

Oracle® Database

Workspace Manager Developer's Guide



19c
E94792-01
January 2019

ORACLE®

Copyright © 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Primary Author: Chuck Murray

Contributors: 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 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 installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. 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 and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon 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, Opteron, the AMD logo, and the AMD Opteron 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	1-5
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	1-19
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	1-37
1.9.1	Referential Integrity Support	1-37
1.9.1.1	Locking with DML Operations on Tables with Referential Integrity Constraints	1-39
1.9.2	Unique Constraints	1-40
1.9.3	SET NULL Constraints	1-40
1.10	Triggers on Version-Enabled Tables	1-40
1.11	Virtual Private Database Considerations	1-41
1.12	Support for Table Synonyms	1-41
1.13	Materialized View Support	1-41
1.14	Spatial and Graph Topology Support	1-42
1.14.1	Locking Considerations with Topologies	1-42
1.14.2	Additional Considerations with Topologies	1-43
1.15	Workspace Manager Reserved Words and Characters	1-43
1.16	DBMS_WM Subprogram Categories	1-44
1.16.1	Table Management Subprograms	1-44
1.16.2	Workspace Management Subprograms	1-45
1.16.3	Savepoint Management Subprograms	1-46
1.16.4	Privilege Management Subprograms	1-47
1.16.5	Lock Management Subprograms	1-47
1.16.6	Conflict Management Subprograms	1-48
1.16.7	Bulk Load Support Subprograms	1-48
1.17	Simplified Examples Using Workspace Manager	1-49
1.17.1	Example: Marketing Budget Options	1-49
1.17.2	Example: Warehouse Expansion Options	1-53

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	3-14
3.6.2	Data Manipulation (DML) Operations	3-14
3.6.2.1	Update Operations	3-14
3.6.2.2	Insert Operations	3-16
3.7	Constraint Management for Valid Time Support	3-16
3.7.1	Referential Integrity Constraints	3-16
3.7.2	Unique Constraints	3-17
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	3-18
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	4-17
4.10	ChangeWorkspaceType	4-17
4.11	CommitBulkLoading	4-18
4.12	CommitDDL	4-21
4.13	CommitResolve	4-23
4.14	CompressWorkspace	4-24
4.15	CompressWorkspaceTree	4-28
4.16	CopyForUpdate	4-32
4.17	CopyWorkspace	4-33
4.18	CreateSavepoint	4-34
4.19	CreateWorkspace	4-35
4.20	Delete_Topo_Geometry_Layer	4-37
4.21	DeleteSavepoint	4-38
4.22	DisableVersioning	4-40
4.23	EnableVersioning	4-43
4.24	Export	4-46
4.25	Export_Schemas	4-50
4.26	FindRICSet	4-52
4.27	FreezeWorkspace	4-53
4.28	GetBulkLoadVersion	4-55
4.29	GetConflictWorkspace	4-57
4.30	GetDiffVersions	4-57
4.31	GetLockMode	4-58
4.32	GetMultiWorkspaces	4-58
4.33	GetOpContext	4-59
4.34	GetOriginalDDL	4-60
4.35	GetPhysicalTableName	4-61
4.36	GetPrivs	4-62
4.37	GetSessionInfo	4-62
4.38	GetSystemParameter	4-64
4.39	GetValidFrom	4-64
4.40	GetValidTill	4-65
4.41	GetVersion	4-66
4.42	GetWMMetadataSpace	4-66
4.43	GetWorkspace	4-67
4.44	GotoDate	4-67

4.45	GotoSavepoint	4-69
4.46	GotoWorkspace	4-70
4.47	GrantGraphPriv	4-71
4.48	GrantPrivsOnPolicy	4-72
4.49	GrantSystemPriv	4-73
4.50	GrantWorkspacePriv	4-75
4.51	Import	4-76
4.52	Import_Schemas	4-79
4.53	Initialize_After_Import	4-81
4.54	IsWorkspaceOccupied	4-82
4.55	LockRows	4-83
4.56	MergeTable	4-85
4.57	MergeWorkspace	4-87
4.58	Move_Proc	4-89
4.59	PurgeTable	4-90
4.60	RecoverAllMigratingTables	4-91
4.61	RecoverFromDroppedUser	4-92
4.62	RecoverMigratingTable	4-93
4.63	RefreshTable	4-95
4.64	RefreshWorkspace	4-96
4.65	RemoveAsParentWorkspace	4-97
4.66	RemoveDeferredWorkspaces	4-98
4.67	RemoveUserDefinedHint	4-99
4.68	RemoveWorkspace	4-100
4.69	RemoveWorkspaceTree	4-101
4.70	RenameSavepoint	4-103
4.71	RenameWorkspace	4-103
4.72	ResolveConflicts	4-104
4.73	RevokeGraphPriv	4-106
4.74	RevokeSystemPriv	4-108
4.75	RevokeWorkspacePriv	4-109
4.76	RollbackBulkLoading	4-110
4.77	RollbackDDL	4-111
4.78	RollbackResolve	4-112
4.79	RollbackTable	4-113
4.80	RollbackToSP	4-114
4.81	RollbackWorkspace	4-116
4.82	SetCaptureEvent	4-117
4.83	SetCompressWorkspace	4-118
4.84	SetConflictWorkspace	4-119
4.85	SetDiffVersions	4-120

4.86	SetLockingOFF	4-122
4.87	SetLockingON	4-122
4.88	SetMultiWorkspaces	4-124
4.89	SetSystemParameter	4-125
4.90	SetTriggerEvents	4-126
4.91	SetValidTime	4-128
4.92	SetValidTimeFilterOFF	4-128
4.93	SetValidTimeFilterON	4-129
4.94	SetWMValidUpdateModeOFF	4-130
4.95	SetWMValidUpdateModeON	4-130
4.96	SetWoOverwriteOFF	4-131
4.97	SetWoOverwriteON	4-132
4.98	SetWorkspaceLockModeOFF	4-132
4.99	SetWorkspaceLockModeON	4-133
4.100	UnfreezeWorkspace	4-135
4.101	UnlockRows	4-136
4.102	UseDefaultValuesForNulls	4-138

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	5-18
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	5-21
5.31	USER_REMOVED_WORKSPACES	5-21
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	5-22
5.37	USER_WM_MODIFIED_TABLES	5-22
5.38	USER_WM_POLICIES	5-22
5.39	USER_WM_PRIVS	5-22
5.40	USER_WM_RIC_INFO	5-22
5.41	USER_WM_TAB_TRIGGERS	5-23
5.42	USER_WM_VERSIONED_TABLES	5-23
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	5-25
5.51	xxx_CONF Views	5-25
5.52	xxx_DIFF Views	5-26
5.53	xxx_HIST Views	5-28
5.54	xxx_LOCK Views	5-29
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	1-13
1-3	Workspace Manager Lock Modes and Data Modification Ability	1-17
1-4	Operations and Incompatible Freeze Modes for Workspace Types	1-20
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	1-45
1-10	Savepoint Management Subprograms	1-47
1-11	Privilege Management Subprograms	1-47
1-12	Lock Management Subprograms	1-47
1-13	Conflict Management Subprograms	1-48
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	4-16
4-9	BeginResolve Procedure Parameters	4-17
4-10	ChangeWorkspaceType Procedure Parameters	4-18
4-11	CommitBulkLoading Procedure Parameters	4-19
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	4-29

4-16	CopyForUpdate Procedure Parameters	4-32
4-17	CopyWorkspace Procedure Parameters	4-33
4-18	CreateSavepoint Procedure Parameters	4-34
4-19	CreateWorkspace Procedure Parameters	4-35
4-20	Delete_Topo_Geometry_Layer Procedure Parameters	4-37
4-21	DeleteSavepoint Procedure Parameters	4-38
4-22	DisableVersioning Procedure Parameters	4-41
4-23	EnableVersioning Procedure Parameters	4-43
4-24	Export Procedure Parameters	4-47
4-25	Export_Schemas Procedure Parameters	4-50
4-26	FindRICSet Procedure Parameters	4-52
4-27	FreezeWorkspace Procedure Parameters	4-54
4-28	GetBulkLoadVersion Function Parameters	4-56
4-29	GetOriginalDDL Procedure Parameters	4-60
4-30	GetPhysicalTableName Function Parameters	4-61
4-31	GetPrivs Function Parameters	4-62
4-32	GetSessionInfo Procedure Parameters	4-63
4-33	GetSystemParameter Procedure Parameters	4-64
4-34	GotoDate Procedure Parameters	4-67
4-35	GotoSavepoint Procedure Parameters	4-69
4-36	GotoWorkspace Procedure Parameters	4-70
4-37	GrantGraphPriv Procedure Parameters	4-71
4-38	GrantPrivsOnPolicy Procedure Parameters	4-73
4-39	GrantSystemPriv Procedure Parameters	4-73
4-40	GrantWorkspacePriv Procedure Parameters	4-75
4-41	Import Procedure Parameters	4-77
4-42	Import_Schemas Procedure Parameters	4-79
4-43	Initialize_After_Import Procedure Parameters	4-81
4-44	IsWorkspaceOccupied Function Parameters	4-82
4-45	LockRows Procedure Parameters	4-83
4-46	MergeTable Procedure Parameters	4-85
4-47	MergeWorkspace Procedure Parameters	4-87
4-48	Move_Proc Procedure Parameters	4-89
4-49	PurgeTable Procedure Parameters	4-90
4-50	RecoverAllMigratingTables Procedure Parameters	4-92
4-51	RecoverFromDroppedUser Procedure Parameters	4-93
4-52	RecoverMigratingTable Procedure Parameters	4-94

4-53	RefreshTable Procedure Parameters	4-95
4-54	RefreshWorkspace Procedure Parameters	4-96
4-55	RemoveAsParentWorkspace Procedure Parameters	4-98
4-56	RemoveWorkspace Procedure Parameters	4-99
4-57	RemoveUserDefinedHint Procedure Parameters	4-100
4-58	RemoveWorkspace Procedure Parameters	4-101
4-59	RemoveWorkspaceTree Procedure Parameters	4-102
4-60	RenameSavepoint Procedure Parameters	4-103
4-61	RenameWorkspace Procedure Parameters	4-104
4-62	ResolveConflicts Procedure Parameters	4-104
4-63	RevokeGraphPriv Procedure Parameters	4-107
4-64	RevokeSystemPriv Procedure Parameters	4-108
4-65	RevokeWorkspacePriv Procedure Parameters	4-109
4-66	RollbackBulkLoading Procedure Parameters	4-110
4-67	RollbackDDL Procedure Parameters	4-111
4-68	RollbackResolve Procedure Parameters	4-112
4-69	RollbackTable Procedure Parameters	4-113
4-70	RollbackToSP Procedure Parameters	4-115
4-71	RollbackWorkspace Procedure Parameters	4-116
4-72	SetCaptureEvent Procedure Parameters	4-117
4-73	SetCompressWorkspace Procedure Parameters	4-118
4-74	SetConflictWorkspace Procedure Parameters	4-119
4-75	SetDiffVersions Procedure Parameters	4-120
4-76	SetLockingON Procedure Parameters	4-123
4-77	SetMultiWorkspaces Procedure Parameters	4-124
4-78	SetSystemParameter Procedure Parameters	4-125
4-79	SetTriggerEvents Procedure Parameters	4-126
4-80	SetValidTime Procedure Parameters	4-128
4-81	SetValidTimeFilterON Procedure Parameters	4-129
4-82	SetWorkspaceLockModeOFF Procedure Parameters	4-133
4-83	SetWorkspaceLockModeON Procedure Parameters	4-134
4-84	UnfreezeWorkspace Procedure Parameters	4-136
4-85	UnlockRows Procedure Parameters	4-136
4-86	UseDefaultValuesForNulls Procedure Parameters	4-138
5-1	Columns in the xxx_CONF Views	5-25
5-2	Columns in the xxx_DIFF Views	5-27
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	5-30

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
boldface	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.
monospace	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.

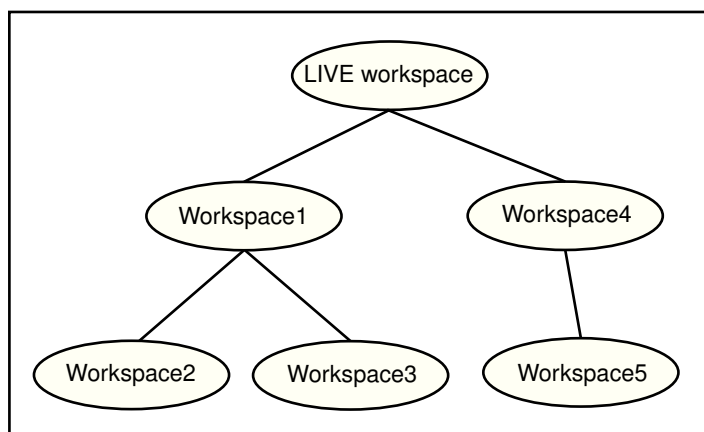
- [Workspace Hierarchy](#)
- [Using Savepoints](#)
- [Merging and Rolling Back Workspace Changes](#)
- [Resolving Conflicts Before a Merge or Refresh Operation](#)
- [Freezing and Unfreezing Workspaces](#)
- [Removing Workspaces](#)
- [Using Workspace Manager Events](#)
- [Autocommitting of Workspace 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

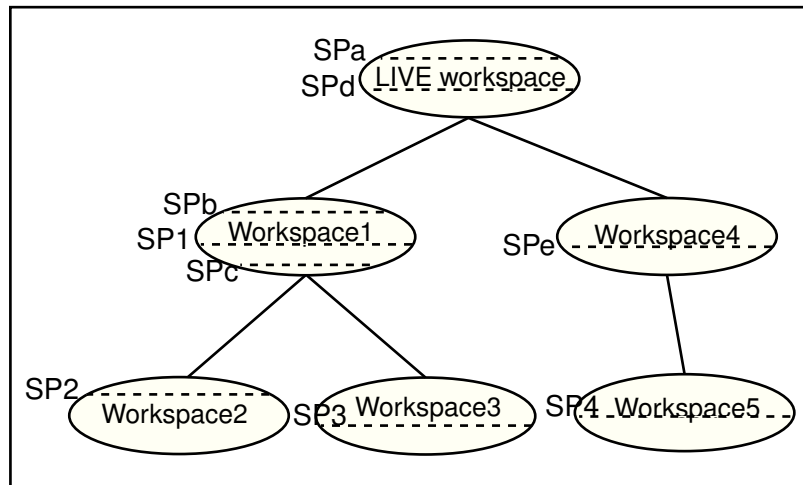


See also [Design Issue: Savepoint or Child Workspace?](#) for a discussion of design issues in deciding whether to create a child workspace or a savepoint for certain needs

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
BeginResolve	Specified workspace: 1WRITER
MergeWorkspace	Specified workspace: NO_ACCESS Parent workspace: READ_ONLY
CompressWorkspace	Specified workspace: NO_ACCESS (Also, checks to ensure that there are no sessions on savepoints other than LATEST.)
CompressWorkspaceTree	Specified workspace: NO_ACCESS (Also, checks to ensure that there are no sessions on savepoints other than LATEST.)
CreateSavepoint	Specified workspace: READ_ONLY
DeleteSavepoint	Specified workspace: NO_ACCESS
CreateWorkspace	Specified workspace: READ_ONLY
RemoveWorkspace	Specified workspace: NO_ACCESS
RefreshWorkspace	Specified workspace: READ_ONLY Parent workspace: READ_ONLY
RollbackResolve	Specified workspace: 1WRITER
RollbackWorkspace	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.

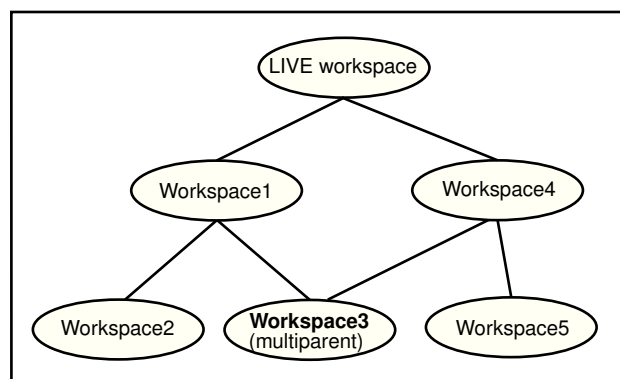
If a workspace is not continually refreshed, you must call the [RefreshWorkspace](#) procedure whenever you want to ensure that data changes in its parent workspace are visible in the workspace.

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 `>= 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	<DB-identifier-length> – 5
Column	<DB-identifier-length> – 2
Index	<DB-identifier-length> (<DB-identifier-length> — 4 if the index is created or altered between calls to the BeginDDL and CommitDDL procedures)
Trigger	<DB-identifier-length>
Constraint	<DB-identifier-length> (<DB-identifier-length> – 4 if the constraint is created or altered between calls to the BeginDDL and CommitDDL 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_W0_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_R EMOVAL, NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_ SESSION, WM_ONLY	None	N/A
BeginResolve	None	N/A	Shared Sub eXclusive	DEFERRED_R EMOVAL	None	N/A
ChangeWorkspaceType	Shared	N/A	Shared Sub eXclusive	DEFERRED_R EMOVAL, NO_ACCESS, ONEWRITER, ONEWRITER_ SESSION, READ_ONLY	Exclusive	NO_ACCESS, ONEWRITER, ONEWRITER_ SESSION, READ_ONLY
CommitResolve	None	N/A	Shared Sub eXclusive	DEFERRED_R EMOVAL	None	N/A
CompressWorkspace	None	N/A	Shared	DEFERRED_R EMOVAL, NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_ SESSION	None	N/A
CompressWorkspace Tree	None	N/A	Shared	DEFERRED_R EMOVAL, NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_ SESSION	None	N/A
CopyWorkspace	None	N/A	Shared	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
CreateSavepoint	None	N/A	Shared (Shared Sub eXclusive when new version must be created)	DEFERRED_R EMOVAL, NO_ACCESS	None	N/A
CreateWorkspace	None	N/A	Shared (Shared Sub eXclusive when new version must be created)	DEFERRED_R EMOVAL, NO_ACCESS	None	N/A
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 ¹	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

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
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
MergeWorkspace (remove_workspace=>false)	Shared(1)	NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_SESSION	Shared Sub eXclusive	DEFERRED_R EMOVAL, NO_ACCESS	Exclusive	NO_ACCESS
MergeWorkspace (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
PurgeTable	None	N/A	None; table locked in Exclusive mode	None	None	N/A
RefreshTable	Shared(1)	NO_ACCESS	Shared	DEFERRED_R EMOVAL, NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_SESSION	Shared	NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_SESSION
RefreshWorkspace	Shared(1)	NO_ACCESS	Shared	DEFERRED_R EMOVAL, NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_SESSION	Exclusive	NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_SESSION
RemoveDeferredWorkspaces	None	N/A	Exclusive +	N/A	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
RemoveWorkspace	Shared(1)	N/A	Exclusive +	NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_SESSION	Shared	N/A
RenameSavepoint	None	N/A	Exclusive +	DEFERRED_REMOVAL, NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_SESSION	None	N/A
RenameWorkspace	None	N/A	Exclusive +	DEFERRED_REMOVAL, NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_SESSION	None	N/A
RollbackResolve	None	N/A	Shared SubeXclusive	DEFERRED_REMOVAL	None	N/A
RollbackTable	None	N/A	Shared SubeXclusive	DEFERRED_REMOVAL, NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_SESSION	None	N/A
RollbackWorkspace	None	N/A	Shared SubeXclusive	DEFERRED_REMOVAL, NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_SESSION	None	N/A
SetConflictWorkspace	None	NO_ACCESS	Sub Shared	DEFERRED_REMOVAL, NO_ACCESS	None	N/A
SetDiffVersions	None	N/A	Sub Shared	DEFERRED_REMOVAL, 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
SetMultiWorkspaces	None	N/A	Sub Shared	DEFERRED_R EMOVAL, NO_ACCESS	None	N/A
SetWorkspaceLockModeOFF	None	N/A	Shared Sub eXclusive	DEFERRED_R EMOVAL, NO_ACCESS	None	N/A
SetWorkspaceLockModeON	None	N/A	Shared Sub eXclusive	DEFERRED_R EMOVAL, NO_ACCESS	None	N/A
UnfreezeWorkspace	None	N/A	Shared Sub eXclusive	DEFERRED_R EMOVAL	None	N/A
UnlockRows	None	N/A	Shared	DEFERRED_R EMOVAL, NO_ACCESS, READ_ONLY, ONEWRITER, ONEWRITER_ SESSION	None	N/A

¹ 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.

Table 1-5 (Cont.) Workspace Manager Privileges

Privilege	Description
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.
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	ON adds columns to the xxx_HIST views (described in xxx_HIST Views) of version-enabled tables. When a table is version-enabled with valid time support (described in Workspace Manager Valid Time Support), 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. OFF (the default) does not add any columns to the xxx_HIST views.
ALLOW_CAPTURE_EVENTS	ON allows Workspace Manager events (described in Workspace Manager Events) 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. OFF (the default) does not allow Workspace Manager events to be captured.
ALLOW_MULTI_PARENT_WORKSPACES	ON allows multiparent workspaces (described in Multiparent Workspaces) 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. OFF (the default) does not allow multiparent workspaces to be created.
ALLOW_NESTED_TABLE_COLUMNS	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. OFF (the default) does not allow tables containing a nested table column to be version-enabled.

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

Parameter Name	Values
COMPRESS_PARENT_AFTER_REMOVE	ON (the default) implicitly calls the CompressWorkspace procedure to compress the parent workspace after a RemoveWorkspace operation, or after a MergeWorkspace operation in which the <code>remove_workspace</code> parameter was set to true. OFF does not compress the parent workspace in these cases.
CR_WORKSPACE_MODE	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. 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. 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.
CREATEWORKSPACE_SHARED_LOCK	Applies to the creation of workspaces that are not continually refreshed. ON causes the CreateWorkspace procedure to acquire only a Shared lock on the parent workspace when creating the workspace, if possible. This allows multiple CreateWorkspace 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. OFF (the default) causes the CreateWorkspace procedure to acquire a Shared Sub eXclusive (SSX) lock on the parent workspace when creating a workspace. This prevents multiple CreateWorkspace 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.

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

Parameter Name	Values
DEFAULT_WORKSPACE	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 GotoWorkspace procedure is explicitly executed. The default is the LIVE workspace. Any workspace specified must have the ACCESS_WORKSPACE privilege granted to PUBLIC.
FIRE_TRIGGERS_FOR_NONDML_EVENTS	ON (the default) causes user-defined triggers on version-enabled tables to be fired when a workspace non-DML operation (such as MergeWorkspace or MergeTable) is executed, unless later overridden for specific triggers by the SetTriggerEvents procedure. OFF causes user-defined triggers on version-enabled tables not to be fired when a workspace non-DML operation (such as MergeWorkspace or MergeTable) is executed, unless later overridden for specific triggers by the SetTriggerEvents procedure.
KEEP_REMOVED_WORKSPACES_INFO	ON keeps information about any workspaces that are removed. This information will be available in the DBA_REMOVED_WORKSPACES, ALL_REMOVED_WORKSPACES, and USER_REMOVED_WORKSPACES views. ON also adds two new columns to xxx_HIST views (described in xxx_HIST Views): WM_CREATEWORKSPACEID and WM_RETIREWORKSPACEID 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. OFF (the default) does not keep this information about any workspaces that are removed.
NONCR_WORKSPACE_MODE	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. 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.

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

Parameter Name	Values
NUMBER_OF_COMPRESS_BATCHES	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.)
REMOVEWORKSPACE_DEFERRED	Specifies the default value for the <code>defer_option</code> parameter of the RemoveWorkspace and RemoveWorkspaceTree procedures. OFF (the default) causes the RemoveWorkspace and RemoveWorkspaceTree operations to completely remove any rows or locks associated with the removed workspace from any version-enabled table that was modified in the workspace. 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 RemoveDeferredWorkspaces procedure is executed. 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 are released. The rows that are contained within version-enabled tables that are part of the workspace will not be removed until the RemoveDeferredWorkspaces procedure is executed.

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

Parameter Name	Values
ROW_LEVEL_LOCKING	ON takes row-level locks on any rows that need to be merged or refreshed during MergeTable, MergeWorkspace, or RefreshWorkspace 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. OFF (the default) takes workspace-level locks for all workspace operations. This setting prevents MergeTable, MergeWorkspace, and RefreshWorkspace operations from running in parallel while operating on the same parent workspace.
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 MergeTable, MergeWorkspace, or RefreshWorkspace 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 UNLIMITED (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 UNDO_SPACE value. You can override the value of the UNDO_SPACE system parameter by specifying the undo_space parameter in the call to the EnableVersioning procedure.

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

Parameter Name	Values
USE_SCALAR_TYPES_FOR_VALIDTIME	ON causes Workspace Manager to use two columns, named WM_VALIDFROM and WM_VALIDTILL, of type <code>TIMESTAMP WITH TIME ZONE</code>, instead of a single column named WM_VALID (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 WM_PERIOD Data Type.) OFF (the default) causes Workspace Manager to a single column named WM_VALID (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 AlterVersionedTable 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>.
USE_TIMESTAMP_TYPE_FOR_HISTORY	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 WM_CREATETIME and WM_RETIRETIME columns. OFF causes Workspace Manager to use the <code>DATE</code> type for WM_CREATETIME and WM_RETIRETIME columns.

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 NVARCHAR2; 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 DEPARTMENT_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 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
EnableVersioning	Version-enables a table, creating the necessary structures to enable the table to support multiple versions of rows.
DisableVersioning	Deletes all support structures that were created to enable the table to support versioned rows.
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).
SetWoOverwriteON	Enables the VIEW_WO_OVERWRITE history option that was disabled by the SetWoOverwriteOFF procedure.

Table 1-8 (Cont.) Table Management Subprograms

Procedure	Description
BeginDDL	Starts a DDL (data definition language) session for a specified table.
CommitDDL	Commits DDL changes made during a DDL session for a specified table, and ends the DDL session.
RollbackDDL	Rolls back (cancels) DDL changes made during a DDL session for a specified table, and ends the DDL session.
RecoverAllMigratingTables	Attempts to complete the migration process on a table that was left in an inconsistent state after the Workspace Manager migration procedure failed.
RecoverAllMigratingTables	Attempts to complete the migration process on all tables that were left in an inconsistent state after the Workspace Manager migration procedure failed.
CopyForUpdate	Allows LOB columns (BLOB, CLOB, or NCLOB) in version-enabled tables to be modified.
Export	Exports data from a version-enabled table (all rows, or as limited by any combination of several parameters) to a staging table.
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.

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
CreateWorkspace	Creates a new workspace in the database.
GotoWorkspace	Moves the current session to the specified workspace.
SetDiffVersions	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.
GetDiffVersions	Returns the names of the (workspace, savepoint) pairs on which the session has performed the SetDiffVersions operation.
MergeTable	Applies changes to a table (all rows or as specified in the WHERE clause) in a workspace to its parent workspace.
MergeWorkspace	Applies all changes in a workspace to its parent workspace, and optionally removes the workspace.
RollbackWorkspace	Discards all data changes made in the workspace to version-enabled tables.
RollbackTable	Discards all changes made in the workspace to a specified table (all rows or as specified in the WHERE clause).

Table 1-9 (Cont.) Workspace Management Subprograms

Procedure	Description
RollbackToSP	Discards all data changes made in the workspace to version-enabled tables since the specified savepoint.
RefreshTable	Applies to a workspace all changes made to a table (all rows or as specified in the WHERE clause) in its parent workspace.
RefreshWorkspace	Applies to a workspace all changes made in its parent workspace.
AlterWorkspace	Modifies the description of a workspace.
ChangeWorkspaceType	Changes a workspace that is not continually refreshed to be continually refreshed.
RemoveWorkspace	Discards all row versions associated with a workspace and deletes the workspace.
RemoveWorkspaceTree	Discards all row versions associated with a workspace and its descendant workspaces, and deletes the affected workspaces.
FreezeWorkspace	Restricts access to a workspace and the ability of users to make changes in the workspace.
UnfreezeWorkspace	Enables access and changes to a workspace, reversing the effect of the FreezeWorkspace procedure.
CompressWorkspace	Deletes removable savepoints in a workspace, and minimizes the Workspace Manager metadata structures for the workspace.
CompressWorkspaceTree	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.
IsWorkspaceOccupied	Checks whether or not a workspace has any active sessions.
CreateSavepoint	Returns the current workspace for the session.
SetMultiWorkspaces	Makes the specified workspace or workspaces visible in the multiworkspace views for version-enabled tables.
GetMultiWorkspaces	Returns the names of workspaces visible in the multiworkspace views for version-enabled tables.
GetOpContext	Returns the context of the current operation for the current session.
AddAsParentWorkspace	Adds a workspace as a parent workspace to a child workspace in a multiparent workspace environment.
RemoveAsParentWorkspace	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
CreateSavepoint	Creates a savepoint for the current version.
GotoSavepoint	Goes to the specified savepoint in the current workspace.
GotoDate	Goes to a point at or near the specified date and time in the current workspace.
GetSessionInfo	Retrieves information about the current workspace and session context; useful for finding the session's current savepoint or instant in time.
AlterSavepoint	Modifies the description of a savepoint.
DeleteSavepoint	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
GrantWorkspacePriv	Grants workspace-level privileges to users, roles, or PUBLIC.
RevokeWorkspacePriv	Revokes workspace-level privileges from users and roles.
GrantSystemPriv	Grants privileges on all workspaces to users, roles, or PUBLIC.
RevokeSystemPriv	Revokes system-level privileges from users and roles.
GetPrivs	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
SetLockingON	Enables Workspace Manager locking for the current session.
SetLockingOFF	Disables Workspace Manager locking for the current session.
SetWorkspaceLockModeON	Enables Workspace Manager locking for the specified workspace.
SetWorkspaceLockModeOFF	Disables Workspace Manager locking for the specified workspace.

Table 1-12 (Cont.) Lock Management Subprograms

Procedure	Description
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.
LockRows	Controls access to versioned rows in a specified table and to corresponding rows in the parent workspace.
UnlockRows	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
SetConflictWorkspace	Determines whether or not conflicts exist between a workspace and its parent workspace.
GetConflictWorkspace	Returns the name of the workspace on which the session has performed the SetConflictWorkspace procedure.
BeginResolve	Starts a conflict resolution session.
ResolveConflicts	Resolves conflicts between workspaces.
CommitResolve	Ends a conflict resolution session and saves (makes permanent) any changes in the workspace since the BeginResolve procedure was executed.
RollbackResolve	Quits a conflict resolution session and discards all changes in the workspace since the BeginResolve 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
GetBulkLoadVersion	Returns a version number to be specified when you call the BeginBulkLoading procedure.
BeginBulkLoading	Starts the bulk loading process for a version-enabled table.
CommitBulkLoading	Ends the bulk loading process for a version-enabled table by committing the bulk load changes.

Table 1-14 (Cont.) Bulk Loading Support Subprograms

Procedure	Description
RollbackBulkLoading	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 MergeTable is invoked with remove_data set to TRUE.
TABLE_MERGE_WO_REMOVE_DATA	When MergeTable is invoked with remove_data set to FALSE.
TABLE_REFRESH	When RefreshTable is invoked.
TABLE_ROLLBACK	When RollbackTable is invoked.
WORKSPACE_COMPRESS	When CompressWorkspace or CompressWorkspaceTree is invoked.
WORKSPACE_CREATE	When CreateWorkspace is invoked.
WORKSPACE_MERGE_W_REMOVE	When MergeWorkspace is invoked with remove_workspace set to TRUE.
WORKSPACE_MERGE_WO_REMOVE	When MergeWorkspace is invoked with remove_workspace set to FALSE.
WORKSPACE_REFRESH	When RefreshWorkspace is invoked.
WORKSPACE_REMOVE	When RemoveWorkspace or RemoveWorkspaceTree is invoked.
WORKSPACE_ROLLBACK	When RollbackWorkspace 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 Using Savepoints .)

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
event_name	VARCHAR2(128)	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)	A nested table of (name,value) pairs that can contain additional information about the event. For TABLE_xxx events, it has one row containing the WHERE clause string used for the operation. For WORKSPACE_VERSION events, it has one row containing the savepoint name associated with the newly created version.

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

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

    sys.dbms_aq.register(reginfolist, 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
DBMS_WM.MIN_TIME	The minimum (earliest) timestamp value supported by Workspace Manager. Currently the beginning of the day on 01-Jan in the year -4712 (4712 BCE).
DBMS_WM.MAX_TIME	The maximum (latest) timestamp value supported by Workspace Manager. Currently the end of the day (11:59.999999000 pm) on 31-Dec-9999.
DBMS_WM.UNTIL_CHANGED	A timestamp that is treated as DBMS_WM.MAX_TIME 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
EnableVersioning	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> .

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

Subprogram	Valid Time Support
DisableVersioning	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.
GetValidFrom	Returns the <code>validFrom</code> timestamp from the session valid time period.
GetValidTill	Returns the <code>validTill</code> timestamp from the session valid time period.
SetValidTime	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.
SetValidTimeFilterOFF	Removes the valid time filter for the current session.
SetValidTimeFilterON	Sets a valid time filter for the current session (that is, a time to be applied to version-enabled tables).
SetWMValidUpdateModeOFF	Disables sequenced and nonsequenced update operations and sequenced delete operations on tables that have valid time support.
SetWMValidUpdateModeON	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
Adams	30000

WM_VALID(VALIDFROM, VALIDTILL)

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(VALIDFROM, VALIDTILL)	

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(VALIDFROM, VALIDTILL)	

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 versioned-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 validTill value that is not DBMS_WM.UNTIL_CHANGED.	The intersecting portion of the row is modified in the target workspace to equal that of the source row.
The validTill value of the source row is not DBMS_WM.UNTIL_CHANGED, and the value for the target is DBMS_WM.UNTIL_CHANGED.	The intersecting portion of the row is modified and the resulting row has a validTill value equal to that of the source row.

Table 3-3 (Cont.) validTill Values and Intersection Result from Merge or Refresh Operation

validTill Values	Intersection Result
The validTill value of the source row is DBMS_WM.UNTIL_CHANGED, and the value for the target row is not DBMS_WM.UNTIL_CHANGED.	The intersecting portion of the row is modified and the resulting row has a validTill value equal to that of the target row.
The validTill value of both the source and target rows are both DBMS_WM.UNTIL_CHANGED.	The intersecting portion is modified and the resulting row has a validTill value of DBMS_WM.UNTIL_CHANGED.

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.