer supported in this version of Workspace Manager.

Action: Do not use this procedure.

**WM\_ERROR\_388 Column value has been improperly modified within a user-defined trigger**

Cause: Either a primary key column value was being updated by a user-defined trigger in an unsupported manner, or a :NEW value was being modified in a delete trigger.

Action: Avoid doing such modifications, either by updating the trigger definition or by executing `dbms_wm.SetTriggerEvents` to specify when the trigger is executed.

**WM\_ERROR\_389 Only identity columns defined as "always" or "by default on null" are supported**

Cause: An attempt was made to version-enable a table that contained an identity column that was not one of the listed types. (For example, it might have been a "generated always" identity column, which is not supported for version-enabling.)

Action: Either change the type of identity column, or do not attempt to version-enable the table.

**WM\_ERROR\_390 Redaction policies are not currently supported for version-enabled tables**

Cause: An attempt was made to version-enable a table that has a redaction policy defined on it.

Action: Remove the redaction policy from the table, or do not attempt to version-enable the table.

**WM\_ERROR\_391 The isTopology parameter is required for the '*string*' table**

Cause: Either `enableversioning` or `disableversioning` was executed on a table that is part of a topology.

Action: Set the `table_name` parameter to the name of the topology and the `isTopology` parameter to `true` in order to include this table.

**WM\_ERROR\_392 A virtual column that is part of the primary key cannot reference non-primary key columns**

Cause: `EnableVersioning` failed due to a virtual column that is included in the primary key and references unsupported columns.

---

Action:Either redefine the virtual column or remove the column as part of the primary key prior to executing EnableVersioning.

**WM\_ERROR\_393 A virtual column that is part of the primary key cannot reference non-primary key columns**

Cause: The LIVE workspace cannot be the source or target workspace Cause: When executing CopyWorkspace, either the source workspace or the target workspace parameters was set to the 'LIVE' workspace .

Action:Choose a different workspace.

**WM\_ERROR\_394 Virtual column expression cannot be modified because a constraint is defined on the column**

Cause: A DDL operation attempted to modify the virtual column expression of a column that has a referential integrity constraint defined on it.

Action:Remove the constraint or do not modify the virtual column expression.

**WM\_ERROR\_395 Deferrable constraints are not supported**

Cause:EnableVersioning or a DDL operation was executed on a table containing a deferrable constraint.

Action:Redefine the constraint so that it is no longer deferrable.

**WM\_ERROR\_396 Deferrable constraints are not supported**

Cause:When executing CopyWorkspace, both the source workspace and the target workspace parameters were specified to be the same workspace.

Action:Specify distinct workspaces for the source and target of a CopyWorkspace operation.

# Glossary

## **active version**

See [current version](#).

## **child workspace**

A workspace created from its parent workspace.

See *also* [parent workspace](#) and [workspace hierarchy](#).

## **conflicts**

Differences in data values resulting from changes to rows in the child and parent workspace. Conflicts are detected at merge time and presented to the user in conflict views.

See *also* [merging \(a workspace\)](#).

## **context**

Information about the workspace that determines what data the session can see in the workspace. The context can be retrieved using the [GetSessionInfo](#) procedure

## **current version**

The version in which the changes are currently being made.

## **exclusive locking**

A Workspace Manager lock mode that prevents any other user from changing a locked row.

See *also* [locks](#).

## **explicit savepoint**

A savepoint that is explicitly created. It can later be used to perform partial rollbacks in workspaces.

See *also* [savepoint](#), [implicit savepoint](#), and [removable savepoint](#).



**freezing (a workspace)**

Causing the condition in which no changes can be made to data in version-enabled rows in a workspace, and access to the workspace is restricted.

**implicit savepoint**

A savepoint that is created automatically whenever a new workspace is created.

See also [savepoint](#), [explicit savepoint](#), and [removable savepoint](#).

**LATEST**

The name of the logical savepoint that refers to the latest version in the workspace.

See also [savepoint](#).

**LIVE**

The name of the topmost workspace in the workspace hierarchy.

See also [workspace hierarchy](#).

**locks**

Version locks provided by Workspace Manager, separate from locks provided by conventional Oracle database transactions. These locks are primarily intended to eliminate row conflicts between a parent workspace and a child workspace. Locking is enabled at a session level and is a session property independent of the workspace in which the session is. When locking is enabled for a session, it locks rows in all workspaces in which it participates.

**merging (a workspace)**

Applying changes made in a workspace to its parent workspace.

**nonwriter site**

A master site in a multimaster group in a Workspace Manager replication environment that is not the writer site. A nonwriter site cannot perform any write operations, but can perform all read operations, such as [CreateSavepoint](#) or SELECT queries on version-enabled tables.

See also [writer site](#).

**parent workspace**

A workspace from which another workspace (a child workspace) was created.

See also [child workspace](#) and [workspace hierarchy](#).

**privileges**

A set of privileges for Workspace Manager that are separate from standard Oracle database privileges. Workspace-level privileges (with names in the form `xxx_WORKSPACE`) that allow the user to affect a specified workspace. System-level privileges (with names in the form `xxx_ANY_WORKSPACE`) that allow the user to affect any workspace.

**removable savepoint**

A workspace that can be deleted by the [CompressWorkspace](#), [CompressWorkspaceTree](#), and [DeleteSavepoint](#) procedures. A savepoint is removable if it is an explicit savepoint or if it is an implicit savepoint that does not have any child dependencies.

See also [savepoint](#), [explicit savepoint](#), and [implicit savepoint](#).

**rolling back (a workspace)**

Deleting either all changes made in the workspace or all changes made after a savepoint (that is, an explicit savepoint).

**savepoint**

A point in the workspace to which operations can be rolled back. It is analogous to a firewall, in that by creating a savepoint you can prevent any damage to the "other side" of the wall (that is, operations performed in the workspace before the savepoint was created).

See also [explicit savepoint](#), [implicit savepoint](#), and [removable savepoint](#).

**session context**

See [context](#).

**shared locking**

A Workspace Manager lock mode that allows only users in the workspace in which the row was locked to modify the row.

See also [locks](#).

**unfreezing (a workspace)**

Reversing the effect of a freeze operation.

See also [freezing \(a workspace\)](#).

**valid time**

The time during which a record is valid, or the ability to specify the time for which a record is valid. Also called *effective dating*.

**valid time filter**

A time that is applied to queries against version-enabled tables that have valid time support. When a valid time filter is set for the current session, only rows that are valid for the specified time are returned.

**version-enabled table**

A table in the database in which all rows in the table can now support multiple versions of data. The versioning infrastructure is not visible to the database users. After a table has been version-enabled, users automatically see the correct version of the record in which they are interested.

**workspace**

A virtual environment that one or more users can share to make changes to the data in the database. Workspace management involves managing one or more workspaces that can be shared by many users.

**workspace hierarchy**

The hierarchy of workspaces in the database. For example, a workspace can be a parent to one or more workspaces. By default, when a workspace is created, it is created from the topmost, or `LIVE`, database workspace.

**workspace management**

The ability of the database to hold different versions of the same record (that is, row) in one or more workspaces.

**writer site**

The master definition site in a Workspace Manager replication environment. Only the writer site can perform workspace operations and DML and DDL operations on version-enabled tables. All other sites in the multimaster group are nonwriter sites.

See also [nonwriter site](#).

# Index

## Symbols

---

### `_LT` tables

- created for Workspace Manager infrastructure, [1-12](#)
- getting name of, [4-60](#)

## A

---

Add\_Topo\_Geometry\_Layer procedure, [4-4](#)

AddAsParentWorkspace procedure, [4-5](#)

AddUserDefinedHint procedure, [4-7](#)

Advanced Queuing

- and Workspace Manager events, [2-1](#)

ALL\_MP\_GRAPH\_WORKSPACES view, [5-3](#)

ALL\_MP\_PARENT\_WORKSPACES view, [5-4](#)

ALL\_REMOVED\_WORKSPACES view, [5-4](#)

ALL\_VERSION\_HVIEW view, [5-5](#)

ALL\_WM\_CONS\_COLUMNS view, [5-6](#)

ALL\_WM\_CONSTRAINT\_VIOLATIONS view, [5-6](#)

ALL\_WM\_CONSTRAINTS view, [5-7](#)

ALL\_WM\_IND\_COLUMNS view, [5-7](#)

ALL\_WM\_IND\_EXPRESSIONS view, [5-8](#)

ALL\_WM\_LOCKED\_TABLES view, [5-8](#)

ALL\_WM\_MODIFIED\_TABLES view, [5-9](#)

ALL\_WM\_POLICIES view, [5-9](#)

ALL\_WM\_RIC\_INFO view, [5-9](#)

ALL\_WM\_TAB\_TRIGGERS view, [5-10](#)

ALL\_WM\_VERSIONED\_TABLES view, [5-12](#)

ALL\_WM\_VT\_ERRORS view, [5-13](#)

ALL\_WORKSPACE\_PRIVS view, [5-14](#)

ALL\_WORKSPACE\_SAVEPOINTS view, [5-14](#)

ALL\_WORKSPACES view, [5-15](#)

ALLOW\_CAPTURE\_EVENTS system parameter, [2-3](#)

altering

- savepoint description, [4-8](#)
- version-enabled table to add valid time support, [4-9](#)
- workspace description, [4-12](#)

AlterSavepoint procedure, [4-8](#)

AlterVersionedTable procedure, [4-9](#)

AlterWorkspace procedure, [4-12](#)

asynchronous notification for Workspace Manager events, [2-6](#)

auditing modifications

- EnableVersioning history option, [4-44](#)

- history views (xxx\_HIST), [5-27](#)

auto\_commit parameter, [1-9](#)

autocommitting of operations, [1-9](#)

## B

---

base row, [1-8](#)

BeginBulkLoading procedure, [4-13](#)

BeginDDL procedure, [4-15](#)

BeginResolve procedure, [4-16](#)

bulk loading, [1-33](#)

- BeginBulkLoading procedure, [4-13](#)

- CommitBulkLoading procedure, [4-18](#)

- RollbackBulkLoading procedure, [4-109](#)

## C

---

ChangeWorkspaceType procedure, [4-17](#)

child row, [1-8](#)

child workspace, [1-4](#)

- as alternative to creating savepoint, [1-6](#)
- merging, [4-86](#)
- refreshing, [4-94](#), [4-95](#)
- removing, [4-100](#)

CommitBulkLoading procedure, [4-18](#)

CommitDDL procedure, [4-21](#)

CommitResolve procedure, [4-22](#)

compressing

- workspaces, [4-23](#), [4-27](#)

compression (Workspace Manager)

- SetCompressWorkspace procedure, [4-117](#)
- WM\_COMPRESS\_BATCH\_SIZES view, [5-23](#)

- WM\_COMPRESSIBLE\_TABLES view, [5-24](#)

CompressWorkspace procedure, [4-23](#)

CompressWorkspaceTree procedure, [4-27](#)

conflict management, [1-47](#), [4-103](#)

- beginning resolution, [4-16](#)
- committing resolution, [4-22](#)
- rolling back resolution, [4-111](#)
- showing conflicts, [4-118](#)

conflict resolution

- example, [5-25](#)

conflict views (xxx\_CONF), [5-25](#)  
 constraints  
   maximum name length for Workspace Manager, [1-12](#)  
   support with Workspace Manager, [1-36](#)  
 context (session), [1-15](#)  
   GetSessionInfo function, [4-61](#)  
 context of current operation  
   getting, [4-58](#)  
 continually refreshed workspaces, [1-10](#)  
   changing workspace type, [4-17](#)  
   creating, [4-35](#)  
 CopyForUpdate procedure, [4-31](#)  
 copying  
   workspace, [4-32](#)  
 CopyWorkspace procedure, [4-32](#)  
 CreateSavepoint procedure, [4-33](#)  
 CreateWorkspace procedure, [4-34](#)  
 creating  
   savepoints, [4-33](#)  
   workspaces, [4-34](#)

## D

DBA\_REMOVED\_WORKSPACES view, [5-17](#)  
 DBA\_WM\_SYS\_PRIVS view, [5-17](#)  
 DBA\_WM\_VERSIONED\_TABLES view, [5-17](#)  
 DBA\_WM\_VT\_ERRORS view, [5-17](#)  
 DBA\_WORKSPACE\_PRIVS view, [5-18](#)  
 DBA\_WORKSPACE\_SAVEPOINTS view, [5-18](#)  
 DBA\_WORKSPACE\_SESSIONS view, [5-18](#)  
 DBA\_WORKSPACES view, [5-18](#)  
 DBMS\_WM package, [1-15](#)  
   Add\_Topo\_Geometry\_Layer, [4-4](#)  
   AddAsParentWorkspace, [4-5](#)  
   AddUserDefinedHint, [4-7](#)  
   AlterSavepoint, [4-8](#)  
   AlterVersionedTable, [4-9](#)  
   AlterWorkspace, [4-12](#)  
   BeginBulkLoading, [4-13](#)  
   BeginDDL, [4-15](#)  
   BeginResolve, [4-16](#)  
   ChangeWorkspaceType, [4-17](#)  
   CommitBulkLoading, [4-18](#)  
   CommitDDL, [4-21](#)  
   CommitResolve, [4-22](#)  
   CompressWorkspace, [4-23](#)  
   CompressWorkspaceTree, [4-27](#)  
   CopyForUpdate, [4-31](#)  
   CopyWorkspace, [4-32](#)  
   CreateSavepoint, [4-33](#)  
   CreateWorkspace, [4-34](#)  
   Delete\_Topo\_Geometry\_Layer, [4-36](#)  
   DeleteSavepoint, [4-37](#)  
   DisableVersioning, [4-39](#)

DBMS\_WM package (*continued*)  
   EnableVersioning, [4-42](#)  
   Export, [4-45](#)  
   Export\_Schemas, [4-48](#)  
   FindRICSet, [4-51](#)  
   FreezeWorkspace, [4-52](#)  
   GetBulkLoadVersion, [4-54](#)  
   GetConflictWorkspace, [4-56](#)  
   GetDiffVersions, [4-56](#)  
   GetLockMode, [4-57](#)  
   GetMultiWorkspaces, [4-57](#)  
   GetOpContext, [4-58](#)  
   GetOriginalDDL, [4-59](#)  
   GetPhysicalTableName, [4-60](#)  
   GetPrivs, [4-61](#)  
   GetSessionInfo, [4-61](#)  
   GetSystemParameter, [4-63](#)  
   GetValidFrom, [4-63](#)  
   GetValidTill, [4-64](#)  
   GetVersion, [4-65](#)  
   GetWMMetadataSpace, [4-65](#)  
   GetWorkspace, [4-66](#)  
   GotoDate, [4-66](#)  
   GotoSavepoint, [4-68](#)  
   GotoWorkspace, [4-69](#)  
   GrantGraphPriv, [4-69](#)  
   GrantPrivsOnPolicy, [4-71](#)  
   GrantSystemPriv, [4-72](#)  
   GrantWorkspacePriv, [4-73](#)  
   Import, [4-75](#)  
   Import\_Schemas, [4-78](#)  
   Initialize\_After\_Import, [4-80](#)  
   IsWorkspaceOccupied, [4-81](#)  
   LockRows, [4-82](#)  
   MergeTable, [4-84](#)  
   MergeWorkspace, [4-86](#)  
   Move\_Proc, [4-88](#)  
   PurgeTable, [4-89](#)  
   RecoverAllMigratingTables, [4-90](#)  
   RecoverFromDroppedUser, [4-91](#)  
   RecoverMigratingTable, [4-93](#)  
   RefreshTable, [4-94](#)  
   RefreshWorkspace, [4-95](#)  
   RemoveAsParentWorkspace, [4-96](#)  
   RemoveDeferredWorkspaces, [4-98](#)  
   RemoveUserDefinedHint, [4-99](#)  
   RemoveWorkspace, [4-99](#)  
   RemoveWorkspaceTree, [4-100](#)  
   RenameSavepoint, [4-102](#)  
   RenameWorkspace, [4-102](#)  
   ResolveConflicts, [4-103](#)  
   RevokeGraphPriv, [4-105](#)  
   RevokeSystemPriv, [4-107](#)  
   RevokeWorkspacePriv, [4-108](#)  
   RollbackBulkLoading, [4-109](#)

## DBMS\_WM package (continued)

- RollbackDDL, [4-110](#)
  - RollbackResolve, [4-111](#)
  - RollbackTable, [4-112](#)
  - RollbackToSP, [4-113](#)
  - RollbackWorkspace, [4-114](#)
  - SetCaptureEvent, [4-115](#)
  - SetCompressWorkspace, [4-117](#)
  - SetConflictWorkspace, [4-118](#)
  - SetDiffVersions, [4-119](#)
  - SetLockingOFF, [4-120](#)
  - SetLockingON, [4-121](#)
  - SetMultiWorkspaces, [4-122](#)
  - SetSystemParameter, [4-123](#)
  - SetTriggerEvents, [4-124](#)
  - SetValidTime, [4-126](#)
  - SetValidTimeFilterOFF, [4-126](#)
  - SetValidTimeFilterON, [4-127](#)
  - SetWMValidUpdateModeOFF, [4-128](#)
  - SetWMValidUpdateModeON, [4-128](#)
  - SetWoOverwriteOFF, [4-129](#)
  - SetWoOverwriteON, [4-130](#)
  - SetWorkspaceLockModeOFF, [4-130](#)
  - SetWorkspaceLockModeON, [4-131](#)
  - UnfreezeWorkspace, [4-133](#)
  - UnlockRows, [4-134](#)
  - UseDefaultValuesForNulls, [4-136](#)
- DDL (data definition language) operations requirements and restrictions, [1-34](#)
- Delete\_Topo\_Geometry\_Layer procedure, [4-36](#)
- DeleteSavepoint procedure, [4-37](#)
- deleting
- deferred workspace, [4-98](#)
  - savepoints, [4-37](#)
  - workspace, [1-9](#), [4-99](#)
- difference views (xxx\_DIFF), [5-26](#)
- DisableVersioning procedure, [4-39](#)
- disabling
- workspace changes, [4-52](#)

## E

- 
- effective dating
- See valid time support
- EnableVersioning procedure, [4-42](#)
- event parameters, [2-2](#)
- events (Workspace Manager), [2-1](#)
- ALLOW\_CAPTURE\_EVENTS system parameter, [2-3](#)
  - asynchronous notification, [2-6](#)
  - capturing, [4-115](#)
  - event parameters, [2-2](#)
  - list of events, [2-2](#)
  - listening for, [2-6](#)
  - WM\_EVENTS\_INFO view, [5-24](#)

## example

- conflict resolution, [5-25](#)
  - using Workspace Manager (Oracle sample schemas), [1-52](#)
- exclusive locks, [1-16](#), [4-121](#)
- explicit savepoints, [1-5](#)
- export considerations, [1-31](#)
- Export procedure, [4-45](#)
- Export\_Schemas procedure, [4-48](#)

## F

- 
- FindRICSet procedure, [4-51](#)
- foreign keys with version-enabled tables, [1-36](#)
- FreezeWorkspace procedure, [4-52](#)
- freezing
- workspace changes, [1-8](#), [4-52](#)

## G

- 
- GetBulkLoadVersion function, [4-54](#)
- GetConflictWorkspace function, [4-56](#)
- GetDiffVersions function, [4-56](#)
- GetLockMode function, [4-57](#)
- GetMultiWorkspaces function, [4-57](#)
- GetOpContext function, [4-58](#)
- GetOriginalDDL procedure, [4-59](#)
- GetPhysicalTableName function, [4-60](#)
- GetPrivs function, [4-61](#)
- GetSessionInfo procedure, [4-61](#)
- GetSystemParameter function, [4-63](#)
- GetValidFrom function, [4-63](#)
- GetValidTill function, [4-64](#)
- GetVersion function, [4-65](#)
- GetWMMetadataSpace function, [4-65](#)
- GetWorkspace function, [4-66](#)
- GotoDate procedure, [4-66](#)
- GotoSavepoint procedure, [4-68](#)
- GotoWorkspace procedure, [4-69](#)
- grant option, [1-25](#)
- GrantGraphPriv procedure, [4-69](#)
- granting
- Workspace Manager privileges
    - multiparent graph workspaces, [4-69](#)
    - OLS-related, [4-71](#)
    - system, [4-72](#)
    - workspace, [4-73](#)
- GrantPrivsOnPolicy procedure, [4-71](#)
- GrantSystemPriv procedure, [4-72](#)
- GrantWorkspacePriv procedure, [4-73](#)

## H

---

hierarchy  
 removing, [4-100](#)  
 workspaces, [1-4](#)  
 historical data, [1-13](#)  
 history option  
 EnableVersioning procedure, [4-42](#)  
 history views (xxx\_HIST), [5-27](#)

## I

---

implicit savepoints, [1-6](#)  
 import considerations, [1-31](#)  
 Import procedure, [4-75](#)  
 Import\_Schemas procedure, [4-78](#)  
 infrastructure for version-enabling of tables, [1-12](#)  
 Initialize\_After\_Import procedure, [4-80](#)  
 invisible index  
 supported on version-enabled table, [1-35](#)  
 IsWorkspaceOccupied function, [4-81](#)

## J

---

join index  
 cannot be created or dropped on version-enabled table, [1-35](#)

## L

---

LATEST savepoint, [1-6](#)  
 length of object names  
 maximums for Workspace Manager, [1-12](#)  
 LOB columns with versioned tables, [4-31](#)  
 lock management, [1-16](#), [1-46](#)  
 with DML operations on tables with referential integrity constraints, [1-38](#)  
 lock mode  
 getting, [4-57](#)  
 lock views (xxx\_LOCK), [5-28](#)  
 locking table rows, [4-82](#)  
 LockRows procedure, [4-82](#)  
 locks  
 disabling, [4-120](#)  
 enabling, [4-121](#)  
 exclusive, [1-16](#)  
 shared, [1-16](#)  
 version-exclusive, [1-16](#)  
 workspace-exclusive, [1-16](#)  
 logging of modifications  
 EnableVersioning history option, [4-44](#)  
 history views (xxx\_HIST), [5-27](#)  
 long transactions, [1-1](#)

LT tables  
 created for Workspace Manager infrastructure, [1-12](#)  
 getting name of \_LT (physical) table, [4-60](#)

## M

---

materialized views  
 version management with, [1-41](#)  
 MergeTable procedure, [4-84](#)  
 MergeWorkspace procedure, [4-86](#)  
 merging  
 table changes, [4-84](#)  
 workspaces, [1-7](#), [4-86](#)  
 metadata space  
 getting, [4-65](#)  
 Move\_Proc procedure, [4-88](#)  
 multilevel referential integrity constraints, [1-37](#)  
 multiparent workspaces, [1-11](#)  
 multiworkspace views (xxx\_MW), [5-29](#)

## N

---

name length of database objects  
 maximums for Workspace Manager, [1-12](#)  
 nonsequenced update operations, [3-15](#)  
 null values  
 using default values for, [4-136](#)

## O

---

OE.WAREHOUSES table  
 Workspace Manager example, [1-52](#)  
 OLS (Oracle Label Security)  
 DDL operations on version-enabled tables, [1-35](#)  
 granting privileges related to, [4-71](#)  
 operation context  
 getting, [4-58](#)  
 operators for valid time support, [3-5](#)  
 WM\_CONTAINS, [3-6](#)  
 WM\_EQUALS, [3-7](#)  
 WM\_GREATERTHAN, [3-8](#)  
 WM\_INTERSECTION, [3-8](#)  
 WM\_LDIFF, [3-9](#)  
 WM\_LESSTHAN, [3-10](#)  
 WM\_MEETS, [3-11](#)  
 WM\_OVERLAPS, [3-12](#)  
 WM\_RDIFF, [3-12](#)  
 Oracle Label Security (OLS)  
 DDL operations on version-enabled tables, [1-35](#)  
 granting privileges related to, [4-71](#)



Oracle sample schemas  
 Workspace Manager example, [1-52](#)  
 OWM\_VERSION  
 Workspace Manager version number, [5-24](#)

## P

---

parent row, [1-8](#)  
 parent workspace, [1-4](#)  
 conflicts with, [4-118](#)  
 partitioned index  
 cannot be created or dropped on version-enabled table, [1-35](#)  
 physical table name  
 and Workspace Manager infrastructure, [1-12](#)  
 getting, [4-60](#)  
 privilege management, [1-46](#)  
 privileges  
 description, [1-24](#)  
 getting, [4-61](#)  
 grant option, [1-25](#)  
 granted to PUBLIC user group, [1-15](#)  
 granting, [4-72](#), [4-73](#)  
 multiparent graph workspaces, [4-69](#)  
 OLS-related, [4-71](#)  
 managing, [1-24](#)  
 revoking, [1-25](#), [4-107](#), [4-108](#)  
 multiparent graph workspaces, [4-105](#)  
 viewing users having Workspace Manager system-level privileges, [5-17](#)  
 PurgeTable procedure, [4-89](#)

## R

---

RecoverAllMigratingTables procedure, [4-90](#)  
 RecoverFromDroppedUser procedure, [4-91](#)  
 RecoverMigratingTable procedure, [4-93](#)  
 referential integrity constraints  
 finding affected tables, [4-51](#)  
 lock management with DML operations, [1-38](#)  
 referential integrity support, [1-36](#)  
 multilevel constraints, [1-37](#)  
 refreshing  
 tables, [4-94](#)  
 workspaces, [4-95](#)  
 RefreshTable procedure, [4-94](#)  
 RefreshWorkspace procedure, [4-95](#)  
 removable savepoints, [1-6](#)  
 RemoveAsParentWorkspace procedure, [4-96](#)  
 RemoveDeferredWorkspaces procedure, [4-98](#)  
 RemoveUserDefinedHint procedure, [4-99](#)  
 RemoveWorkspace procedure, [4-99](#)  
 RemoveWorkspaceTree procedure, [4-100](#)  
 removing deferred workspaces, [4-98](#)  
 removing workspaces, [1-9](#), [4-99](#)

RenameSavepoint procedure, [4-102](#)  
 RenameWorkspace procedure, [4-102](#)  
 renaming  
 savepoint, [4-102](#)  
 workspace, [4-102](#)  
 reserved words and characters with Workspace Manager, [1-42](#)  
 ResolveConflicts procedure, [4-103](#)  
 resolving conflicts, [4-103](#)  
 beginning, [4-16](#)  
 committing, [4-22](#)  
 rolling back, [4-111](#)  
 RETURNING clause  
 not supported for INSERT or UPDATE on version-enabled tables, [1-13](#)  
 reverse index  
 supported on version-enabled table, [1-35](#)  
 RevokeGraphPriv procedure, [4-105](#)  
 RevokeSystemPriv procedure, [4-107](#)  
 RevokeWorkspacePriv procedure, [4-108](#)  
 revoking privileges, [1-25](#), [4-107](#), [4-108](#)  
 multiparent graph workspaces, [4-105](#)  
 ROLE\_WM\_PRIVS view, [5-20](#)  
 RollbackBulkLoading procedure, [4-109](#)  
 RollbackDDL procedure, [4-110](#)  
 RollbackResolve procedure, [4-111](#)  
 RollbackTable procedure, [4-112](#)  
 RollbackToSP procedure, [4-113](#)  
 RollbackWorkspace procedure, [4-114](#)  
 rolling back  
 changes in a table, [4-112](#)  
 workspace changes, [1-7](#), [4-114](#)  
 workspaces to savepoint, [4-113](#)  
 row versions  
 creation of, [1-13](#)  
 row-level security (VPD)  
 Workspace Manager considerations, [1-40](#)  
 rows  
 locking, [4-82](#)  
 unlocking, [4-134](#)  
 rule-based subscription for Workspace Manager events, [2-5](#)

## S

---

sample schemas  
 Workspace Manager example, [1-52](#)  
 savepoint management, [1-46](#)  
 savepoints, [1-3](#)  
 altering description of, [4-8](#)  
 as alternative to creating child workspaces, [1-6](#)  
 creating, [4-33](#)  
 deleting, [4-37](#)  
 explicit, [1-5](#)



savepoints (*continued*)  
 going to, 4-68  
 implicit, 1-6  
 removable, 1-6  
 renaming, 4-102  
 rolling back to, 4-113

sequenced delete operations, 3-15  
 disabling, 4-128  
 enabling, 4-128

sequenced update operations, 3-14  
 disabling, 4-128  
 enabling, 4-128

session context, 1-15  
 GetSessionInfo function, 4-61

SET NULL constraints  
 not supported for version-enabled tables, 1-40

SetCaptureEvent procedure, 4-115

SetCompressWorkspace procedure, 4-117

SetConflictWorkspace procedure, 4-118

SetDiffVersions procedure, 4-119

SetLockingOFF procedure, 4-120

SetLockingON procedure, 4-121

SetMultiWorkspaces procedure, 4-122

SetSystemParameter procedure, 4-123

SetTriggerEvents procedure, 4-124

SetValidTime procedure, 4-126

SetValidTimeFilterOFF procedure, 4-126

SetValidTimeFilterON procedure, 4-127

SetWMValidUpdateModeOFF procedure, 4-128

SetWMValidUpdateModeON procedure, 4-128

SetWoOverwriteOFF procedure, 4-129

SetWoOverwriteON procedure, 4-130

SetWorkspaceLockModeOFF procedure, 4-130

SetWorkspaceLockModeON procedure, 4-131

shared locks, 1-16, 4-121

skeleton tables, 1-34

spatial topologies  
 version management with, 1-41

subscription (rule-based) for Workspace Manager  
 events, 2-5

synonyms  
 support for, 1-40

system parameters  
 for Workspace Manager, 1-25  
 shown in WM\_INSTALLATION view, 5-24

system privileges, 4-72  
 viewing users having Workspace Manager  
 system-level privileges, 5-17

---

## T

table management, 1-43

table name  
 and Workspace Manager infrastructure, 1-12

table name (*continued*)  
 getting physical table name, 4-60  
 maximum length for Workspace Manager, 1-12

table synonyms, 1-40

topologies  
 version management with, 1-41

topology geometry layer  
 adding, 4-4  
 deleting, 4-36

trigger events  
 setting, 4-124

triggers on version-enabled tables, 1-40

---

## U

UnfreezeWorkspace procedure, 4-133

unfreezing  
 workspaces, 1-8, 4-133

unique constraints  
 support with Workspace Manager, 1-39

unlocking  
 table rows, 4-134

UnlockRows procedure, 4-134

UseDefaultValuesForNulls procedure, 4-136

USER\_MP\_GRAPH\_WORKSPACES view, 5-20

USER\_MP\_PARENT\_WORKSPACES view, 5-20

USER\_REMOVED\_WORKSPACES view, 5-20

USER\_WM\_CONS\_COLUMNS view, 5-21

USER\_WM\_CONSTRAINTS view, 5-21

USER\_WM\_IND\_COLUMNS view, 5-21

USER\_WM\_IND\_EXPRESSIONS view, 5-21

USER\_WM\_LOCKED\_TABLES view, 5-21

USER\_WM\_MODIFIED\_TABLES view, 5-21

USER\_WM\_POLICIES view, 5-21

USER\_WM\_PRIVS view, 5-22

USER\_WM\_RIC\_INFO view, 5-22

USER\_WM\_TAB\_TRIGGERS view, 5-22

USER\_WM\_VERSIONED\_TABLES view, 5-22

USER\_WM\_VT\_ERRORS view, 5-23

USER\_WORKSPACE\_PRIVS view, 5-23

USER\_WORKSPACE\_SAVEPOINTS view, 5-23

USER\_WORKSPACES view, 5-23

user-defined hints  
 adding, 4-7  
 removing, 4-99

---

## V

valid time filter  
 removing, 4-126  
 setting, 4-127

valid time support, 3-1  
 altering version-enabled table to add valid  
 time support, 4-9

valid time support (*continued*)  
 operators, 3-5  
 setting valid time for session, 4-126

version number  
 Workspace Manager (OWM\_VERSION),  
 5-24

version-enabled tables  
 definition, 1-3  
 maximum name length for Workspace  
 Manager, 1-12

version-exclusive locks, 1-16

versioning  
 disabling, 4-39  
 enabling, 4-42  
 infrastructure created for, 1-12

VIEW\_WO\_OVERWRITE mode  
 disabling, 4-129  
 enabling, 4-130

virtual private databases  
 with Workspace Manager, 1-40

VPD (virtual private database)  
 with Workspace Manager, 1-40

## W

---

WM\_ADMIN\_ROLE role, 1-25

WM\_COMPRESS\_BATCH\_SIZES view, 5-23

WM\_COMPRESSIBLE\_TABLES view, 5-24

WM\_CONTAINS operator, 3-6

WM\_EQUALS operator, 3-7

WM\_EVENTS\_INFO view, 5-24

WM\_GREATER\_THAN operator, 3-8

WM\_INSTALLATION view, 5-24

WM\_INTERSECTION operator, 3-8

WM\_LDIFF operator, 3-9

WM\_LESSTHAN operator, 3-10

WM\_MEETS operator, 3-11

WM\_OVERLAPS operator, 3-12

WM\_RDIFF operator, 3-12

WMSYS schema and user, 1-15

workspace lock mode  
 disabling, 4-130  
 enabling, 4-131

Workspace Manager metadata space  
 getting, 4-65

Workspace Manager version  
 getting, 4-65

workspace-exclusive locks, 1-16

workspaces  
 altering description of, 4-12  
 changing type, 4-17  
 child  
   as alternative to creating savepoints, 1-6  
 compressing, 4-23, 4-27  
 continually refreshed, 1-10, 4-17, 4-35  
 copying, 4-32  
 creating, 4-34  
 freezing, 1-8, 4-52  
 getting, 4-66  
 going to, 4-69  
 hierarchy, 1-4  
 management of, 1-3, 1-44  
 merging, 1-7  
 multiparent, 1-11  
 renaming, 4-102  
 rolling back, 1-7  
 unfreezing, 1-8, 4-133