Oracle9*i*

Application Developer's Guide - Workspace Manager

Release 1 (9.0.1)

June 2001

Part No. A88806-01

ORACLE®

Oracle9*i* Application Developer's Guide - Workspace Manager, Release 1 (9.0.1)

Part No. A88806-01

Primary Author: Chuck Murray

Contributors: Sanjay Agarwal, Gopalan Arun, Ramkrishna Chatterjee, David Mor, Deborah Owens, Chaya Ramanujan, Ramesh Vasudevan

Contents

2 Procedures: Reference

List of Examples

List of Figures

List of Tables

Send Us Your Comments

Oracle9*i* Application Developer's Guide - Workspace Manager, Release 1 (9.0.1)

Part No. A88806-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and the chapter and section or page number (if available). You can send comments to us in the following ways:

- Electronic mail: nedc-doc_us@oracle.com
- FAX: 603.897.3825 Attn: Workspace Manager Documentation
- Postal service:
  Oracle Corporation
  Workspace Manager Documentation
  One Oracle Drive
  Nashua, NH 03062-2804
  USA

If you would like a reply, please include your name and contact information.

If you have problems with the software, please contact Oracle Support Services.

Preface

Oracle9i Application Developer's Guide - Workspace Manager describes the Oracle long-transaction framework built on a workspace management system.

This preface contains these topics:

Audience

Oracle9i Application Developer's Guide - Workspace Manager is intended for application designers and developers who want to use Oracle Database Workspace Manager, often referred to as Workspace Manager, to work with long transactions. It is assumed that you have some experience programming in PL/SQL.

Organization

This document contains the following:

Chapter 1, "Introduction"

Explains workspace management concepts.

Chapter 2, "Procedures: Reference"

Provides reference information about the Workspace Manager application programming interface.

Chapter 3, "Metadata Views"

Describes metadata views created and maintained by Workspace Manager.

Appendix A, "Installing Workspace Manager with Custom Databases"

Describes how to install Workspace Manager with Oracle databases other than the seed database and databases created using the Database Configuration Assistant (DBCA). (Workspace Manager is installed by default in the seed database and databases using DBCA.)

Appendix B, "Error Messages"

Lists the error messages for Workspace Manager, with the cause and suggested user action for each error.

Glossary

Defines important terms specific to Workspace Manager

Related Documentation

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

- *Oracle Spatial User's Guide and Reference*
- *Oracle Call Interface Programmer's Guide*
- *Oracle9i Database Concepts*
- *PL/SQL User's Guide and Reference*

In North America, printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com/`

Customers in Europe, the Middle East, and Africa (EMEA) can purchase documentation from

`http://www.oraclebookshop.com/`

Other customers can contact their Oracle representative to purchase printed documentation.

Conventions

In examples, an implied carriage return occurs at the end of each line, unless otherwise noted. You must press the Return key at the end of a line of input.

The following conventions are used in this guide:

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
. . .	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted
boldface text	Boldface text indicates a term defined in the text, the glossary, or in both locations.
< >	Angle brackets enclose user-supplied names.
[ ]	Brackets enclose optional clauses from which you can choose one or none.
%	The percent sign represents the system prompt on a UNIX system.

Documentation Accessibility

Oracle's goal is to make our products, services, and supporting documentation accessible to the disabled community with good usability. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation

JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

1
Introduction

The need to manage complex, long-duration database transactions is common in today's engineering applications, AM/FM, geographic information systems (GIS), and computer-aided design (CAD) applications.

Conventional database management systems (DBMS) are designed to handle short-duration transactions, such as those in financial applications. However, these techniques are insufficient for engineering applications where a design can take days to complete and multiple users must access the same database. Traditional concurrency control techniques are well-defined and adequate for handling short-duration transactions. These techniques are highly robust for handling very large numbers of short, distinct transactions that are open for milliseconds or minutes. A conventional short transaction locks all records that are changed until the transaction is completed (either merged or rolled back). These concurrency control techniques, however, are insufficient for handling applications that must support long-duration transactions.

Database Workspace Manager, often referred to as Workspace Manager, provides a long-transaction framework built on a workspace management system. It uses a series of short transactions and multiple data versions to implement a complete long-transaction event that maintains atomicity and concurrency. Changes are stored in the database as 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 long transaction are stored persistently, assuring concurrency and consistency.

This chapter explains the terminology used and also describes the workings of this product. You must understand this background information to use Workspace Manager procedures. This chapter has the following main sections:

- Section 1.3, "Lock Management"
- Section 1.4, "Privilege Management"
- Section 1.5, "Import and Export Considerations"
- Section 1.6, "Referential Integrity Support"
- Section 1.7, "Triggers on Version-Enabled Tables"
- Section 1.8, "Procedure Categories"
- Section 1.9, "Simplified Example"

For a complete example of Workspace Manager, see Section 1.9. However, you may want to read the rest of this chapter first, to understand the concepts that the example illustrates.

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 Appendix A, "Installing Workspace Manager with Custom Databases".

1.1 Workspace Management

Workspace management refers to the ability of the database to hold different versions of the same record (that is, row) in one or more workspaces. Users of the database can then change these versions independently. There are two fundamental benefits of versioning in a database system:

- Versioning improves concurrent access of data in the database. In a database without the versioning ability, users wanting to change the same record are serialized by means of locks. Relaxing the locking strictness to improve concurrency leads to undesirable side effects, such as cascading rollbacks.
- Multiple *what-if* analyses can be run against the data simultaneously. Each analysis works on a separate version of the data. After the analyses are complete, the results can be stored in the database for quick lookup.

The unit of versioning in the product is a database table. A table in the database can be **version-enabled**, which means that all rows in the table can now support multiple versions of data. The versioning infrastructure is not visible to the users of

the database. After a table has been version-enabled, users automatically see the correct version of the record in which they are interested.

1.2 Workspace Concepts and Operations

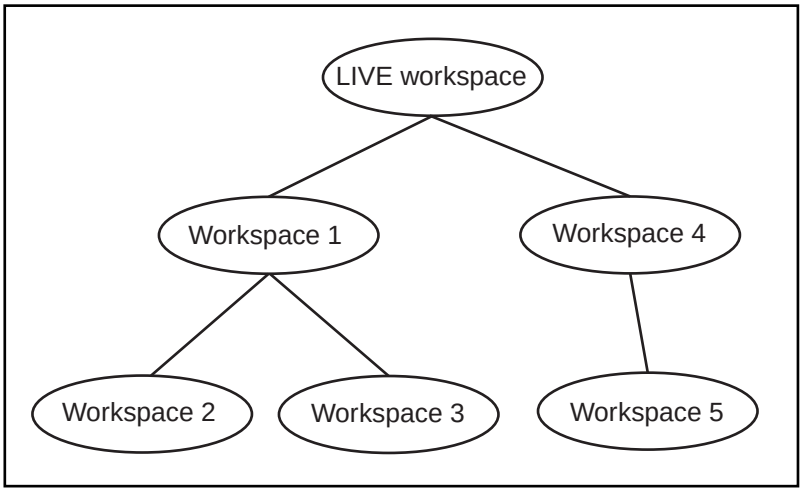
A **workspace** is a virtual environment that one or more users can share to make changes to the data in the database. Workspace management involves managing one or more workspaces that can be shared by many users.

1.2.1 Hierarchy of Workspaces

There can be one or more versions of a row in a workspace from one or more version-enabled tables. The current or active version in a workspace refers to the version number in which the changes are currently being made. All changes made in a workspace are made by conventional short transactions. There can be a hierarchy of workspaces in the database. For example, a workspace can be a parent to one or more workspaces. By default, when a workspace is created, it is created from the topmost, or `LIVE`, database workspace. (Workspace 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.) Users are included in workspaces by a GotoWorkspace operation.

Figure 1–1 shows a hierarchy of workspaces. `Workspace 1` and `Workspace 4` were formed off the `LIVE` database workspace; `Workspace 2` and `Workspace 3` were formed off `Workspace 1`, and `Workspace 5` was formed off `Workspace 4`. After `Workspace 1` was created, a user executed a GotoWorkspace operation specifying `Workspace 1`, and then executed CreateWorkspace operations to create `Workspace 2` and `Workspace 3`. A comparable sequence was followed with `Workspace 4` and `Workspace 5`.

Figure 1–1 Workspace Tree

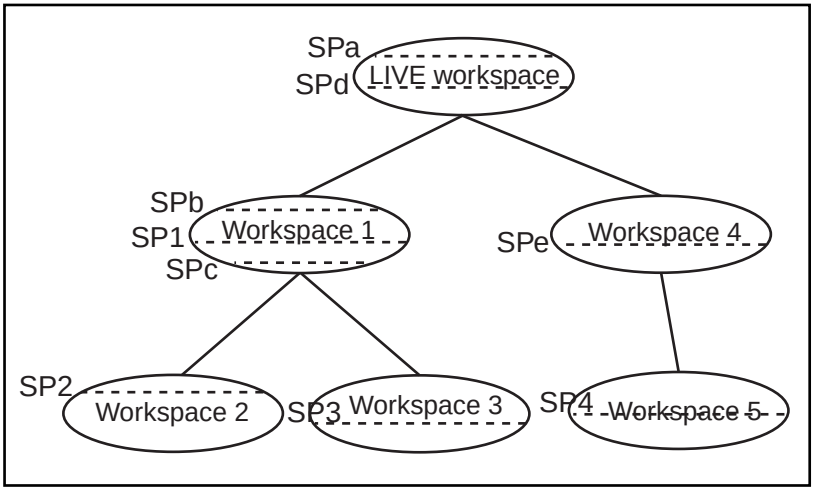


1.2.2 Using Savepoints

A **savepoint** is a point in the workspace to which operations can be rolled back. It is analogous to a firewall, in that by creating a savepoint you can prevent any damage to the "other side" of the wall (that is, operations performed in the workspace before the savepoint was created).

Explicit savepoints can thus be created and later used to effect partial rollbacks in workspaces. In Figure 1–2, SP1, SP2, SP3 and SP4 are explicit savepoints that have been 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. Thus, in Figure 1–2 two implicit savepoints (SPa and SPd) are created in the LIVE workspace corresponding to `Workspace 1` and `Workspace 4` creation; two implicit savepoints (SPb and SPc) are created in `Workspace 1` corresponding to `Workspace 2` and `Workspace 3` creation; and one implicit savepoint (SPe) is created in `Workspace 4` corresponding to `Workspace 5` creation. 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.

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 procedure.

1.2.3 Merging and Rolling Back Workspace Changes

Workspaces can be merged or rolled back.

Merging a workspace involves applying changes made in a workspace to its parent workspace, after which the workspace that had been merged ceases to exist (that is, it is removed). To merge a workspace, use the MergeWorkspace procedure.

Rolling back a workspace involves deleting either all changes made in the workspace or all changes made after a savepoint (that is, 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 have been 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 Figure 1–2 in Section 1.2.2, 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 regular 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 Section 1.2.7.)

1.2.4 Autocommitting of Workspace Manager Operations

Many Workspace Manager procedures by default commit the caller's current open regular transaction, if there is one. For example, if you compress a workspace or delete a savepoint, and if a regular transaction is open, that transaction is automatically committed when the requested Workspace Manager operation completes, unless you specify otherwise.

To override the automatic commit operation in such cases, you can specify the `auto_commit` parameter with a value of `FALSE` for many procedures. The `auto_commit` parameter has a default value of `TRUE`; however, if you specify a value of `FALSE`, any currently open regular transaction remains open, and no operations (including the Workspace Manager procedure) are committed until the transaction is committed.

Specifying `FALSE` for the `auto_commit` parameter is useful if you want to perform transactional operations unrelated to Workspace Manager before and after calling a Workspace Manager procedure, without having the Workspace Manager procedure terminate the transaction and make permanent the changes. If you later decide to roll back the transaction, all the operations (including the Workspace Manager procedure) are rolled back. Note that if you specify `FALSE` for the `auto_commit` parameter, you must remember to commit or roll back the transaction.

1.2.5 Resolving Conflicts

On a merge, the changes in the workspace are incorporated in its parent workspace. Rows that are changed in the child and parent workspace may lead to data conflicts. Conflicts are detected at merge time and presented to the user in conflict views. There is one conflict view per table, as described in Section 3.21. This view lists the primary key of the rows in conflict and also the column values of the rows in the two workspaces that form the conflict.

Conflicts have to be resolved manually by using the ResolveConflicts procedure. When there are no conflicts between the parent and child workspaces, the data in the two workspaces can be merged. Conflicts must be resolved before a MergeWorkspace or RefreshWorkspace operation can be performed.

Typically, you discover that conflicts exist when you attempt to merge or refresh a workspace and encounter an exception that refers to conflicts. The general process for resolving conflicts is as follows:

1. Examine the xxx_CONF views (described in Section 3.21) to see what conflicts exist.
2. Execute the BeginResolve procedure.
3. Execute the ResolveConflicts procedure as often as needed: once per affected combination of table and workspace. After each successful execution of ResolveConflicts, perform a standard database commit operation and execute the MergeWorkspace procedure. (However, any changes are not made permanent in the database until you execute CommitResolve, as described in the next step.)
4. After resolving all conflicts, execute one of the following procedures:
   - CommitResolve to make permanent all changes from the preceding step
   - RollbackResolve to discard all changes from the preceding step

1.2.6 Freezing and Unfreezing Workspaces

A workspace can be *frozen* or not frozen. If a workspace is frozen, no changes can be made to data in version-enabled rows, and access to the workspace is restricted.

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 Chapter 2.)

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: READ_ONLY (Also, checks to ensure that there are no sessions on savepoints other than LATEST.)
CreateSavepoint	Specified workspace: READ_ONLY
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

1.2.7 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.

1.3 Lock Management

In addition to locks provided by conventional Oracle short 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. Locking is enabled at a session level and is a session property independent of the workspace that the session is in. When locking is enabled for a session, it locks rows in all workspaces in which it participates. The two types of version locks are:

- **Exclusive locks** - The locks are very similar to short transaction locks in that once an exclusive lock has been 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.
- **Shared locks** - Once a shared lock has been 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.

Locks persist for the duration of the workspace. Merge or rollback of the workspace removes the locks.

The xxx_LOCK metadata views (described in Section 3.23) contain information about locks in each version-enabled table.

1.4 Privilege Management

Workspace Manager has 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 1–2 lists the Workspace Manager privileges.

Table 1–2 Privileges

Privilege	Description
ACCESS_WORKSPACE	Allows the user to go to a specified workspace. ACCESS_WORKSPACE or ACCESS_ANY_WORKSPACE privilege is needed for all other privileges.
ACCESS_ANY_WORKSPACE	Allows the user to go to any workspace. ACCESS_WORKSPACE or ACCESS_ANY_WORKSPACE privilege is needed for all other privileges.
CREATE_WORKSPACE	Allows the user to create a child workspace in a specified workspace.
CREATE_ANY_WORKSPACE	Allows the user to create a child workspace in any workspace.
REMOVE_WORKSPACE	Allows the user to remove a specified workspace.
REMOVE_ANY_WORKSPACE	Allows the user to remove any workspace.
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.
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.

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_ROLE role has all Workspace Manager privileges with the grant option. By default, the database administrator (DBA role) is granted the WM_ADMIN_ROLE 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 Import and Export Considerations

Standard Oracle database import and export operations can be performed on version-enabled databases; 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).
- Only database-wide import and export operations are supported for version-enabled databases. No other export modes (such as schema, table, partition, and workspace) are supported.
- For an import operation, you must specify IGNORE=Y.
- The FROMUSER and TOUSER capabilities of the Oracle9*i* Import utility are not supported with version-enabled databases.

1.6 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 DEAPRTMENT table definitions, with a foreign key constraint added after the creation (that is, the DEPT_ID in each EMPLOYEE row must match an existing DEPT_ID 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;
```

DEPARTMENT is considered the parent and EMPLOYEE is considered the child in the referential integrity relationship in this example; 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).

- Referential integrity constraints cannot be added when a table is version-enabled; they must be defined before a table is version-enabled.
- A child table in a referential integrity relationship is allowed to be version-enabled without the parent table being version-enabled.
- A version-enabled table cannot be both a child and a parent in a referential integrity relationship, unless it is a self-referential constraint. (In a self-referential constraint, the same table is both the parent and child table in a referential integrity relationship.)

Workspace Manager uses the ALL_WM_RIC_INFO and USER_WM_RIC_INFO views (described in Chapter 3) to hold information pertinent to referential integrity support.

1.7 Triggers on Version-Enabled Tables

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

- The triggers must be defined before the table is version-enabled.
- Only per-row triggers are supported. Per-statement triggers are not supported.
- Only whole-row triggers are supported. Before-update and after-update triggers for specific columns are not supported.
- Triggers on nested table columns 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.

1.8 Procedure Categories

Workspace Manager provides PL/SQL procedures that you call in order to use the product's features. The procedures are in a single PL/SQL package, but they can be logically grouped into the following categories described in this section.

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

Most functions have names starting with *Get* (such as GetConflictWorkspace and GetWorkspace).

Reference information for all interfaces is in Chapter 2.

1.8.1 Table Management

Table management procedures enable and disable workspace management on a table.

Table 1–3 shows the procedures available for table management.

Table 1–3 Table Management Procedures

Procedure	Description
EnableVersioning	Enables a table for workspace management.
DisableVersioning	Removes workspace management capabilities from a table.

1.8.2 Workspace Management

Workspace management procedures perform operations on workspaces.

Table 1–4 shows the procedures available for workspace management.

Table 1–4 Workspace Management Procedures

Procedure	Description
CreateWorkspace	Creates a new workspace in the database.
GotoWorkspace	Adds the user 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.

Table 1–4 Workspace Management Procedures (Cont.)

Procedure	Description
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 changes made in the workspace since the last merge.
RollbackToSP	Returns the workspace to a 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.
RemoveWorkspace	Rolls back and deletes a workspace.
RemoveWorkspaceTree	Deletes a workspace and its descendant workspaces.
FreezeWorkspace	Disables changes to a workspace.
UnfreezeWorkspace	Enables changes to a workspace after it has been frozen.
CompressWorkspace	Deletes explicit savepoints (all or some) in a workspace, and minimizes the Workspace Manager metadata structures.
CompressWorkspaceTree	Deletes explicit savepoints in a workspace and all its descendant workspaces, and minimizes the Workspace Manager metadata structures for the affected workspaces.
IsWorkspaceOccupied	Checks whether or not a workspace has any active sessions.
GetWorkspace	Returns the current workspace for the session.

1.8.3 Savepoint Management

Savepoint management procedures perform operations related to savepoints.

Table 1–5 shows the procedures available for savepoint management.

Table 1–5 Savepoint Management Procedures

Procedure	Description
CreateSavepoint	Creates a new savepoint in the workspace.
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.
AlterSavepoint	Modifies the description of a savepoint.
DeleteSavepoint	Deletes a savepoint.

1.8.4 Privilege Management

Privilege management procedures grant and revoke Workspace Manager privileges.

Table 1–6 shows the procedures available for privilege management.

Table 1–6 Privilege Management Procedures

Procedure	Description
GrantWorkspacePriv	Grants workspace-level privileges to users, roles, or `PUBLIC`.
RevokeWorkspacePriv	Revokes workspace-level privileges.
GrantSystemPriv	Grants privileges on all workspaces to users, roles, or `PUBLIC`.
RevokeSystemPriv	Revokes system-level privileges.
GetPrivs	Returns a comma-separated list of all privileges that the current user has for the specified workspace.

1.8.5 Lock Management

Lock management procedures control Workspace Manager locking.

Table 1–7 shows the procedures available for lock management.

Table 1–7 Lock Management Procedures

Procedure	Description
SetLockingON	Turns locking on for the current session.
SetLockingOFF	Turns locking off for the current session.

Table 1–7 Lock Management Procedures

Procedure	Description
SetWorkspaceLockModeON	Sets the default mode for the row-level locking in the workspace.
SetWorkspaceLockModeOFF	Enables access to versioned rows in the specified workspace and to corresponding rows in the parent workspace.
GetLockMode	Returns the locking mode, which determines whether or not access is enabled to versioned rows and corresponding rows in the parent workspace.
LockRows	Disables 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.

1.8.6 Conflict Management

Conflict management procedures detect and resolve conflicts between workspaces.

Table 1–8 shows the procedures available for conflict management.

Table 1–8 Conflict Management Procedures

Procedure	Description
SetConflictWorkspace	Determines the conflicts between a workspace and its parent.
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 changes.
RollbackResolve	Quits a conflict resolution session and does not save changes.
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.

1.9 Simplified Example

This section presents a simplified example of using Workspace Manager to try out some scenarios. It refers to concepts that were explained in this chapter, and it uses procedures documented in Chapter 2.

In Example 1–1, 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 one (`cola_b`) market, the company wants to do "what-if" analyses involving different managers and budget amounts.

Example 1–1 Simplified Example Using Workspace Manager

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

-- Grant regular privileges.
GRANT connect, resource to wm_developer;
GRANT 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/wm_developer

-- 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) markets in a
-- given geography (such as city or state).
-- Each row will contain budget data for a specific
-- cola. 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 (
  mkt_id NUMBER PRIMARY KEY,
  mkt_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.
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:
-- cola_a, Alvarez, 2.0
-- cola_b, Baker, 1.5
-- cola_c, Chen, 1.5
-- cola_d, Davis, 3.5
```

```
-------------------------------------------------------------------------
-- CREATE WORKSPACES --
-------------------------------------------------------------------------
-- Create workspaces for the following scenario: a major marketing focus
-- in the cola_b area. Managers and budget amounts for each
-- market 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 areas' budgets).
-- The other scenario (B_focus_2) features a manager with less expensive
-- plans (which means less money taken from other areas' 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
-- area 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 mkt_name = 'cola_b';
UPDATE cola_marketing_budget
  SET budget = 3 WHERE mkt_name = 'cola_b';
UPDATE cola_marketing_budget
  SET budget = 1.5 WHERE mkt_name = 'cola_a';
UPDATE cola_marketing_budget
  SET budget = 1 WHERE mkt_name = 'cola_c';
UPDATE cola_marketing_budget
  SET budget = 3 WHERE mkt_name = 'cola_d';
COMMIT;

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

-- Freeze this workspace. (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');

--------------------------------------------------------------------------
-- WORK 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 mkt_name = 'cola_b';
UPDATE cola_marketing_budget
  SET budget = 2 WHERE mkt_name = 'cola_b';
UPDATE cola_marketing_budget
  SET budget = 3 WHERE mkt_name = 'cola_d';
COMMIT;

-- Relevant data values now in B_focus_2 workspace::
-- cola_a, Alvarez, 2.0 (no change from LIVE)
-- cola_b, Burton, 2.0
-- cola_c, Chen, 1.5 (no change from LIVE)
-- 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 mkt_name = 'cola_b';
UPDATE cola_marketing_budget
  SET budget = 2.5 WHERE mkt_name = 'cola_d';
COMMIT;

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

-- Discard this scenario; rollback 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).
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 the other 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 removed by default after MergeWorkspace.
EXECUTE DBMS_WM.MergeWorkspace ('B_focus_2');

-- Display the current data values (which are in the LIVE database
-- workspace, which is the only workspace currently existing).
SELECT * FROM cola_marketing_budget;

------------------------------------------------------------------------------
-- DISABLE VERSIONING --
------------------------------------------------------------------------------
-- Disable versioning on the table because you are finished testing scenarios.
-- Also, users with version enabled tables cannot be dropped, in case you
-- want to drop the wm_developer user.
-- 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);
```

2

Procedures: Reference

Workspace Manager includes procedures that perform the available features of the product. This chapter provides reference information on each procedure.

> **Note:** Most Workspace Manager interfaces are procedures, but a few are functions. (A function returns a value; a procedure does not return a value.)
>
> Most functions have names starting with *Get* (such as GetConflictWorkspace and GetWorkspace).
>
> In this guide, the term *procedures* is often used to refer generally to both procedures and functions.

The procedures are presented in alphabetical order. For a brief description of procedures according to their logical groupings, see Section 1.8.

Errors (exceptions) that can occur with Workspace Manager procedures are documented in Appendix B, including the cause and suggested user action for each error.

Syntax notes:

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

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

AlterSavepoint

Purpose

Modifies the description of a savepoint.

Syntax

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

Parameters

Table 2–1 AlterSavepoint Procedure Parameters

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

Usage Notes

An exception is raised if the user is not the workspace owner or savepoint owner or does not have the WM_ADMIN_ROLE role.

Examples

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

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

AlterWorkspace

Purpose

Modifies the description of a workspace.

Syntax

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

Parameters

Table 2–2 AlterWorkspace Procedure Parameters

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

Usage Notes

An exception is raised if the user is not the workspace owner or does not have the WM_ADMIN_ROLE role.

Examples

The following example modifies the description of the NEWWORKSPACE workspace.

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

BeginResolve

Purpose

Starts a conflict resolution session.

Syntax

```
DBMS_WM.BeginResolve(
   workspace  IN VARCHAR2);
```

Parameters

Table 2–3 BeginResolve Procedure Parameters

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

Usage Notes

This procedure starts a conflict resolution session. While this procedure is executing, the workspace is frozen in `1WRITER` mode, as explained in Section 1.2.6.

After calling this procedure, you can execute the ResolveConflicts procedure as needed for various tables that have conflicts, and then call either the CommitResolve or RollbackResolve procedure. For more information about conflict resolution, see Section 1.2.5.

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

- There are one or more open regular transactions in `workspace`.
- The user executing the BeginResolve procedure does not have the privilege to access `workspace` and its parent workspace.

Examples

The following example starts a conflict resolution session in `Workspace1`.

```
EXECUTE  DBMS_WM.BeginResolve ('Workspace1');
```

CommitResolve

Purpose

Ends a conflict resolution session and saves (makes permanent) any changes in the workspace since the BeginResolve procedure was executed.

Syntax

```
DBMS_WM.CommitResolve(
   workspace  IN VARCHAR2);
```

Parameters

Table 2–4 CommitResolve Procedure Parameters

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

Usage Notes

This procedure ends the current conflict resolution session (started by the BeginResolve procedure), and saves all changes in the workspace since the start of the conflict resolution session. Contrast this procedure with RollbackResolve, which discards all changes.

For more information about conflict resolution, see Section 1.2.5.

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

- There are one or more open regular transactions in `workspace`.
- The procedure was called by a user that does not have the `WM_ADMIN_ROLE` role or that did not execute the BeginResolve procedure on `workspace`.

Examples

The following example ends the conflict resolution session in `Workspace1` and saves all changes.

```
EXECUTE  DBMS_WM.CommitResolve ('Workspace1');
```

CompressWorkspace

Purpose

Deletes explicit savepoints in a workspace and minimizes the Workspace Manager metadata structures for the workspace.

Syntax

```
DBMS_WM.CompressWorkspace(
   workspace                   IN VARCHAR2,
   compress_view_wo_overwrite  IN BOOLEAN
   [, firstSP                  IN VARCHAR2 DEFAULT NULL
   [, secondSP                 IN VARCHAR2 DEFAULT NULL] ]
   [, auto_commit              IN BOOLEAN DEFAULT TRUE]);
```

or

```
DBMS_WM.CompressWorkspace(
   workspace                   IN VARCHAR2
   [, firstSP                  IN VARCHAR2 DEFAULT NULL
   [, secondSP                 IN VARCHAR2 DEFAULT NULL] ]
   [, auto_commit              IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 2–5 CompressWorkspace Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.
compress_view_wo_overwrite	A Boolean value (TRUE or FALSE). TRUE causes history information between the affected savepoints to be deleted even if VIEW_WO_OVERWRITE was specified when versioning was enabled. FALSE causes history information (between the affected savepoints) for a table not to be deleted if VIEW_WO_OVERWRITE was specified when versioning was enabled. (If VIEW_WO_OVERWRITE was not specified for a table, history information for the table is deleted regardless of the parameter value.)

Table 2–5 CompressWorkspace Procedure Parameters (Cont.)

Parameter	Description
firstSP	First explicit savepoint. Savepoint names are case sensitive. If only workspace and firstSP are specified, all explicit savepoints between workspace creation and firstSP (but not including firstSP) are deleted. If workspace, firstSP, and secondSP are specified, all explicit savepoints from firstSP (and including firstSP if it is an explicit savepoint) to secondSP (but not including secondSP) are deleted. If only workspace is specified (no savepoints), all explicit savepoints in the workspace are deleted.
secondSP	Second explicit savepoint. All explicit savepoints from firstSP (and including firstSP if it is an explicit savepoint) to secondSP (but not including secondSP) are deleted. Savepoint names are case sensitive.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction. For more information, see Section 1.2.4.

Usage Notes

You can compress a workspace when the explicit savepoints (all or some of them) in the workspace are no longer needed. The compression operation is useful for the following reasons:

- You can reuse savepoint names after they are deleted. (You cannot create a savepoint that has the same name as an existing savepoint.)
- Runtime performance for Workspace Manager operations is improved.
- Less disk storage is used for Workspace Manager structures.

While this procedure is executing, the current workspace is frozen in READ_ONLY mode, as explained in Section 1.2.6.

A workspace cannot be compressed if there are any sessions with an open regular transaction, or if any user has executed a GotoDate operation or a GotoSavepoint operation specifying a savepoint in the workspace.

For information about VIEW_WO_OVERWRITE and other history options, see the information about the EnableVersioning procedure.

An exception is raised if the user does not have the privilege to access and merge changes in workspace.

To compress a workspace and all its descendant workspaces, use the CompressWorkspaceTree procedure.

Examples

The following example compresses NEWWORKSPACE.

```
EXECUTE DBMS_WM.CompressWorkspace ('NEWWORKSPACE');
```

The following example compresses NEWWORKSPACE, deleting all explicit savepoints between the creation of the workspace and the savepoint SP1.

```
EXECUTE DBMS_WM.CompressWorkspace ('NEWWORKSPACE', 'SP1');
```

The following example compresses NEWWORKSPACE, deleting the explicit savepoint SP1 and all explicit savepoints up to but not including SP2.

```
EXECUTE DBMS_WM.CompressWorkspace ('NEWWORKSPACE', 'SP1', 'SP2');
```

The following example compresses B_focus_1, accepts the default values for the firstSP and secondSP parameters (that is, deletes all explicit savepoints), and specifies FALSE for the auto_commit parameter.

```
EXECUTE DBMS_WM.CompressWorkspace ('B_focus_1', NULL, NULL, FALSE);
```

CompressWorkspaceTree

Purpose

Deletes explicit 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.

Syntax

```
DBMS_WM.CompressWorkspaceTree(
   workspace                      IN VARCHAR2
   [, compress_view_wo_overwrite  IN BOOLEAN DEFAULT FALSE]
   [, auto_commit                 IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 2–6 CompressWorkspaceTree Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.
compress_view_wo_overwrite	A Boolean value (TRUE or FALSE). TRUE causes history information to be deleted even if VIEW_WO_OVERWRITE was specified when versioning was enabled. FALSE (the default) causes history information for a table not to be deleted if VIEW_WO_OVERWRITE was specified when versioning was enabled. (If VIEW_WO_OVERWRITE was not specified for a table, history information for the table is deleted regardless of the parameter value.)
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction. For more information, see Section 1.2.4.

Usage Notes

You can compress a workspace and all its descendant workspaces when the explicit savepoints in the affected workspaces are no longer needed (for example, if you will not need to go to or roll back to any of these savepoints). For example, in the hierarchy shown in Figure 1–1 in Section 1.2.1, a CompressWorkspaceTree operation specifying Workspace 1 compresses Workspace 1, Workspace 2, and Workspace 3. (For an explanation of database workspace hierarchy, see Section 1.2.1.)

The compression operation is useful for the following reasons:

- You can reuse savepoint names after they are deleted. (You cannot create a savepoint that has the same name as an existing savepoint.)
- Runtime performance for Workspace Manager operations is improved.
- Less disk storage is used for Workspace Manager structures.

A workspace cannot be compressed if there are any sessions with an open regular transaction, or if any user has executed a GotoDate operation or a GotoSavepoint operation specifying a savepoint in the workspace.

An exception is raised if the user does not have the privilege to access and merge changes in `workspace`.

If the CompressWorkspaceTree operation fails in any affected workspace, the entire operation is rolled back, and no workspaces are compressed.

To compress a single workspace (deleting all explicit savepoints or just some of them), use the CompressWorkspace procedure.

Examples

The following example compresses `NEWWORKSPACE` and all its descendant workspaces.

```
EXECUTE DBMS_WM.CompressWorkspaceTree ('NEWWORKSPACE');
```

The following example compresses `NEWWORKSPACE` and all its descendant workspaces, accepts the default value for the `compress_view_wo_overwrite` parameter, and specifies `FALSE` for the `auto_commit` parameter.

```
EXECUTE DBMS_WM.CompressWorkspaceTree ('B_focus_1', NULL, FALSE);
```

CopyForUpdate

Purpose

Allows LOB columns (BLOB, CLOB, or NCLOB) in version-enabled tables to be modified. Use this procedure only if a version-enabled table has any LOB columns.

Syntax

```
DBMS_WM.CopyForUpdate(
   table_name       IN VARCHAR2,
   [, where_clause  IN VARCHAR2 DEFAULT '']);
```

Parameters

Table 2–7 CopyForUpdate Procedure Parameters

Parameter	Description
table_name	Name of the table containing one or more LOB columns. The name is not case sensitive.
where_clause	The WHERE clause (excluding the WHERE keyword) identifying the rows affected. Example: 'department_id = 20' The WHERE clause cannot contain a subquery. If where_clause is not specified, all rows in table_name are affected.

Usage Notes

This procedure is intended for use only with version-enabled tables containing one or more large object (LOB) columns. The CopyForUpdate procedure must be used because updates performed using the DBMS_LOB package do not fire INSTEAD OF triggers on the versioning views. Workspace Manager creates INSTEAD OF triggers on the versioning views to implement the copy-on-write semantics. (For non-LOB columns, you can directly perform the update operation, and the triggers work.)

Examples

The following example updates the SOURCE_CLOB column of TABLE1 for the document with DOC_ID = 1.

```
Declare
  clob_var
```

```
Begin
   /* This procedure copies the LOB columns if necessary, that is,
      if the row with doc_id = 1 has not been versioned in the
      current version */
   vm.copyForUpdate('table1', 'doc_id = 1');

   select source_clob into clob_var
   from   table1
   where  doc_id = 1 for update;

   dbms_lob.write(clob_var,<amount>, <offset>, buff);

End;
```

CreateSavepoint

Purpose

Creates a savepoint for the current version.

Syntax

```
DBMS_WM.CreateSavepoint(
   workspace       IN VARCHAR2,
   savepoint_name  IN VARCHAR2
   [, description  IN VARCHAR2 DEFAULT NULL]
   [, auto_commit  IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 2–8 CreateSavepoint Procedure Parameters

Parameter	Description
workspace	Name of the workspace in which to create the savepoint. The name is case sensitive.
savepoint_name	Name of the savepoint to be created. The name is case sensitive.
description	Description of the savepoint to be created.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction. For more information, see Section 1.2.4.

Usage Notes

There are no explicit privileges associated with savepoints; any user who can access a workspace can create a savepoint in the workspace.

This procedure can be performed while there are users in the workspace; a quiet point is not required.

While this procedure is executing, the current workspace is frozen in READ_ONLY mode, as explained in Section 1.2.6.

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

- The user is not in the latest version in the workspace (for example, if the user has called the GotoDate procedure).
- workspace does not exist.
- savepoint_name already exists.
- The user does not have the privilege to go to the specified workspace.

Examples

The following example creates a savepoint named Savepoint1 in the NEWWORKSPACE workspace.

```
EXECUTE DBMS_WM.CreateSavepoint ('NEWWORKSPACE', 'Savepoint1');
```

CreateWorkspace

Purpose

Creates a new workspace in the database.

Syntax

```
DBMS_WM.CreateWorkspace(
   workspace       IN VARCHAR2
   [, description  IN VARCHAR2 DEFAULT NULL]
   [, auto_commit  IN BOOLEAN DEFAULT TRUE]);
```

or

```
DBMS_WM.CreateWorkspace(
   workspace       IN VARCHAR2,
   isrefreshed     IN BOOLEAN
   [, description  IN VARCHAR2 DEFAULT NULL]
   [, auto_commit  IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 2–9 CreateWorkspace Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive, and it must be unique (no other workspace of the same name).
isrefreshed	A Boolean value (TRUE or FALSE). TRUE causes the workspace to be continually refreshed. In a **continually refreshed workspace**, changes made in the parent workspace are automatically applied to the workspace after a merge or rollback operation in the parent workspace. That is, you do not need to call the RefreshWorkspace procedure to apply the changes. A continually refreshed workspace must be created as a child of the LIVE workspace. FALSE causes the workspace not to be continually refreshed. To refresh the workspace, you must call the RefreshWorkspace procedure. If you use the syntax without the isrefreshed parameter, the workspace is not continually refreshed.
description	Description of the workspace.

Table 2–9 CreateWorkspace Procedure Parameters (Cont.)

Parameter	Description
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction. For more information, see Section 1.2.4.

Usage Notes

The new workspace is a child of the current workspace. If the session has not explicitly entered a workspace, it is in the LIVE database workspace, and the new workspace is a child of the LIVE workspace. For an explanation of database workspace hierarchy, see Section 1.2.1.

An implicit savepoint is created in the current version of the current workspace. (The current version does not have to be the latest version in the current workspace.) For an explanation of savepoints (explicit and implicit), see Section 1.2.2.

While this procedure is executing, the current workspace is frozen in READ_ONLY mode, as explained in Section 1.2.6.

This procedure does not implicitly go to the workspace created. To go to the workspace, use the GotoWorkspace procedure.

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

- workspace already exists.
- The user does not have the privilege to create a workspace.

Examples

The following example creates a workspace named NEWWORKSPACE in the database.

```
EXECUTE DBMS_WM.CreateWorkspace ('NEWWORKSPACE');
```

DeleteSavepoint

Purpose

Deletes a savepoint.

Syntax

```
DBMS_WM.DeleteSavepoint(
   workspace                      IN VARCHAR2,
   savepoint_name                 IN VARCHAR2)
   [, compress_view_wo_overwrite  IN BOOLEAN DEFAULT FALSE]
   [, auto_commit                 IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 2–10 DeleteSavepoint Procedure Parameters

Parameter	Description
workspace	Name of the workspace in which the savepoint was created. The name is case sensitive.
savepoint_name	Name of the savepoint to be deleted. The name is case sensitive.
compress_view_wo_overwrite	A Boolean value (TRUE or FALSE). TRUE causes history information to be deleted even if VIEW_WO_OVERWRITE was specified when versioning was enabled. FALSE (the default) causes history information for a table not to be deleted if VIEW_WO_OVERWRITE was specified when versioning was enabled. (If VIEW_WO_OVERWRITE was not specified for a table, history information for the table is deleted regardless of the parameter value.)
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction. For more information, see Section 1.2.4.

Usage Notes

You can delete a savepoint when it is no longer needed (for example, you will not need to go to it or roll back to it).

Deleting a savepoint is useful for the following reasons:

- You can reuse a savepoint name after it is deleted. (You cannot create a savepoint that has the same name as an existing savepoint.)
- Runtime performance for Workspace Manager operations is improved.
- Less disk storage is used for Workspace Manager structures.

To delete a savepoint, you must have the `WM_ADMIN_ROLE` role or be the owner of the workspace or the savepoint.

This procedure cannot be executed if there are any sessions with an open regular transaction, or if any user has executed a GotoDate operation or a GotoSavepoint operation specifying a savepoint in the workspace.

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

- The user is not in the latest version in the workspace (for example, if the user has called GotoDate).
- `workspace` does not exist.
- `savepoint_name` does not exist.
- The user does not have the privilege to go to the specified workspace.

Examples

The following example deletes a savepoint named `Savepoint1` in the `NEWWORKSPACE` workspace.

```
EXECUTE DBMS_WM.DeleteSavepoint ('NEWWORKSPACE', 'Savepoint1');
```

DisableVersioning

Purpose

Deletes all support structures that were created to enable the table to support versioned rows.

Syntax

```
DBMS_WM.DisableVersioning(
   table_name  IN VARCHAR2
   [, force    IN BOOLEAN DEFAULT FALSE]);
```

Parameters

Table 2–11 DisableVersioning Procedure Parameters

Parameter	Description
table_name	Name of the table. The name is not case sensitive.
force	A Boolean value (TRUE or FALSE). TRUE forces all data in workspaces other than LIVE to be discarded before versioning is disabled. FALSE (the default) prevents versioning from being disabled if table_name was modified in any workspace other than LIVE and if the workspace that modified table_name still exists.

Usage Notes

This procedure is used to reverse the effect of the EnableVersioning procedure. It deletes the Workspace Manager infrastructure (support structures) for versioning of rows, but does not affect any user data in the LIVE workspace. The workspace hierarchy and any savepoints still exist, but all rows are the same as in the LIVE workspace. (If there are multiple versions in the LIVE workspace of a row in the table for which versioning is disabled, only the most recent version of the row is kept.)

The DisableVersioning operation fails if the force value is FALSE and any of the following apply:

- The table is being modified by any user in any workspace other than the LIVE workspace.

- There are versioned rows of the table in any workspace other than the LIVE workspace.

Only the owner of a table or a user with the WM_ADMIN_ROLE role can disable versioning on the table.

Tables that are version-enabled and users that own version-enabled tables cannot be deleted. You must first disable versioning on the relevant table or tables.

An exception is raised if the table is not version-enabled.

Examples

The following example disables the EMPLOYEE table for versioning.

```
EXECUTE DBMS_WM.DisableVersioning ('employee');
```

EnableVersioning

Purpose

Creates the necessary structures to enable the table to support multiple versions of rows.

Syntax

```
DBMS_WM.EnableVersioning(
   table_name  IN VARCHAR2
   [, hist     IN VARCHAR2 DEFAULT 'NONE']);
```

Parameters

Table 2–12 EnableVersioning Procedure Parameters

Parameter	Description
table_name	Name of the table. The length of a table name must not exceed 25 characters. The name is not case sensitive.
hist	History option, for tracking modifications to table_name. Must be one of the following values: NONE: No modifications to the table are tracked. (This is the default.) VIEW_W_OVERWRITE: The *with* overwrite (*W*_OVERWRITE) option. A view named <table_name>_HIST (described in Section 3.24) is created to contain history information, but it will show only the most recent modifications to the same version of the table. A history of modifications to the version is not maintained; that is, subsequent changes to a row in the same version overwrite earlier changes. (The CREATETIME column of the <table_name>_HIST view contains only the time of the most recent update.) VIEW_WO_OVERWRITE: The *without* overwrite (*WO*_OVERWRITE) option. A view named <table_name>_HIST (described in Section 3.24) is created to contain history information, and it will show all modifications to the same version of the table. A history of modifications to the version is maintained; that is, subsequent changes to a row in the same version do not overwrite earlier changes.

Usage Notes

The table that is being version-enabled must have a primary key defined.

Only the owner of a table can enable versioning on the table.

Tables that are version-enabled and users that own version-enabled tables cannot be deleted. You must first disable versioning on the relevant table or tables.

Tables owned by SYS cannot be version-enabled.

An exception is raised if the table is already version-enabled.

If the table is version-enabled with the VIEW_WO_OVERWRITE *hist* option specified, this option can later be disabled and re-enabled by calling the SetWoOverwriteOFF and SetWoOverwriteON procedures. However, the VIEW_WO_OVERWRITE *hist* option can be overridden when a workspace is compressed by specifying the compress_view_wo_overwrite parameter as TRUE with the CompressWorkspace or CompressWorkspaceTree procedure.

The history option enables you to log and audit modifications.

The history option affects the behavior of the GotoDate procedure. See the Usage Notes for that procedure.

Current notes and restrictions include the following:

- If you have referential integrity constraints on version-enabled tables, note the considerations and restrictions in Section 1.6.
- If you have triggers defined on version-enabled tables, note the considerations and restrictions in Section 1.7.
- Constraints and privileges defined on the table are carried over to the version-enabled table.
- DDL operations are not allowed on version-enabled tables.
- Index-organized tables cannot be version-enabled.
- Object tables cannot be version-enabled.
- A table with one or more columns of LONG data type cannot be version-enabled.

Examples

The following example enables versioning on the EMPLOYEE table.

```
EXECUTE DBMS_WM.EnableVersioning('employee');
```

FreezeWorkspace

Purpose

Disables changes in a workspace and prevents subsequent sessions from entering the workspace.

Syntax

```
DBMS_WM.FreezeWorkspace(
   workspace        IN VARCHAR2
   [, freezemode    IN VARCHAR2 DEFAULT 'NO_ACCESS']
   [, freezewriter  IN VARCHAR2 DEFAULT NULL]
   [, force         IN BOOLEAN DEFAULT FALSE]);
```

Parameters

Table 2–13 FreezeWorkspace Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.
freezemode	Mode for the frozen workspace. Must be one of the following values: NO_ACCESS: No sessions are allowed in the workspace. (This is the default.) READ_ONLY: Sessions are allowed in the workspace, but no write operations (insert, update, delete) are allowed. 1WRITER: Sessions are allowed in the workspace, but only one user (see the freezewriter parameter) is allowed to perform write operations (insert, update, delete). WM_ONLY: Only Workspace Manager operations are permitted. No sessions can directly modify data values or perform queries involving table data; however, child workspaces can be merged into the workspace, and savepoints can be created in the workspace.
freezewriter	The user that is allowed to make changes in the workspace. Can be specified only if freezemode is 1WRITER. The default is USER (the current user).

Table 2–13 FreezeWorkspace Procedure Parameters (Cont.)

Parameter	Description
force	A Boolean value (TRUE or FALSE). TRUE forces the workspace to be frozen even if it is already frozen. For example, this value lets you freeze the workspace with a different freezemode parameter value without having first to call the UnfreezeWorkspace procedure. FALSE (the default) prevents the workspace from being frozen if it is already frozen.

Usage Notes

The operation fails if any sessions are active in workspace (unless force is TRUE) and freezemode is NO_ACCESS.

If freezemode is READ_ONLY or 1WRITER, the workspace cannot be frozen if there is an active regular transaction.

Only the owner of the workspace or a user with WM_ADMIN_ROLE can freeze a workspace. There are no specific privileges associated with freezing a workspace.

The LIVE workspace can be frozen only if freezemode is READ_ONLY or 1WRITER.

To reverse the effect of FreezeWorkspace, use the UnfreezeWorkspace procedure.

Examples

The following example freezes the NEWWORKSPACE workspace.

```
EXECUTE DBMS_WM.FreezeWorkspace ('NEWWORKSPACE');
```

GetConflictWorkspace

Purpose

Returns the name of the workspace on which the session has performed the SetConflictWorkspace procedure.

Format

```
DBMS_WM.GetConflictWorkspace() RETURN VARCHAR2;
```

Parameters

None.

Usage Notes

If the SetConflictWorkspace procedure has not been executed, the name of the current workspace is returned.

Examples

The following example displays the name of the workspace on which the session has performed the SetConflictWorkspace procedure.

```
SELECT DBMS_WM.GetConflictWorkspace FROM DUAL;

GETCONFLICTWORKSPACE
--------------------------------------------------------------------------------
B_focus_2
```

GetDiffVersions

Purpose

Returns the names of the (workspace, savepoint) pairs on which the session has performed the SetDiffVersions operation.

Format

```
DBMS_WM.GetDiffVersions() RETURN VARCHAR2;
```

Parameters

None.

Usage Notes

The returned string is in the format `'(WS1,SP1), (WS2,SP2)'`. This format, including the parentheses, is intended to help you if you later want to use parts of the returned string in a call to the SetDiffVersions procedure.

Examples

The following example displays the names of the (workspace, savepoint) pairs on which the session has performed the SetDiffVersions operation.

```
SELECT DBMS_WM.GetDiffVersions FROM DUAL;

GETDIFFVERSIONS
--------------------------------------------------------------------------------
(B_focus_1, LATEST), (B_focus_2, LATEST)
```

GetLockMode

Purpose

Returns the locking mode, which determines whether or not access is enabled to versioned rows and corresponding rows in the previous version.

Format

```
DBMS_WM.GetLockMode() RETURN VARCHAR2;
```

Parameters

None.

Usage Notes

This function returns E, S, C, or NULL.

- For explanations of E (exclusive), S (shared), and C (carry-forward), see the description of the lockmode parameter of the SetLockingON procedure.
- NULL indicates that locking is not in effect. (Calling the SetLockingOFF procedure results in this setting.)

For an explanation of Workspace Manager locking, see Section 1.3. See also the descriptions of the SetLockingON and SetLockingOFF procedures.

Examples

The following example displays the locking mode in effect for the session.

```
SELECT DBMS_WM.GetLockMode FROM DUAL;

GETLOCKMODE
--------------------------------------------------------------------------------
C
```

GetMultiWorkspaces

Purpose

Returns the names of workspaces visible in the multiworkspace views for version-enabled tables.

Format

```
DBMS_WM.GetMultiWorkspaces() RETURN VARCHAR2;
```

Parameters

None.

Usage Notes

This procedure returns the names of workspaces visible in the multiworkspace views, which are described in Section 3.25.

If no workspaces are visible in the multiworkspace views, `NULL` is returned. If more than one workspace name is returned, names are separated by a comma (for example: `workspace1,workspace2,workspace3`).

To make a workspace visible in the multiworkspace views, use the SetMultiWorkspaces procedure.

Examples

The following example displays the names of workspaces visible in the multiworkspace views.

```
SELECT DBMS_WM.GetMultiWorkspaces FROM DUAL;
```

GetOpContext

Purpose

Returns the context of the current operation.

Format

```
DBMS_WM.GetOpContext() RETURN VARCHAR2;
```

Parameters

None.

Usage Notes

This function returns one of the following values:

- DML: The current operation is driven by data manipulation language (DML) initiated by the user.
- MERGE_REMOVE: The current operation was initiated by a MergeWorkspace procedure call with the remove_workspace parameter set to TRUE or a MergeTable procedure call with the remove_data parameter set to TRUE.
- MERGE_NOREMOVE: The current operation was initiated by a MergeWorkspace procedure call with the remove_workspace parameter set to FALSE or a MergeTable procedure call with the remove_data parameter set to FALSE.

Examples

The following example displays the context of the current operation.

```
SELECT DBMS_WM.GetOpContext FROM DUAL;

GETOPCONTEXT
--------------------------------------------------------------------------------
DML
```

GetPrivs

Purpose

Returns a comma-separated list of all privileges that the current user has for the specified workspace.

Format

```
DBMS_WM.GetPrivs(
   workspace  VARCHAR2) RETURN VARCHAR2;
```

Parameters

Table 2–14 GetPrivs Function Parameters

Parameter	Description
workspace	Name of the workspace for which to return the list of privileges. The name is case sensitive.

Usage

For information about Workspace Manager privileges, see Section 1.4.

Examples

The following example displays the privileges that the current user has for the B_focus_2 workspace.

```
SELECT DBMS_WM.GetPrivs ('B_focus_2') FROM DUAL;

DBMS_WM.GETPRIVS('B_FOCUS_2')
--------------------------------------------------------------------------------
ACCESS,MERGE,CREATE,REMOVE,ROLLBACK
```

GetWorkspace

Purpose

Returns the current workspace for the session.

Format

```
DBMS_WM.GetWorkspace() RETURN VARCHAR2;
```

Parameters

None.

Usage Notes

None.

Examples

The following example displays the current workspace for the session.

```
SELECT DBMS_WM.GetWorkspace FROM DUAL;

GETWORKSPACE
--------------------------------------------------------------------------------
B_focus_2
```

GotoDate

Purpose

Goes to a point at or near the specified date and time in the current workspace.

Syntax

```
DBMS_WM.GotoDate(
   in_date  IN DATE);
```

Parameters

Table 2–15 GotoDate Procedure Parameters

Parameter	Description
in_date	Date and time for the read-only view of the workspace. (See the Usage Notes for details.)

Usage Notes

You are presented a read-only view of the current workspace at or near the specified date and time. The exact time point depends on the history option for tracking modifications, as set by the EnableVersioning procedure or modified by the SetWoOverwriteOFF or SetWoOverwriteON procedure:

- NONE: The read-only view reflects the first savepoint after in_date.
- VIEW_W_OVERWRITE: The read-only view contents can vary depending on when updates were performed and if or when savepoints were created. The view reflects the data values in effect at in_date except for rows that have been modified *both* (1) between in_date and the most recent savepoint before in_date *and* (2) between in_date and the next savepoint after in_date; for these rows the view reflects the date in effect at the most recent savepoint *before* in_date. Therefore, be careful if you use this procedure when the VIEW_W_OVERWRITE option is enabled.
- VIEW_WO_OVERWRITE: The read-only view reflects the data values in effect at in_date.

For an explanation of the history options, see the description of the hist parameter for the EnableVersioning procedure. The following example scenario shows the

effects of the VIEW_W_OVERWRITE and VIEW_WO_OVERWRITE settings. Assume the following sequence of events:

1. The MANAGER_NAME value in a row is Adams.
2. Savepoint SP1 is created.
3. The MANAGER_NAME value is changed to Baxter.
4. The time point that will be specified as in_date (in step 7) occurs.
5. The MANAGER_NAME value is changed to Chang. (Thus, the value has been changed both before and after in_date since the first savepoint and before the second savepoint.)
6. Savepoint SP2 is created.
7. A GotoDate operation is executed, specifying the time point in step 4 as in_date.

In the preceding scenario:

- If the history option in effect is VIEW_W_OVERWRITE, the MANAGER_NAME value after step 7 is Adams.
- If the history option in effect is VIEW_WO_OVERWRITE, the MANAGER_NAME value after step 7 is Baxter.

The GotoDate procedure should be executed while users exist in the workspace. There are no explicit privileges associated with this procedure.

Examples

The following example goes to a point at or near midnight at the start of 30-Jun-2000, depending on the history option currently in effect.

```
EXECUTE DBMS_WM.GotoDate ('30-JUN-00');
```

GotoSavepoint

Purpose

Goes to the specified savepoint in the current workspace.

Syntax

```
DBMS_WM.GotoSavePoint(
   [savepoint_name  IN VARCHAR2 DEFAULT 'LATEST']);
```

Parameters

Table 2–16 GotoSavepoint Procedure Parameters

Parameter	Description
savepoint_name	Name of the savepoint. The name is case sensitive. If savepoint_name is not specified, the default is LATEST.

Usage Notes

You are presented a read-only view of the workspace at the time of savepoint creation. This procedure is useful for examining the workspace from different savepoints before performing a rollback to a specific savepoint by calling the RollbackToSP procedure to delete all rows from that savepoint forward.

This operation can be executed while users exist in the workspace. There are no explicit privileges associated with this operation.

If you do not want to roll back to the savepoint, you can call the GotoSavepoint procedure with a null parameter to go to the currently active version in the workspace. (This achieves the same result as calling the GotoWorkspace procedure and specifying the workspace.)

For more information about savepoints, including the LATEST savepoint, see Section 1.2.2.

Examples

The following example goes to the savepoint named Savepoint1.

```
EXECUTE DBMS_WM.GotoSavepoint ('Savepoint1');
```

GotoWorkspace

Purpose

Moves the current session to the specified workspace.

Syntax

```
DBMS_WM.GotoWorkspace(
   workspace  IN VARCHAR2);
```

Parameters

Table 2–17 GotoWorkspace Procedure Parameters

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

Usage Notes

After a user goes to a workspace, modifications to data can be made there.

To go to the live database, specify workspace as LIVE. Because many operations are prohibited when any users (including you) are in the workspace, it is often convenient to go to the LIVE workspace before performing operations on created workspaces.

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

- workspace does not exist.
- The user does not have ACCESS_WORKSPACE privilege for workspace.
- workspace has been frozen to new users (see the FreezeWorkspace procedure).

Examples

The following example includes the user in the NEWWORKSPACE workspace. The user will begin to work in the latest version in that workspace.

```
EXECUTE DBMS_WM.GotoWorkspace ('NEWWORKSPACE');
```

The following example includes the user in the LIVE database workspace. By default, when users connect to a database, they are placed in this workspace.

```
EXECUTE DBMS_WM.GotoWorkspace ('LIVE');
```

GrantSystemPriv

Purpose

Grants system-level privileges (not restricted to a particular workspace) to users and roles. The `grant_option` parameter enables the grantee to grant the specified privileges to other users and roles.

Syntax

```
DBMS_WM.GrantSystemPriv(
   priv_types       IN VARCHAR2,
   grantee          IN VARCHAR2
   [, grant_option  IN VARCHAR2 DEFAULT 'NO']
   [, auto_commit   IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 2–18 GrantSystemPriv Procedure Parameters

Parameter	Description
priv_types	A string of one or more keywords representing privileges. (Section 1.4 discusses Workspace Manager privileges.) Use commas to separate privilege keywords. The available keywords are ACCESS_ANY_WORKSPACE, MERGE_ANY_WORKSPACE, CREATE_ANY_WORKSPACE, REMOVE_ANY_WORKSPACE, and ROLLBACK_ANY_WORKSPACE.
grantee	Name of the user (can be the PUBLIC user group) or role to which to grant priv_types.
grant_option	Specify YES to enable the grant option for grantee, or NO (the default) to disable the grant option for grantee. The grant option allows grantee to grant the privileges specified in priv_types to other users and roles.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction. For more information, see Section 1.2.4.

Usage Notes

Contrast this procedure with GrantWorkspacePriv, which grants workspace-level Workspace Manager privileges with keywords that do not contain ANY and which has a workspace parameter.

If a user gets a privilege from more than one source and if any of those sources has the grant option for that privilege, the user has the grant option for the privilege. For example, assume that user SCOTT has been granted the ACCESS_ANY_WORKSPACE privilege with grant_option as NO, but that the PUBLIC user group has been granted the ACCESS_ANY_WORKSPACE privilege with grant_option as YES. Because user SCOTT is a member of PUBLIC, user SCOTT has the ACCESS_ANY_WORKSPACE privilege with the grant option.

The WM_ADMIN_ROLE role has all Workspace Manager privileges with the grant option. The WM_ADMIN_ROLE role is automatically given to the DBA role.

The ACCESS_WORKSPACE or ACCESS_ANY_WORKSPACE privilege is needed for all other Workspace Manager privileges.

To revoke system-level privileges, use the RevokeSystemPriv procedure.

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

- grantee is not a valid user or role in the database.
- You do not have the privilege to grant priv_types.

Examples

The following example enables user Smith to access any workspace in the database, but does not allow Smith to grant the ACCESS_ANY_WORKSPACE privilege to other users.

```
EXECUTE DBMS_WM.GrantSystemPriv ('ACCESS_ANY_WORKSPACE', 'Smith', 'NO');
```

GrantWorkspacePriv

Purpose

Grants workspace-level privileges to users and roles. The `grant_option` parameter enables the grantee to grant the specified privileges to other users and roles.

Syntax

```
DBMS_WM.GrantWorkspacePriv(
   priv_types       IN VARCHAR2,
   workspace        IN VARCHAR2,
   grantee          IN VARCHAR2
   [, grant_option  IN VARCHAR2 DEFAULT 'NO']
   [, auto_commit   IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 2–19 GrantWorkspacePriv Procedure Parameters

Parameter	Description
priv_types	A string of one or more keywords representing privileges. (Section 1.4 discusses Workspace Manager privileges.) Use commas to separate privilege keywords. The available keywords are ACCESS_WORKSPACE, MERGE_WORKSPACE, CREATE_WORKSPACE, REMOVE_WORKSPACE, and ROLLBACK_WORKSPACE.
workspace	Name of the workspace. The name is case sensitive.
grantee	Name of the user (can be the PUBLIC user group) or role to which to grant priv_types.
grant_option	Specify YES to enable the grant option for grantee, or NO (the default) to disable the grant option for grantee. The grant option allows grantee to grant the privileges specified in priv_types on the workspace specified in workspace to other users and roles.

Table 2–19 GrantWorkspacePriv Procedure Parameters (Cont.)

Parameter	Description
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction. For more information, see Section 1.2.4.

Usage Notes

Contrast this procedure with GrantSystemPriv, which grants system-level Workspace Manager privileges with keywords in the form *xxx*_ANY_WORKSPACE (ACCESS_ANY_WORKSPACE, MERGE_ANY_WORKSPACE, and so on).

If a user gets a privilege from more than one source and if any of those sources has the grant option for that privilege, the user has the grant option for the privilege. For example, assume that user SCOTT has been granted the ACCESS_WORKSPACE privilege with grant_option as NO, but that the PUBLIC user group has been granted the ACCESS_WORKSPACE privilege with grant_option as YES. Because user SCOTT is a member of PUBLIC, user SCOTT has the ACCESS_WORKSPACE privilege with the grant option.

The WM_ADMIN_ROLE role has all Workspace Manager privileges with the grant option. The WM_ADMIN_ROLE role is automatically given to the DBA role.

The ACCESS_WORKSPACE or ACCESS_ANY_WORKSPACE privilege is needed for all other Workspace Manager privileges.

To revoke workspace-level privileges, use the RevokeWorkspacePriv procedure.

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

- grantee is not a valid user or role in the database.
- You do not have the privilege to grant priv_types.

Examples

The following example enables user Smith to access the NEWWORKSPACE workspace and merge changes in that workspace, and allows Smith to grant the two specified privileges on NEWWORKSPACE to other users.

```
DBMS_WM.GrantWorkspacePriv ('ACCESS_WORKSPACE, MERGE_WORKSPACE', 'NEWWORKSPACE',
'Smith', 'YES');
```

IsWorkspaceOccupied

Purpose

Checks whether or not a workspace has any active sessions.

Syntax

```
DBMS_WM.IsWorkspaceOccupied(
   workspace  IN VARCHAR2) RETURN VARCHAR2;
```

Parameters

Table 2–20 IsWorkspaceOccupied Function Parameters

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

Usage Notes

This function returns YES if the workspace has any active sessions, and it returns NO if the workspace has no active sessions.

An exception is raised if the LIVE workspace is specified or if the user does not have the privilege to access the workspace.

Examples

The following example checks if any sessions are active in the B_focus_2 workspace.

```
SELECT DBMS_WM.IsWorkspaceOccupied('B_focus_2') FROM DUAL;

DBMS_WM.ISWORKSPACEOCCUPIED('B_FOCUS_2')
--------------------------------------------------------------------------------
YES
```

LockRows

Purpose

Controls access to versioned rows in a specified table and to corresponding rows in the parent workspace.

Syntax

```
DBMS_WM.LockRows(
   workspace        IN VARCHAR2,
   table_name       IN VARCHAR2
   [, where_clause  IN VARCHAR2 DEFAULT '']
   [, lock_mode     IN VARCHAR2 DEFAULT 'E']);
```

Parameters

Table 2–21 LockRows Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The latest versions of rows visible from the workspace are locked. If a row has not been modified in this workspace, the locked version could be in an ancestor workspace. The name is case sensitive.
table_name	Name of the table in which rows are to be locked. The name is not case sensitive.
where_clause	The WHERE clause (excluding the WHERE keyword) identifying the rows to be locked. Example: 'department_id = 20' Only primary key columns can be specified in the WHERE clause. The WHERE clause cannot contain a subquery. If where_clause is not specified, all rows in table_name are locked.
lock_mode	Mode with which to set the locks: E (exclusive) or S (shared). The default is E.

Usage Notes

This procedure affects Workspace Manager locking, which occurs in addition to any standard Oracle server locking. For an explanation of Workspace Manager locking, see Section 1.3.

This procedure does not affect whether Workspace Manager locking is set on or off (determined by the SetLockingON and SetLockingOFF procedures).

To unlock rows, use the UnlockRows procedure.

Examples

The following example locks rows in the EMPLOYEES table where last_name = 'Smith' in the NEWWORKSPACE workspace.

```
EXECUTE DBMS_WM.LockRows ('NEWWORKSPACE', 'employees', 'last_name = ''Smith''');
```

MergeTable

Purpose

Applies changes to a table (all rows or as specified in the WHERE clause) in a workspace to its parent workspace.

Syntax

```
DBMS_WM.MergeTable(
   workspace            IN VARCHAR2,
   table_id             IN VARCHAR2
   [, where_clause      IN VARCHAR2 DEFAULT '']
   [, create_savepoint  IN BOOLEAN DEFAULT FALSE]
   [, remove_data       IN BOOLEAN DEFAULT FALSE]
   [, auto_commit       IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 2–22 MergeTable Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.
table_id	Name of the table containing rows to be merged into the parent workspace. The name is not case sensitive.
where_clause	The WHERE clause (excluding the WHERE keyword) identifying the rows to be merged into the parent workspace. Example: `'department_id = 20'` Only primary key columns can be specified in the WHERE clause. The WHERE clause cannot contain a subquery. If where_clause is not specified, all rows in table_name are merged.
create_savepoint	A Boolean value (TRUE or FALSE). TRUE creates an implicit savepoint in the parent workspace before the merge operation. (Implicit and explicit savepoints are described in Section 1.2.2.) FALSE (the default) does not create an implicit savepoint in the parent workspace before the merge operation.

Table 2–22 MergeTable Procedure Parameters (Cont.)

Parameter	Description
remove_data	A Boolean value (TRUE or FALSE). TRUE removes the data in the table (as specified by where_clause) in the child workspace after the merge operation, by rolling back to when the workspace was created. FALSE (the default) does not remove the data in the table in the child workspace after the merge operation; the table data in the child workspace is unchanged.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction. For more information, see Section 1.2.4.

Usage Notes

All data that satisfies the where_clause in the version-enabled table table_name in workspace is applied to the parent workspace of workspace.

Any locks that are held by rows being merged are released.

If there are conflicts between the workspace being merged and its parent workspace, the merge operation fails and the user must manually resolve conflicts using the <table_name>_CONF view. (Conflict resolution is explained in Section 1.2.5.)

A table cannot be merged in the LIVE workspace (because that workspace has no parent workspace).

A table cannot be merged or refreshed if there is an open regular transaction affecting the table.

An exception is raised if the user does not have access to table_id, or if the user does not have the MERGE_WORKSPACE privilege for workspace or the MERGE_ANY_WORKSPACE privilege.

Examples

The following example merges changes to the EMP table (in the USER3 schema) where last_name = 'Smith' in NEWWORKSPACE to its parent workspace.

```
EXECUTE DBMS_WM.MergeTable ('NEWWORKSPACE', 'user3.emp', 'last_name =
''Smith''');
```

MergeWorkspace

Purpose

Applies all changes in a workspace to its parent workspace, and optionally removes the workspace.

Syntax

```
DBMS_WM.MergeWorkspace(
   workspace            IN VARCHAR2
   [, create_savepoint  IN BOOLEAN DEFAULT FALSE]
   [, remove_workspace  IN BOOLEAN DEFAULT FALSE]
   [, auto_commit       IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 2–23 MergeWorkspace Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.
create_savepoint	A Boolean value (TRUE or FALSE). TRUE creates an implicit savepoint in the parent workspace before the merge operation. (Implicit and explicit savepoints are described in Section 1.2.2.) FALSE (the default) does not create an implicit savepoint in the parent workspace before the merge operation.
remove_workspace	A Boolean value (TRUE or FALSE). TRUE removes workspace after the merge operation. FALSE (the default) does not remove workspace after the merge operation; the workspace continues to exist.

Table 2–23 MergeWorkspace Procedure Parameters (Cont.)

Parameter	Description
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction. For more information, see Section 1.2.4.

Usage Notes

All data in all version-enabled tables in workspace is merged to the parent workspace of workspace, and workspace is removed if remove_workspace is TRUE.

While this procedure is executing, the current workspace is frozen in NO_ACCESS mode and the parent workspace is frozen in READ_ONLY mode, as explained in Section 1.2.6.

If there are conflicts between the workspace being merged and its parent workspace, the merge operation fails and the user must manually resolve conflicts using the <table_name>_CONF view. (Conflict resolution is explained in Section 1.2.5.)

If the remove_workspace parameter value is TRUE, the workspace to be merged must be a leaf workspace, that is, a workspace with no descendant workspaces. (For an explanation of workspace hierarchy, see Section 1.2.1.)

An exception is raised if the user does not have the MERGE_WORKSPACE privilege for workspace or the MERGE_ANY_WORKSPACE privilege.

Examples

The following example merges changes in NEWWORKSPACE to its parent workspace and removes (by default) NEWWORKSPACE.

```
EXECUTE DBMS_WM.MergeWorkspace ('NEWWORKSPACE');
```

RefreshTable

Purpose

Applies to a workspace all changes made to a table (all rows or as specified in the WHERE clause) in its parent workspace.

Syntax

```
DBMS_WM.RefreshTable(
   workspace        IN VARCHAR2,
   table_id         IN VARCHAR2
   [, where_clause  IN VARCHAR2 DEFAULT '']
   [, auto_commit   IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 2–24 RefreshTable Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.
table_id	Name of the table containing the rows to be refreshed using values from the parent workspace. The name is not case sensitive.
where_clause	The WHERE clause (excluding the WHERE keyword) identifying the rows to be refreshed from the parent workspace. Example: 'department_id = 20' Only primary key columns can be specified in the WHERE clause. The WHERE clause cannot contain a subquery. If where_clause is not specified, all rows in table_name are refreshed.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction. For more information, see Section 1.2.4.

Usage Notes

This procedure applies to workspace all changes in rows that satisfy the where_clause in the version-enabled table table_id in the parent workspace since the time when workspace was created or last refreshed.

If there are conflicts between the workspace being refreshed and its parent workspace, the refresh operation fails and the user must manually resolve conflicts using the <table_name>_CONF view. (Conflict resolution is explained in Section 1.2.5.)

A table cannot be refreshed in the LIVE workspace (because that workspace has no parent workspace).

A table cannot be merged or refreshed if there is an open regular transaction affecting the table.

An exception is raised if the user does not have access to table_id, or if the user does not have the MERGE_WORKSPACE privilege for workspace or the MERGE_ANY_WORKSPACE privilege.

Examples

The following example refreshes NEWWORKSPACE by applying changes made to the EMPLOYEES table where last_name = 'Smith' in its parent workspace.

```
EXECUTE DBMS_WM.RefreshTable ('NEWWORKSPACE', 'employees', 'last_name =
''Smith''');
```

RefreshWorkspace

Purpose

Applies to a workspace all changes made in its parent workspace.

Syntax

```
DBMS_WM.RefreshWorkspace(
   workspace      IN VARCHAR2
   [, auto_commit IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 2–25 RefreshWorkspace Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction. For more information, see Section 1.2.4.

Usage Notes

This procedure applies to workspace all changes made to version-enabled tables in the parent workspace since the time when workspace was created or last refreshed.

If there are conflicts between the workspace being refreshed and its parent workspace, the refresh operation fails and the user must manually resolve conflicts using the <table_name>_CONF view. (Conflict resolution is explained in Section 1.2.5.)

The specified workspace and the parent workspace are frozen in READ_ONLY mode, as explained in Section 1.2.6.

The LIVE workspace cannot be refreshed (because it has no parent workspace).

An exception is raised if the user does not have the MERGE_WORKSPACE privilege for workspace or the MERGE_ANY_WORKSPACE privilege.

Examples

The following example refreshes NEWWORKSPACE by applying changes made in its parent workspace.

```
EXECUTE DBMS_WM.RefreshWorkspace ('NEWWORKSPACE');
```

RemoveWorkspace

Purpose

Rolls back the data in the workspace and removes all support structures created for the workspace. The workspace ceases to exist.

Syntax

```
DBMS_WM.RemoveWorkspace(
   workspace       IN VARCHAR2
   [, auto_commit  IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 2–26 RemoveWorkspace Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction. For more information, see Section 1.2.4.

Usage Notes

The RemoveWorkspace operation can only be performed on leaf workspaces (the bottom-most workspaces in a branch in the hierarchy). For an explanation of database workspace hierarchy, see Section 1.2.1.

There must be no other users in the workspace being removed.

An exception is raised if the user does not have the REMOVE_WORKSPACE privilege for workspace or the REMOVE_ANY_WORKSPACE privilege.

Examples

The following example removes the NEWWORKSPACE workspace.

```
EXECUTE DBMS_WM.RemoveWorkspace('NEWWORKSPACE');
```

RemoveWorkspaceTree

Purpose

Removes the specified workspace and all its descendant workspaces. The data in the workspaces is rolled back and the workspace structure is removed.

Syntax

```
DBMS_WM.RemoveWorkspaceTree(
   workspace       IN VARCHAR2
   [, auto_commit  IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 2–27 RemoveWorkspaceTree Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction. For more information, see Section 1.2.4.

Usage Notes

The RemoveWorkspaceTree operation should be used with extreme caution, because it removes support structures and rolls back changes in a workspace and all its descendants down to the leaf workspace or workspaces. For example, in the hierarchy shown in Figure 1–1 in Section 1.2.1, a RemoveWorkspaceTree operation specifying Workspace 1 removes Workspace 1, Workspace 2, and Workspace 3. (For an explanation of database workspace hierarchy, see Section 1.2.1.)

There must be no other users in `workspace` or any of its descendant workspaces.

An exception is raised if the user does not have the `REMOVE_WORKSPACE` privilege for `workspace` or any of its descendant workspaces.

Examples

The following example removes the NEWWORKSPACE workspace and all its descendant workspaces.

```
EXECUTE DBMS_WM.RemoveWorkspaceTree('NEWWORKSPACE');
```

ResolveConflicts

Purpose

Resolves conflicts between workspaces.

Syntax

```
DBMS_WM.ResolveConflicts(
   workspace     IN VARCHAR2,
   table_name    IN VARCHAR2,
   where_clause  IN VARCHAR2,
   keep          IN VARCHAR2);
```

Parameters

Table 2–28 ResolveConflicts Procedure Parameters

Parameter	Description
workspace	Name of the workspace to check for conflicts with other workspaces. The name is case sensitive.
table_name	Name of the table to check for conflicts. The name is not case sensitive.
where_clause	The WHERE clause (excluding the WHERE keyword) identifying the rows to be refreshed from the parent workspace. Example: `'department_id = 20'` Only primary key columns can be specified in the WHERE clause. The WHERE clause cannot contain a subquery.
keep	Workspace in favor of which to resolve conflicts: PARENT, CHILD, or BASE. PARENT causes the parent workspace rows to be copied to the child workspace. CHILD does not cause the child workspace rows to be copied immediately to the parent workspace. However, the conflict is considered resolved, and the child workspace rows are copied to the parent workspace when the child workspace is merged. BASE causes the base rows to be copied to the child workspace but not to the parent workspace. However, the conflict is considered resolved; and when the child workspace is merged, the base rows are copied to the parent workspace.

Usage Notes

This procedure checks the condition identified by `table_name` and `where_clause`, and it finds any conflicts between row values in `workspace` and its parent workspace. This procedure resolves conflicts by using the row values in the parent or child workspace, as specified in the `keep` parameter; however, the conflict resolution is not actually merged until you commit the transaction (standard database commit operation) and call the CommitResolve procedure to end the conflict resolution session. (For more information about conflict resolution, including an overall view of the process, see Section 1.2.5.)

For example, assume that for Department 20 (`DEPARTMENT_ID = 20`), the `MANAGER_NAME` in the `LIVE` and `Workspace1` workspaces is `Tom`. Then, the following operations occur:

1. The `manager_name` for Department 20 is changed in the `LIVE` database workspace from `Tom` to `Mary`.
2. The change is committed (a standard database commit operation).
3. The `manager_name` for Department 20 is changed in `Workspace1` from `Tom` to `Franco`.
4. The MergeWorkspace procedure is called to merge `Workspace1` changes to the `LIVE` workspace.

   At this point, however, a conflict exists with respect to `MANAGER_NAME` for Department 20 in `Workspace1` (`Franco`, which conflicts with `Mary` in the `LIVE` workspace), and therefore the call to MergeWorkspace does not succeed.
5. The ResolveConflicts procedure is called with the following parameters: (`'Workspace1'`, `'department'`, `'department_id = 20'`, `'child'`).

   After the MergeWorkspace operation in step 7, the `MANAGER_NAME` value will be `Franco` in both the `Workspace1` and `LIVE` workspaces.
6. The change is committed (a standard database commit operation).
7. The MergeWorkspace procedure is called to merge `Workspace1` changes to the `LIVE` workspace.

For more information about conflict resolution, see Section 1.2.5.

Examples

The following example resolves conflicts involving rows in the `DEPARTMENT` table in `Workspace1` where `DEPARTMENT_ID` is 20, and uses the values in the child workspace to resolve all such conflicts. It then merges the results of the conflict

resolution by first committing the transaction (standard commit) and then calling the MergeWorkspace procedure.

```
EXECUTE DBMS_WM.BeginResolve ('Workspace1');
EXECUTE  DBMS_WM.ResolveConflicts ('Workspace1', 'department', 'department_id =
20', 'child');
COMMIT;
EXECUTE DBMS_WM.CommitResolve ('Workspace1');
```

RevokeSystemPriv

Purpose

Revokes (removes) system-level privileges from users and roles.

Syntax

```
DBMS_WM.RevokeSystemPriv(
   priv_types      IN VARCHAR2,
   grantee         IN VARCHAR2
   [, auto_commit  IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 2–29 RevokeSystemPriv Procedure Parameters

Parameter	Description
priv_types	A string of one or more keywords representing privileges. (Section 1.4 discusses Workspace Manager privileges.) Use commas to separate privilege keywords. The available keywords are ACCESS_ANY_WORKSPACE, MERGE_ANY_WORKSPACE, CREATE_ANY_WORKSPACE, REMOVE_ANY_WORKSPACE, and ROLLBACK_ANY_WORKSPACE.
grantee	Name of the user (can be the PUBLIC user group) or role from which to revoke priv_types.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction. For more information, see Section 1.2.4.

Usage Notes

Contrast this procedure with RevokeWorkspacePriv, which revokes workspace-level Workspace Manager privileges with keywords in the form *xxx*_WORKSPACE (ACCESS_WORKSPACE, MERGE_WORKSPACE, and so on).

To grant system-level privileges, use the GrantSystemPriv procedure.

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

- grantee is not a valid user or role in the database.
- You do not have the privilege to revoke priv_types.

Examples

The following example disallows user Smith from accessing workspaces and merging changes in workspaces.

```
EXECUTE DBMS_WM.RevokeSystemPriv ('ACCESS_ANY_WORKSPACE, MERGE_ANY_WORKSPACE',
'Smith');
```

RevokeWorkspacePriv

Purpose

Revokes (removes) workspace-level privileges from users and roles for a specified workspace.

Syntax

```
DBMS_WM.RevokeWorkspacePriv(
   priv_types      IN VARCHAR2,
   workspace       IN VARCHAR2,
   grantee         IN VARCHAR2
   [, auto_commit  IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 2–30 RevokeWorkspacePriv Procedure Parameters

Parameter	Description
priv_types	A string of one or more keywords representing privileges. (Section 1.4 discusses Workspace Manager privileges.) Use commas to separate privilege keywords. The available keywords are ACCESS_WORKSPACE, MERGE_WORKSPACE, CREATE_WORKSPACE, REMOVE_WORKSPACE, and ROLLBACK_WORKSPACE.
workspace	Name of the workspace. The name is case sensitive.
grantee	Name of the user (can be the PUBLIC user group) or role from which to revoke priv_types.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction. For more information, see Section 1.2.4.

Usage Notes

Contrast this procedure with RevokeSystemPriv, which revokes system-level Workspace Manager privileges with keywords in the form *xxx*_ANY_WORKSPACE (ACCESS_ANY_WORKSPACE, MERGE_ANY_WORKSPACE, and so on).

To grant workspace-level privileges, use the GrantWorkspacePriv procedure.

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

- `grantee` is not a valid user or role in the database.
- You do not have the privilege to revoke `priv_types`.

Examples

The following example disallows user `Smith` from accessing the `NEWWORKSPACE` workspace and merging changes in that workspace.

```
EXECUTE DBMS_WM.RevokeWorkspacePriv ('ACCESS_WORKSPACE, MERGE_WORKSPACE',
'NEWWORKSPACE', 'Smith');
```

RollbackResolve

Purpose

Quits a conflict resolution session and discards all changes in the workspace since the BeginResolve procedure was executed.

Syntax

```
DBMS_WM.RollbackResolve(
   workspace  IN VARCHAR2);
```

Parameters

Table 2–31 RollbackResolve Procedure Parameters

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

Usage Notes

This procedure quits the current conflict resolution session (started by the BeginResolve procedure), and discards all changes in the workspace since the start of the conflict resolution session. Contrast this procedure with CommitResolve, which saves all changes.

While the conflict resolution session is being rolled back, the workspace is frozen in 1WRITER mode, as explained in Section 1.2.6.

For more information about conflict resolution, see Section 1.2.5.

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

- There are one or more open regular transactions in workspace.
- The procedure was called by a user that does not have the WM_ADMIN_ROLE role or that did not execute the BeginResolve procedure on workspace.

Examples

The following example quits the conflict resolution session in Workspace1 and discards all changes.

```
EXECUTE  DBMS_WM.RollbackResolve ('Workspace1');
```

RollbackTable

Purpose

Discards all changes made in the workspace to a specified table (all rows or as specified in the WHERE clause).

Syntax

```
DBMS_WM.RollbackTable(
   workspace        IN VARCHAR2,
   table_id         IN VARCHAR2,
   [, sp_name       IN VARCHAR2 DEFAULT '']
   [, where_clause  IN VARCHAR2 DEFAULT '']
   [, remove_locks  IN BOOLEAN DEFAULT TRUE]
   [, auto_commit   IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 2–32 RollbackTable Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.
table_id	Name of the table containing rows to be discarded. The name is not case sensitive.
sp_name	Name of the savepoint to which to roll back. The name is case sensitive. The default is to discard all changes (that is, ignore any savepoints).
where_clause	The WHERE clause (excluding the WHERE keyword) identifying the rows to be discarded. Example: 'department_id = 20' Only primary key columns can be specified in the WHERE clause. The WHERE clause cannot contain a subquery. If where_clause is not specified, all rows that meet the criteria of the other parameters are discarded.
remove_locks	A Boolean value (TRUE or FALSE). TRUE (the default) releases those locks on rows in the parent workspace that satisfy the condition in where_clause and that were not versioned in the child workspace. This option has no effect if the table has been rolled back to a savepoint. FALSE does not release any locks in the parent workspace.

Table 2–32 RollbackTable Procedure Parameters (Cont.)

Parameter	Description
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction. For more information, see Section 1.2.4.

Usage Notes

You cannot roll back to a savepoint if any implicit savepoints have been created since the specified savepoint, unless you first merge or remove the descendant workspaces that caused the implicit savepoints to be created. For example, referring to Figure 1–2 in Section 1.2.2, the user in Workspace 1 cannot roll back to savepoint SP1 until Workspace 3 (which caused implicit savepoint SPc to be created) is merged or removed.

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

- workspace does not exist.
- You do not have the privilege to roll back workspace or any affected table.
- A regular transaction affecting table_id is active in workspace.

Examples

The following example rolls back all changes made to the EMP table (in the USER3 schema) in the NEWWORKSPACE workspace since that workspace was created.

```
EXECUTE DBMS_WM.RollbackTable ('NEWWORKSPACE', 'user3.emp');
```

RollbackToSP

Purpose

Discards all changes made after a specified savepoint in the workspace to all tables.

Syntax

```
DBMS_WM.RollbackToSP(
   workspace       IN VARCHAR2,
   savepoint_name  IN VARCHAR2
   [, auto_commit  IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 2–33 RollbackToSP Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.
savepoint_name	Name of the savepoint to which to roll back changes. The name is case sensitive.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction. For more information, see Section 1.2.4.

Usage Notes

While this procedure is executing, the workspace is frozen in NO_ACCESS mode.

Contrast this procedure with RollbackWorkspace, which rolls back all changes made since the creation of the workspace.

You cannot roll back to a savepoint if any implicit savepoints have been created since the specified savepoint, unless you first merge or remove the descendant workspaces that caused the implicit savepoints to be created. For example, referring to Figure 1–2 in Section 1.2.2, the user in Workspace 1 cannot roll back to savepoint

SP1 until Workspace 3 (which caused implicit savepoint SPc to be created) is merged or removed.

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

- `workspace` does not exist.
- `savepoint_name` does not exist.
- One or more implicit savepoints have been created in `workspace` after `savepoint_name`, and the descendant workspaces that caused the implicit savepoints to be created still exist.
- You do not have the privilege to roll back `workspace` or any affected table.
- Any sessions are active in `workspace`.

Examples

The following example rolls back any changes made in the `NEWWORKSPACE` workspace to all tables since the creation of `Savepoint1`.

```
EXECUTE DBMS_WM.RollbackToSP ('NEWWORKSPACE', 'Savepoint1');
```

RollbackWorkspace

Purpose

Discards all changes made in the workspace to all tables.

Syntax

```
DBMS_WM.RollbackWorkspace(
   workspace       IN VARCHAR2
   [, auto_commit  IN BOOLEAN DEFAULT TRUE]);
```

Parameters

Table 2–34 RollbackWorkspace Procedure Parameters

Parameter	Description
workspace	Name of the workspace. The name is case sensitive.
auto_commit	A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction. For more information, see Section 1.2.4.

Usage Notes

Only leaf workspaces can be rolled back. That is, a workspace cannot be rolled back if it has any descendant workspaces. (For an explanation of workspace hierarchy, see Section 1.2.1.)

Contrast this procedure with RollbackToSP, which rolls back changes to a specified savepoint.

Like the RemoveWorkspace procedure, RollbackWorkspace deletes the data in the workspace; however, unlike the RemoveWorkspace procedure, RollbackWorkspace does not delete the Workspace Manager workspace structure.

While this procedure is executing, the specified workspace is frozen in NO_ACCESS mode, as explained in Section 1.2.6.

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

- workspace has any descendant workspaces.
- workspace does not exist.
- You do not have the privilege to roll back workspace or any affected table.
- Any sessions are active in workspace.

Examples

The following example rolls back any changes made in the NEWWORKSPACE workspace since that workspace was created.

```
EXECUTE DBMS_WM.RollbackWorkspace ('NEWWORKSPACE');
```

SetConflictWorkspace

Purpose

Determine whether or not conflicts exist between a workspace and its parent.

Syntax

```
DBMS_WM.SetConflictWorkspace(
   workspace  IN VARCHAR2);
```

Parameters

Table 2–35 SetConflictWorkspace Procedure Parameters

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

Usage Notes

This procedure checks for any conflicts between `workspace` and its parent workspace, and it modifies the content of the <table_name>_CONF views (explained in Section 3.21) as needed.

A `SELECT` operation from the <table_name>_CONF views for all tables modified in a workspace displays all rows in the workspace that are in conflict with the parent workspace. (To obtain a list of tables that may have been changed in the workspace, use the SQL statement `SELECT * FROM ALL_VERSIONED_TABLES`. The SQL statement `SELECT * FROM <table_name>_CONF` displays conflicts for <table_name> between the current workspace and its parent workspace.)

Any conflicts must be resolved before a workspace can be merged or refreshed. To resolve a conflict, you must use the ResolveConflicts procedure (and then merge the result of the resolution by using the MergeWorkspace procedure).

Examples

The following example checks for any conflicts between `B_focus_2` and its parent workspace, and modifies the contents of the <table_name>_CONF views as needed.

```
EXECUTE DBMS_WM.SetConflictWorkspace ('B_focus_2');
```

SetDiffVersions

Purpose

Finds differences in values in version-enabled tables for two savepoints and their common ancestor (base). It modifies the contents of the differences views that describe these differences.

Syntax

```
DBMS_WM.SetDiffVersions(
   workspace1  IN VARCHAR2,
   workspace2  IN VARCHAR2);
```

or

```
DBMS_WM.SetDiffVersions(
   workspace1  IN VARCHAR2,
   savepoint1  IN VARCHAR2,
   workspace2  IN VARCHAR2,
   savepoint2  IN VARCHAR2);
```

Parameters

Table 2–36 SetDiffVersions Procedure Parameters

Parameter	Description
workspace1	Name of the first workspace to be checked for differences in version-enabled tables. The name is case sensitive.
savepoint1	Name of the savepoint in workspace1 for which values are to be checked. The name is case sensitive. If savepoint1 and savepoint2 are not specified, the rows in version-enabled tables for the LATEST savepoint in each workspace are checked. (The LATEST savepoint is explained in Section 1.2.2.)
workspace2	Name of the second workspace to be checked for differences in version-enabled tables. The name is case sensitive.
savepoint2	Name of the savepoint in workspace2 for which values are to be checked. The name is case sensitive.

Usage Notes

This procedure modifies the contents of the differences views (xxx_DIFF), which are described in Section 3.22. Each call to the procedure populates one or more sets of three rows, each set consisting of:

- Values for the common ancestor
- Values for `workspace1` (`savepoint1` or `LATEST` savepoint values)
- Values for `workspace2` (`savepoint2` or `LATEST` savepoint values)

You can then select rows from the appropriate xxx_DIFF view or views to check comparable table values in the two savepoints and their common ancestor. The common ancestor (or *base*) is identified as `DiffBase` in xxx_DIFF view rows.

Examples

The following example checks the differences in version-enabled tables for the `B_focus_1` and `B_focus_2` workspaces. (The output has been reformatted for readability.)

```
SQL> -- Add rows to difference view: COLA_MARKETING_BUDGET_DIFF
SQL> EXECUTE DBMS_WM.SetDiffVersions ('B_focus_1', 'B_focus_2');

SQL> -- View the rows that were just added.
SQL> SELECT * from COLA_MARKETING_BUDGET_DIFF;

MKT_ID  MKT_NAME  MANAGER  BUDGET  WM_DIFFVER          WMCODE
------  --------  -------  ------  -----------         --------
   1    cola_a    Alvarez   2      DiffBase            NC
   1    cola_a    Alvarez   1.5    B_focus_1, LATEST   U
   1    cola_a    Alvarez   2      B_focus_2, LATEST   NC
   2    cola_b    Burton    2      DiffBase            NC
   2    cola_b    Beasley   3      B_focus_1, LATEST   U
   2    cola_b    Burton    2.5    B_focus_2, LATEST   U
   3    cola_c    Chen      1.5    DiffBase            NC
   3    cola_c    Chen      1      B_focus_1, LATEST   U
   3    cola_c    Chen      1.5    B_focus_2, LATEST   NC
   4    cola_d    Davis     3.5    DiffBase            NC
   4    cola_d    Davis     3      B_focus_1, LATEST   U
   4    cola_d    Davis     2.5    B_focus_2, LATEST   U

12 rows selected.
```

Section 3.22 explains how to interpret and use the information in the differences (xxx_DIFF) views.

SetLockingOFF

Purpose

Enables access to versioned rows and to corresponding rows in the parent workspace.

Syntax

```
DBMS_WM.SetLockingOFF();
```

Parameters

None.

Usage Notes

This procedure turns off Workspace Manager locking that had been set on by the SetLockingON procedure. Existing locks applied by this session remain locked. All new changes by this session are not locked.

Examples

The following example sets locking off for the session.

```
EXECUTE DBMS_WM.SetLockingOFF;
```

SetLockingON

Purpose

Controls access to versioned rows and to corresponding rows in the previous version.

Syntax

```
DBMS_WM.SetLockingON(
   lockmode  IN VARCHAR2);
```

Parameters

Table 2–37 SetLockingON Procedure Parameters

Parameter	Description
lockmode	Locking mode. Must be E, S, or C. E (exclusive) mode locks the rows in the previous version and the corresponding rows in the current version; no other users in the workspace for either version can change any values. S (shared) mode locks the rows in the previous version and the corresponding rows in the current version; however, other users in the workspace for the current version (but no users in the workspace for the previous version) can change values in these rows. C (carry-forward) mode locks rows in the current workspace with the same locking mode as the corresponding rows in the previous version. (If a row is not locked in the previous version, its corresponding row in the current version is not locked.)

Usage Notes

This procedure affects Workspace Manager locking, which occurs in addition to any standard Oracle server locking. Workspace Manager locks can be used to prevent conflicts. When a user locks a row, the corresponding row in the parent workspace is also locked. Thus, when this workspace merges with the parent at merge time, it is guaranteed that this row will not have a conflict.

Exclusive locking prevents the use of *what-if* scenarios in which different values for one or more columns are tested. Thus, plan any testing of scenarios when exclusive locking is not in effect.

Locking is enabled at the user session level, and the locking mode stays in effect until any of the following occurs:

- The session goes to another workspace or connects to the database, in which case the locking mode is set to C (carry-forward) unless another locking mode has been specified using the SetWorkspaceLockModeON procedure.
- The session executes the SetLockingOFF procedure.

The locks remain in effect for the duration of the workspace, unless unlocked by the UnlockRows procedure. (Existing locks are not affected by the SetLockingOFF procedure.)

There are no specific privileges associated with locking. Any session that can go to a workspace can set locking on.

Examples

The following example sets exclusive locking on for the session.

```
EXECUTE DBMS_WM.SetLockingON ('E');
```

All rows locked by this user remain locked until the workspace is merged or rolled back.

SetMultiWorkspaces

Purpose

Makes the specified workspace or workspaces visible in the multiworkspace views for version-enabled tables.

Syntax

```
DBMS_WM.SetMultiWorkspaces(
   workspaces  IN VARCHAR2);
```

Parameters

Table 2–38 SetMultiWorkspaces Procedure Parameters

Parameter	Description
workspaces	The workspace or workspaces for which information is to be added to the multiworkspace views (described in Section 3.25). The workspace names are case sensitive. To specify more than one workspace (but no more than eight), use a comma to separate workspace names. For example: `'workspace1,workspace2'`

Usage Notes

This procedure adds rows to the multiworkspace views (xxx_MS). See Section 3.25 for information about the contents and uses of these views.

To see the names of workspaces visible in the multiworkspace views, use the GetMultiWorkspaces function.

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

- The user does not have the privilege to go to one or more of the workspaces named in `workspaces`.
- A workspace named in `workspaces` is not valid.
- More than eight workspace names are specified in `workspaces`.

Examples

The following example adds information to the multiworkspace views for version-enabled tables in the `B_focus_1` workspace.

```
SQL> EXECUTE DBMS_WM.SetMultiWorkspaces ('B_focus_1');
```

SetWoOverwriteOFF

Purpose

Disables the VIEW_WO_OVERWRITE history option that had been enabled by the EnableVersioning or SetWoOverwriteON procedure, changing the option to VIEW_W_OVERWRITE (*with overwrite*).

Syntax

```
DBMS_WM.SetWoOverwriteOFF();
```

Parameters

None.

Usage Notes

This procedure affects the recording of history information in the views named <table_name>_HIST by changing the VIEW_WO_OVERWRITE option to VIEW_W_OVERWRITE. That is, from this point forward, the views show only the most recent modifications to the same version of the table. A history of modifications to the version is not maintained; that is, subsequent changes to a row in the same version overwrite earlier changes.

This procedure affects only tables that were version-enabled with the hist parameter set to VIEW_WO_OVERWRITE in the call to the EnableVersioning procedure.

The <table_name>_HIST views are described in Section 3.24. The VIEW_WO_OVERWRITE and VIEW_W_OVERWRITE options are further described in the description of the EnableVersioning procedure.

The history option affects the behavior of the GotoDate procedure. See the Usage Notes for that procedure.

The result of the SetWoOverwriteOFF procedure remains in effect only for the duration of the current session. To reverse the effect of this procedure, use the SetWoOverwriteON procedure.

Examples

The following example disables the VIEW_WO_OVERWRITE history option.

```
EXECUTE DBMS_WM.SetWoOverwriteOFF;
```

SetWoOverwriteON

Purpose

Enables the VIEW_WO_OVERWRITE history option that had been disabled by the SetWoOverwriteOFF procedure.

Syntax

```
DBMS_WM.SetWoOverwriteON();
```

Parameters

None.

Usage Notes

This procedure affects the recording of history information in the views named <table_name>_HIST by changing the VIEW_W_OVERWRITE option to VIEW_WO_OVERWRITE (*without overwrite*). That is, from this point forward, the views show all modifications to the same version of the table. A history of modifications to the version is maintained; that is, subsequent changes to a row in the same version do not overwrite earlier changes.

This procedure affects only tables that were affected by a previous call to the SetWoOverwriteOFF procedure.

The <table_name>_HIST views are described in Section 3.24. The VIEW_WO_OVERWRITE and VIEW_W_OVERWRITE options are further described in the description of the EnableVersioning procedure.

The VIEW_WO_OVERWRITE history option can be overridden when a workspace is compressed by specifying the compress_view_wo_overwrite parameter as TRUE with the CompressWorkspace or CompressWorkspaceTree procedure.

The history option affects the behavior of the GotoDate procedure. See the Usage Notes for that procedure.

To reverse the effect of this procedure, use the SetWoOverwriteOFF procedure.

Examples

The following example enables the VIEW_WO_OVERWRITE history option.

```
EXECUTE DBMS_WM.SetWoOverwriteON;
```

SetWorkspaceLockModeOFF

Purpose

Enables access to versioned rows in the specified workspace and to corresponding rows in the parent workspace.

Syntax

```
DBMS_WM.SetWorkspaceLockModeOFF(
   workspace  IN VARCHAR2);
```

Parameters

Table 2–39 SetWorkspaceLockModeOFF Procedure Parameters

Parameter	Description
workspace	Name of the workspace for which to set the locking mode off. The name is case sensitive.

Usage Notes

This procedure turns off Workspace Manager locking that had been set on by the SetWorkspaceLockModeON procedure. Existing locks applied by this session remain locked. All new changes by this session or a subsequent session are not locked, unless the session turns locking on by executing the SetLockingON procedure.

An exception is raised if any of the following occurs:

- The user does not have the WM_ADMIN_ROLE role or is not the owner of workspace.
- There are any open regular transactions in workspace.
- workspace is a continually refreshed workspace (see the description of the isrefreshed parameter of the CreateWorkspace procedure).

Examples

The following example sets locking off for the workspace named NEWWORKSPACE.

```
EXECUTE DBMS_WM.SetWorkspaceLockModeOFF('NEWWORKSPACE');
```

SetWorkspaceLockModeON

Purpose

Sets the default mode for the row-level locking in the workspace.

Syntax

```
DBMS_WM.SetWorkspaceLockModeON(
   workspace    IN VARCHAR2,
   lockmode     IN VARCHAR2
   [, override  IN BOOLEAN DEFAULT FALSE]);
```

Parameters

Table 2–40 SetWorkspaceLockModeON Procedure Parameters

Parameter	Description
workspace	Name of the workspace for which to set the locking mode. The name is case sensitive.
lockmode	Locking mode. Must be E, S, or C. E (exclusive) mode locks the rows in the parent workspace and the corresponding rows in the current workspace; no other users in either workspace can change any values. S (shared) mode locks the rows in the parent workspace and the corresponding rows in the current workspace; however, other users in the current workspace (but no users in the parent workspace) can change values in these rows. C (carry-forward) mode locks rows in the current workspace with the same locking mode as the corresponding rows in the parent workspace. (If a row is not locked in the parent workspace, its corresponding row in the child workspace is not locked.)
override	A Boolean value (TRUE or FALSE) TRUE allows a session in the workspace to change the lockmode value by using the SetLockingON and SetLockingOFF procedures. FALSE (the default) prevents a session in the workspace from changing the lockmode value.

Usage Notes

This procedure affects Workspace Manager locking, which occurs in addition to any standard Oracle server locking. Workspace Manager locks can be used to prevent conflicts. When a user locks a row, the corresponding row in the parent workspace is also locked. Thus, when this workspace merges with the parent at merge time, it is guaranteed that this row will not have a conflict.

Exclusive locking prevents the use of *what-if* scenarios in which different values for one or more columns are tested. Thus, plan any testing of scenarios when exclusive locking is not in effect.

If the override parameter value is `TRUE`, locking can also be enabled and disabled at the user session level with the SetLockingON and SetLockingOFF procedures, respectively.

All new changes by this session or a subsequent session are locked, unless the session turns locking off by executing the SetLockingOFF procedure.

An exception is raised if any of the following occurs:

- The user does not have the `WM_ADMIN_ROLE` role or is not the owner of `workspace`.
- There are any open regular transactions in `workspace`.
- `workspace` is a continually refreshed workspace (see the description of the `isrefreshed` parameter of the CreateWorkspace procedure).

Examples

The following example sets exclusive locking on for the workspace named `NEWWORKSPACE`.

```
EXECUTE DBMS_WM.SetWorkspaceLockModeON ('NEWWORKSPACE', 'E');
```

All locked rows remain locked until the workspace is merged or rolled back.

UnfreezeWorkspace

Purpose

Enables changes to a workspace, reversing the effect of the FreezeWorkspace procedure.

Syntax

```
DBMS_WM.UnfreezeWorkspace(
   workspace  IN VARCHAR2);
```

Parameters

Table 2–41 UnfreezeWorkspace Procedure Parameters

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

Usage Notes

The operation fails if any sessions are active in workspace.

Only the owner of the workspace or a user with WM_ADMIN_ROLE can unfreeze a workspace. There are no specific privileges associated with freezing a workspace.

Examples

The following example unfreezes the NEWWORKSPACE workspace.

```
EXECUTE DBMS_WM.FreezeWorkspace ('NEWWORKSPACE');
```

UnlockRows

Purpose

Enables access to versioned rows in a specified table and to corresponding rows in the parent workspace.

Syntax

```
DBMS_WM.UnlockRows(
   workspace        IN VARCHAR2,
   table_name       IN VARCHAR2
   [, where_clause  IN VARCHAR2 DEFAULT ''']
   [, all_or_user   IN VARCHAR2 DEFAULT 'USER']
   [, lock_mode     IN VARCHAR2 DEFAULT 'ES']);
```

Parameters

Table 2–42 UnlockRows Procedure Parameters

Parameter	Description
workspace	Name of the workspace: locked rows in this workspace and corresponding rows in the parent workspace will be unlocked, as specified in the remaining parameters. The name is case sensitive.
table_name	Name of the table in which rows are to be unlocked. The name is not case sensitive.
where_clause	The WHERE clause (excluding the WHERE keyword) identifying the rows to be unlocked. Example: 'department_id = 20' Only primary key columns can be specified in the WHERE clause. The WHERE clause cannot contain a subquery. If where_clause is not specified, all rows in table_name are made accessible.
all_or_user	Scope of the request: ALL or USER. ALL: All locks accessible by the user in the current workspace are considered. USER (default): Only locks owned by the user in the current workspace are considered.

Table 2–42 UnlockRows Procedure Parameters (Cont.)

Parameter	Description
lock_mode	Locking mode: E, S, or ES. E: Only exclusive mode locks are considered. S: Only shared mode locks are considered. ES (default): Both exclusive mode and shared mode locks are considered.

Usage Notes

This procedure affects Workspace Manager locking, which occurs in addition to any standard Oracle server locking. For an explanation of Workspace Manager locking, see Section 1.3.

This procedure unlocks rows that had been previously locked (see the LockRows procedure). It does not affect whether Workspace Manager locking is set on or off (determined by the SetLockingON and SetLockingOFF procedures).

Examples

The following example unlocks the EMPLOYEES table where last_name = 'Smith' in the NEWWORKSPACE workspace.

```
EXECUTE DBMS_WM.UnlockRows ('employees', 'NEWWORKSPACE', 'last_name =
''Smith''');
```

3

Metadata Views

Workspace Manager creates and maintains metadata views to hold information about such things as version-enabled tables, workspaces, savepoints, users, privileges, locks, and conflicts. These views are read-only to users. You can use the information in these views to help administer the long transaction environment and diagnose problems.

- USER_WM_VERSIONED_TABLES (Section 3.17) and ALL_WM_VERSIONED_TABLES (Section 3.6) contain information about version-enabled tables.
- USER_WM_MODIFIED_TABLES (Section 3.13) and ALL_WM_MODIFIED_TABLES (Section 3.3) contain information about version-enabled tables that have been modified.
- USER_WORKSPACES (Section 3.20) and ALL_WORKSPACES (Section 3.9) contain information about workspaces.
- USER_WORKSPACE_SAVEPOINTS (Section 3.19) and ALL_WORKSPACE_SAVEPOINTS (Section 3.8) contain information about savepoints.
- USER_WORKSPACE_PRIVS (Section 3.18) and ALL_WORKSPACE_PRIVS (Section 3.7) contain information about privileges specific to Workspace Manager.
- USER_WM_PRIVS (Section 3.14) contains information about privileges that the current user has in each workspace.
- ROLE_WM_PRIVS (Section 3.11) contains information about privileges that all roles granted to the current user have in each workspace.
- USER_WM_LOCKED_TABLES (Section 3.12) and ALL_WM_LOCKED_TABLES (Section 3.2) contain information about locks placed in the current workspace on rows in version-enabled tables.

- DBA_WORKSPACE_SESSIONS (Section 3.10) contains information about all users in all workspaces other than LIVE.
- USER_WM_RIC_INFO (Section 3.15) and ALL_WM_RIC_INFO (Section 3.4) contain information about referential integrity constraints.
- USER_WM_TAB_TRIGGERS (Section 3.16) and ALL_WM_TAB_TRIGGERS (Section 3.5) contain information about triggers defined on version-enabled tables.
- ALL_VERSION_HVIEW (Section 3.1) contains information about the version hierarchy.

There are also views created per version-enabled table, as follows:

- Conflict view, each having a name in the form <table_name>_CONF. (See Section 3.21.)
- Difference view, each having a name in the form <table_name>_DIFF. (See Section 3.22.)
- Lock view, each having a name in the form <table_name>_LOCK. (See Section 3.23.)
- History view (if history tracking is enabled), each having a name in the form <table_name>_HIST. (See Section 3.24.)
- Multiworkspace view, each having a name in the form <table_name>_MW. (See Section 3.25.)

3.1 ALL_VERSION_HVIEW

ALL_VERSION_HVIEW contains information about the version hierarchy. It is used by Workspace Manager to perform queries against the xxx_HIST views (described in Section 3.24).

Column	Datatype	Null?	Description
VERSION	NUMBER(38)	NOT NULL	Version number of the workspace identified in the WORKSPACE column.
PARENT_VERSION	NUMBER(38)		Version number of the parent version of the version identified in the VERSION column.
WORKSPACE	VARCHAR2(30)		Name of the workspace associated with the version number in the VERSION column.

3.2 ALL_WM_LOCKED_TABLES

ALL_WM_LOCKED_TABLES contains information about Workspace Manager locks on rows in version-enabled tables that the current user can access.

Related View

- USER_WM_LOCKED_TABLES (Section 3.12) contains information about Workspace Manager locks on rows in version-enabled tables of which the current user is the owner.

Column	Datatype	Null?	Description
TABLE_OWNER	VARCHAR2(40)		User name of the table owner.
TABLE_NAME	VARCHAR2(40)		Name of the table.
LOCK_MODE	VARCHAR2(9)		Type of lock: EXCLUSIVE or SHARED.
LOCK_OWNER	VARCHAR2(4000)		User name of the owner of the lock.
LOCKING_WORKSPACE	VARCHAR2(4000)		Workspace in which the lock was placed.

3.3 ALL_WM_MODIFIED_TABLES

ALL_WM_MODIFIED_TABLES contains information about all version-enabled tables that have been modified and on which the current user has one or more of the following privileges: SELECT, INSERT, DELETE, UPDATE.

Related View

- USER_WM_MODIFIED_TABLES (Section 3.13) contains information about version-enabled tables that have been modified and of which the current user is the owner.

Column	Datatype	Null?	Description
TABLE_NAME	VARCHAR2(61)	NOT NULL	Name of a version-enabled table.
WORKSPACE	VARCHAR2(30)	NOT NULL	Workspace in which the modification occurred.
SAVEPOINT	VARCHAR2(30)		Name of the savepoint associated with the most recent modification, or LATEST if a savepoint does not yet exist is the workspace.

3.4 ALL_WM_RIC_INFO

ALL_WM_RIC_INFO contains information about referential integrity constraints in version-enabled tables that the current user can access. Workspace Manager uses this information to provide referential integrity support, which is described in Section 1.6.

Related View

- USER_WM_RIC_INFO (Section 3.15) contains information about referential integrity constraints in version-enabled tables of which the current user is the owner.

Column	Datatype	Null?	Description
CT_OWNER	VARCHAR2(40)	NOT NULL	Owner of the child table in the referential integrity constraint.
CT_NAME	VARCHAR2(40)		Name of the child table in the referential integrity constraint.
PT_OWNER	VARCHAR2(40)		Owner of the parent table in the referential integrity constraint.
PT_NAME	VARCHAR2(40)		Name of the parent table in the referential integrity constraint.
RIC_NAME	VARCHAR2(40)	NOT NULL	Name of the referential integrity constraint.
CT_COLS	VARCHAR2(4000)		List of foreign key columns in the child table in the referential integrity constraint.
PT_COLS	VARCHAR2(4000)		List of foreign key columns in the parent table in the referential integrity constraint.
R_CONSTRAINT_NAME	VARCHAR2(40)		Name of the unique constraint defined on the parent table in the referential integrity constraint.
DELETE_RULE	VARCHAR2(2)		Rule to apply when deletion occurs in the parent table. C (Cascade) causes related child table rows to be deleted; R (Restrict) prevents the deletion if any related child table rows exist.
STATUS	VARCHAR2(8)		ENABLED if the referential integrity constraint is enabled, or DISABLED if the referential integrity constraint is disabled.

3.5 ALL_WM_TAB_TRIGGERS

ALL_WM_TAB_TRIGGERS contains information about triggers that the current user created and for version-enabled tables owned by the current user that have triggers defined on them. If the current user has the CREATE ANY TRIGGER privilege, trigger information is displayed for all version-enabled tables.

Related View

- USER_WM_TAB_TRIGGERS (Section 3.16) contains information about triggers that are owned by the current user and that are on version-enabled tables.

Column	Datatype	Null?	Description
TRIGGER_OWNER	VARCHAR2(50)	NOT NULL	Owner (schema) of the trigger.
TRIGGER_NAME	VARCHAR2(50)	NOT NULL	Name of the trigger.
TABLE_OWNER	VARCHAR2(50)		Owner (schema) of the table on which the trigger is defined.
TABLE_NAME	VARCHAR2(50)		Name of the table on which the trigger is defined.
TRIGGER_TYPE	VARCHAR2(3)		Trigger type: one of the codes described following this table.
STATUS	VARCHAR2(10)		ENABLED if the trigger is enabled; DISABLED if the trigger is disabled.
WHEN_CLAUSE	VARCHAR2(4000)		Clause that must evaluate to TRUE for the trigger body (TRIGGER_BODY) to execute.
DESCRIPTION	VARCHAR2(4000)		Description of the trigger. Useful if the trigger must be re-created.
TRIGGER_BODY	LONG		Statements executed by the trigger.

TRIGGER_TYPE is one of the following values:

- BIR: before insert for each row
- AIR: after insert for each row
- BUR: before update for each row
- AUR: after update for each row
- BDR: before delete for each row
- ADR: after delete for each row
- BIS: before insert for each statement
- AIS: after insert for each statement
- BUS: before update for each statement
- AUS: after update for each statement
- BDS: before delete for each statement
- ADS: after delete for each statement

3.6 ALL_WM_VERSIONED_TABLES

ALL_WM_VERSIONED_TABLES contains information about all version-enabled tables on which the current user has one or more of the following privileges: SELECT, INSERT, DELETE, UPDATE.

Related View

- USER_WM_VERSIONED_TABLES (Section 3.17) contains information about version-enabled tables of which the current user is the owner.

Column	Datatype	Null?	Description
TABLE_NAME	VARCHAR2(30)	NOT NULL	Name of a version-enabled table.
OWNER	VARCHAR2(30)	NOT NULL	Owner (schema) of the table.
NOTIFICATION	VARCHAR2(3)		YES if conflict notification is enabled; NO if conflict notification is not enabled.
NOTIFYWORKSPACES	VARCHAR2(3999)		Names of workspaces where a user needs to be notified if a conflict occurs on the table.
CONFLICT	VARCHAR2(4000)		YES if there are any conflicts on the table between the workspace that performed the SetConflictWorkspace operation and its parent workspace; otherwise, NO.
DIFF	VARCHAR2(4000)		YES if there are any differences for this table as a result of a SetDiffVersions operation; otherwise, NO

3.7 ALL_WORKSPACE_PRIVS

ALL_WORKSPACE_PRIVS contains information about Workspace Manager privileges in all workspaces that the current user can access.

Related View

- USER_WORKSPACE_PRIVS (Section 3.18) contains information about Workspace Manager privileges in workspaces created by the current user.

Column	Datatype	Null?	Description
GRANTEE	VARCHAR2(30)		User or role to which the privilege was granted.
WORKSPACE	VARCHAR2(30)		Name of the workspace.
PRIVILEGE	VARCHAR2(22)		Name of the Workspace Manager privilege.
GRANTOR	VARCHAR2(30)		User or role that granted the privilege.

Column	Datatype	Null?	Description
GRANTABLE	VARCHAR2(3)		YES if grantee was given the grant option (that is, can grant the privilege to other users); NO if grantee was not given the grant option.

3.8 ALL_WORKSPACE_SAVEPOINTS

ALL_WORKSPACE_SAVEPOINTS contains information about savepoints in all workspaces that the current user can access.

Related View

- USER_WORKSPACE_SAVEPOINTS (Section 3.19) contains information about savepoints in workspaces created by the current user.

Column	Datatype	Null?	Description
SAVEPOINT	VARCHAR2(30)	NOT NULL	Name of the savepoint. Explicit savepoints are named by users; implicit savepoints are named by Workspace Manager.
WORKSPACE	VARCHAR2(30)	NOT NULL	Workspace in which the savepoint was created.
IMPLICIT	VARCHAR2(3)		YES if the savepoint is implicit (that is, was created automatically by Workspace Manager); NO if the savepoint is explicit (that is, was created by a user).
POSITION	NUMBER(38)		Position of the savepoint in the sequence in which savepoints were created. In a RollbackToSP operation, if any implicit savepoints have greater position values than the position of the savepoint to be rolled back to, you must first merge or remove the workspaces that caused these intervening implicit savepoints to be created.
OWNER	VARCHAR2(30)		Name of the user that created the savepoint.
CREATETIME	DATE		Date and time that the savepoint was created.
DESCRIPTION	VARCHAR2(1000)		Description of the savepoint.

3.9 ALL_WORKSPACES

ALL_WORKSPACES contains information about all workspaces that the current user can access.

Related View

- USER_WORKSPACES (Section 3.20) contains information about workspaces created by the current user.

Column	Datatype	Null?	Description
WORKSPACE	VARCHAR2(30)		Name of the workspace.
PARENT_WORKSPACE	VARCHAR2(30)		Parent workspace of this workspace.
PARENT_SAVEPOINT	VARCHAR2(30)		Implicit savepoint that was created in the parent workspace when this workspace was created.
OWNER	VARCHAR2(30)		User that created the workspace.
CREATETIME	DATE		Date and time that the workspace was created.
DESCRIPTION	VARCHAR2(1000)		Description of the workspace.
FREEZE_STATUS	VARCHAR2(8)		FROZEN if the workspace is frozen (by a FreezeWorkspace operation); UNFROZEN if the workspace is not frozen.
FREEZE_MODE	VARCHAR2(12)		NO_ACCESS, READ_ONLY, or 1WRITER. See the freezemode parameter description for the FreezeWorkspace procedure in Chapter 2.
FREEZE_WRITER	VARCHAR2(30)		The user allowed to make changes in the workspace; or null if the workspace is not frozen or if it is frozen in NO_ACCESS or READ_ONLY mode. See the freezewriter parameter description for the FreezeWorkspace procedure in Chapter 2.
RESOLVE_STATUS	VARCHAR2(8)		ACTIVE if a conflict resolution session is in progress; INACTIVE if a conflict resolution session is not in progress.
RESOLVE_USER	VARCHAR2(30)		Name of the user that started the conflict resolution session if resolve_status is ACTIVE; otherwise, null.
CONTINUALLY_ REFRESHED	VARCHAR2(3)		YES if the workspace is continually refreshed (see the description of the isrefreshed parameter of the CreateWorkspace procedure); NO if the workspace is not continually refreshed.
WORKSPACE_LOCKMODE	VARCHAR2(9)		EXCLUSIVE if the locking mode is exclusive; SHARED is the locking mode is shared; CARRY if the locking mode is carry-forward. See the lockmode parameter description for the SetWorkspaceLockModeON procedure in Chapter 2.
WORKSPACE_LOCKMODE_ OVERRIDE	VARCHAR2(3)		YES if the override option is TRUE; NO if the override option is FALSE; null if the workspace lock mode is not set. See the override parameter description for the SetWorkspaceLockModeON procedure in Chapter 2.

3.10 DBA_WORKSPACE_SESSIONS

DBA_WORKSPACE_SESSIONS contains information about all users and workspaces (except for the LIVE workspace). This view is only available to users with the WM_ADMIN_ROLE role. It is useful for monitoring users in the different workspaces.

Column	Datatype	Null?	Description
USERNAME	VARCHAR2(30)		User name.
WORKSPACE	VARCHAR2(30)	NOT NULL	Workspace that the user is currently in.
SID	NUMBER		Session ID.
SERIAL#	NUMBER		Internal serial number.
STATUS	VARCHAR2(8)		ACTIVE if the user currently has an open transaction (that is, a regular transaction, as opposed to a "long transaction" in the Workspace Manager sense); INACTIVE if the user does not have an open transaction.

3.11 ROLE_WM_PRIVS

ROLE_WM_PRIVS contains information about privileges that all roles granted to the current user have in each workspace.

Related View

- USER_WM_PRIVS (Section 3.14) contains information about privileges that the current user has in each workspace.

Column	Datatype	Null?	Description
ROLE	VARCHAR2(30)		Name of the role.
WORKSPACE	VARCHAR2(30)		Name of the workspace.
PRIVILEGE	VARCHAR2(22)		Name of the Workspace Manager privilege.
GRANTABLE	VARCHAR2(3)		YES if the role was given the grant option (that is, can grant the privilege to other users); NO if the role was not given the grant option.

3.12 USER_WM_LOCKED_TABLES

USER_WM_LOCKED_TABLES contains information about Workspace Manager locks on rows in version-enabled tables of which the current user is the owner. Its columns are the same as those in ALL_WM_LOCKED_TABLES in Section 3.2.

3.13 USER_WM_MODIFIED_TABLES

USER_WM_MODIFIED_TABLES contains information about version-enabled tables that have been modified and of which the current user is the owner. Its columns are the same as those in ALL_WM_MODIFIED_TABLES in Section 3.3.

3.14 USER_WM_PRIVS

USER_WM_PRIVS contains information about privileges that the current user has in each workspace.

Related View

- ROLE_WM_PRIVS (Section 3.11) contains information about privileges that all roles granted to the current user have in each workspace.

Column	Datatype	Null?	Description
WORKSPACE	VARCHAR2(30)		Name of the workspace.
PRIVILEGE	VARCHAR2(22)		Name of the Workspace Manager privilege.
GRANTOR	VARCHAR2(30)		Name of the user that granted the privilege to the current user.
GRANTABLE	VARCHAR2(3)		YES if the user was given the grant option (that is, can grant the privilege to other users); NO if the user was not given the grant option.

3.15 USER_WM_RIC_INFO

USER_WM_RIC_INFO contains information about referential integrity constraints in version-enabled tables of which the current user is the owner. Its columns are the same as those in ALL_WM_RIC_INFO in Section 3.4.

Workspace Manager uses this information to provide referential integrity support, which is described in Section 1.6.

3.16 USER_WM_TAB_TRIGGERS

USER_WM_TAB_TRIGGERS contains information about triggers that are owned by the current user and that are on version-enabled tables. Its columns are the same as those in ALL_WM_TAB_TRIGGERS in Section 3.5, except that it does not contain the TRIGGER_OWNER column.

3.17 USER_WM_VERSIONED_TABLES

USER_WM_VERSIONED_TABLES contains information about version-enabled tables of which the current user is the owner. Its columns are the same as those in ALL_WM_VERSIONED_TABLES in Section 3.6.

3.18 USER_WORKSPACE_PRIVS

USER_WORKSPACE_PRIVS contains information about Workspace Manager privileges in workspaces created by the current user. Its columns are the same as those in ALL_WORKSPACE_PRIVS in Section 3.7.

3.19 USER_WORKSPACE_SAVEPOINTS

USER_WORKSPACE_SAVEPOINTS contains information about savepoints in workspaces created by the current user. Its columns are the same as those in ALL_WORKSPACE_SAVEPOINTS in Section 3.8.

3.20 USER_WORKSPACES

USER_WORKSPACES contains information about workspaces created by the current user. Its columns are the same as those in ALL_WORKSPACES in Section 3.9.

3.21 xxx_CONF Views

There is one conflict view per version-enabled table. Each conflict view has a name in the form <table_name>_CONF. For example, if the EMPLOYEE table is version-enabled, the EMPLOYEE_CONF metadata view exists.

Each conflict view contains the columns shown in Table 3–1.

Table 3–1 Columns in the xxx_CONF Views

Column	Datatype	Description
WM_WORKSPACE	VARCHAR2(256)	Workspace in which the conflict exists.
(One column for each column in original table)	(Same as column in original table)	Value of the column in this workspace.
WM_DELETED	VARCHAR2(3)	YES if the row has been deleted; NO if the row has not been deleted; NE if the row is nonexistent (does not exist).

The following example lists the key value and all column values of conflicting rows in the EMPLOYEE table in the current workspace and the parent workspace. This view is available after the SetConflictWorkspace procedure has been called specifying the child workspace (the current workspace in this case).

```
SELECT * FROM EMPLOYEE_CONF;
```

If ID, NAME, and CITY are the columns in the EMPLOYEE table, then assume the following values:

WM_WORKSPACE	ID	NAME	CITY	WM_DEL
NEWWORKSPACE	12	SMITH	NASHUA	NO
DiffBase	12	SMITH	NY	NO
LIVE	12	SMITH	BOSTON	NO

The database row identified by ID = 12 was changed in NEWWORKSPACE and LIVE workspaces. In NEWWORKSPACE the city was changed to NASHUA, and in the LIVE workspace the city was changed to BOSTON. When NEWWORKSPACE is merged into LIVE, this row will show up as a conflict. The application must pick between the choices and resolve conflicts in favor of the workspace with the desired value.

The following example begins a conflict resolution session, calls the ResolveConflicts procedure to delete the conflicting row from the NEWWORKSPACE workspace and to insert the value in the parent workspace (LIVE) into both workspaces, commits the transaction, and ends the conflict resolution session.

```
DBMS_WM.BeginResolve ('NEWWORKSPACE');
DBMS_WM.ResolveConflicts ('NEWWORKSPACE', 'EMPLOYEE', 'ID = 12', 'PARENT');
COMMIT;
DBMS_WM.CommitResolve ('NEWWORKSPACE');
```

For additional information about conflict resolution, see Section 1.2.5.

3.22 xxx_DIFF Views

There is one difference view for each version-enabled table. Each difference view has a name in the form <table_name>_DIFF. For example, if the EMPLOYEE table is version-enabled, the EMPLOYEE_DIFF metadata view exists. Rows are added to one or more xxx_DIFF views each time the SetDiffVersions procedure is executed.

Each difference view contains the columns shown in Table 3–2.

Table 3–2 Columns in the xxx_DIFF Views

Column	Datatype	Description
(One column for each column in original table)	(Same as column in original table)	Value of the column in this workspace.
WM_DIFFVER	VARCHAR2(256)	Branch from which the values in the preceding columns are taken. (See the explanation following this table.)
WM_CODE	VARCHAR2(4000)	One of the following codes describing the change: U (updated), D (deleted), I (inserted), NC (no change), NE (nonexistent).

The WM_DIFFVER value is in one of the following formats:

- '<workspace1>, <savepoint1>'
- '<workspace2>, <savepoint2>'
- 'DiffBase'

If the two-parameter version of the SetDiffVersions procedure was used, the value of savepoint1 or savepoint2 is LATEST.

Note the following about the possible values for WM_CODE:

- NC will appear for rows in workspaces that did not change the value when another workspace did change the value. For example, if '<workspace2>, <savepoint2>' updated the row, the code for that row is U, but the code for the '<workspace1>, <savepoint1>' and 'DiffBase' rows is NC if they did not modify the row.
- NE will appear for 'DiffBase' if a row is inserted in one or more branches, and NE will appear for 'DiffBase' and a branch if only one branch has had any insert operations.

For more information, including an example showing rows being added to a differences view, see the section on the SetDiffVersions procedure in Chapter 2.

3.23 xxx_LOCK Views

There is one lock view per version-enabled table. Each lock view has a name in the form <table_name>_LOCK. For example, if the EMPLOYEE table is version-enabled, the EMPLOYEE_LOCK metadata view exists.

Each lock view contains the columns shown in Table 3–3.

Table 3–3 Columns in the xxx_LOCK Views

Column	Datatype	Description
(One column for each column in original table)	(Same as column in original table)	Value of the column in this workspace.
WM_LOCKMODE	VARCHAR2(9)	Type of lock: E (exclusive) or S (shared).
WM_USERNAME	VARCHAR2(4000)	User name of the owner of the lock.
WM_LOCKINGWORKSPACE	VARCHAR2(4000)	Name of the workspace in which the lock was placed.
WM_INCURWORKSPACE	VARCHAR2(3)	Contains YES if the row is contained in the current workspace, and NO if the row is not contained in the current workspace.

3.24 xxx_HIST Views

There is one history view per version-enabled table if the table was version-enabled with the hist parameter set to VIEW_W_OVERWRITE or VIEW_WO_OVERWRITE in the call to the EnableVersioning procedure. Each history view has a name in the form <table_name>_HIST. For example, if the EMPLOYEE table is version-enabled with the hist parameter set to VIEW_W_OVERWRITE or VIEW_WO_OVERWRITE, the EMPLOYEE_HIST metadata view exists.

You can use the history views to log and audit modifications to version-enabled tables.

Each history view contains the columns shown in Table 3–4.

Table 3–4 Columns in the xxx_HIST Views

Column	Datatype	Description
(One column for each column in original table)	(Same as column in original table)	Value of the column in this workspace.
WORKSPACE	VARCHAR2(30)	Name of the workspace containing the row.

Table 3–4 Columns in the xxx_HIST Views (Cont.)

Column	Datatype	Description
VERSION	INTEGER	Version number of the row with which the data is associated.
USER_NAME	VARCHAR2(4000)	Name of the user that created the row.
TYPE_OF_CHANGE	VARCHAR2(1)	Type of change operation that was performed on the row: D (delete), I (insert), or U (update).
CREATETIME	DATE	Time when the row was created or updated.
RETIRETIME	DATE	Time when the row was deleted or modified.

3.25 xxx_MW Views

There is one multiworkspace view per version-enabled table. Each multiworkspace view has a name in the form <table_name>_MW. For example, if the EMPLOYEE table is version-enabled, the EMPLOYEE_MW metadata view exists. Rows are added to one or more xxx_MW views each time the SetMultiWorkspaces procedure (described in Chapter 2) is executed.

Each multiworkspace view contains the columns shown in Table 3–5.

Table 3–5 Columns in the xxx_MW Views

Column	Datatype	Description
(One column for each column in original table)	(Same as column in original table)	Value of the column in this workspace.
MODIFIED_BY	VARCHAR2(4000)	Workspace containing the row that was modified.
OPTYPE	VARCHAR2(4000)	One of the following codes describing the change: U (updated), D (deleted), I (inserted).

You can use the <table_name>_MW view to see changes in another workspace without leaving the current workspace (for example, to check if there is a conflict with the other workspace). Each row in the view shows the data as it would be in that workspace if the workspace had been merged when the row was inserted in the view.

You can also use the <table_name>_DIFF view (see Section 3.22) to see changes in another workspace without leaving the current workspace; however, the <table_

name>_DIFF view can be used for only two workspaces, whereas the <table_ name>_MW view can be used for any number of workspaces.

A

Installing Workspace Manager with Custom Databases

Workspace Manager is installed by default in the seed database and in all databases created by the Database Configuration Assistant (DBCA). However, in all other Oracle databases, such as those you create with a customized procedure, you must install Workspace Manager before you can use its features.

To install Workspace Manager in a custom database, do the following:

1. At the system command prompt, change the current directory to the directory that contains Workspace Manager installation script and packages, as shown in the following example:

   ```
   cd <ORACLE_HOME>/rdbms/admin
   ```

2. Connect as SYS to the Oracle9*i* instance with a command in the following format:

   ```
   sqlplus sys/<sys-password>
   ```

3. Run the `owminst.plb` script:

   ```
   SQL> @owminst.plb
   ```

4. Verify the installation of Workspace Manager by entering the following command while connected as any valid database user, and ensure that the output is as shown here:

   ```
   SQL> select dbms_wm.getWorkspace from dual;

   GETWORKSPACE
   --------------------------------------------------------------------------------
   LIVE
   ```

B

Error Messages

This appendix lists the Workspace Manager error messages, including the cause and recommended user action for each.

WM_ERROR_1 name of column *'string'* has more than 28 characters

Cause: An attempt was made to version-enable a table that had a column with a name that has more than 28 characters.

Action: Ensure that all column names for the table are 28 characters or less.

WM_ERROR_2 *'string'* is not allowed for workspace: *'string'* frozen in *'string'* mode

Cause: An operation was executed on a workspace that was frozen.

Action: Unfreeze the workspace before retrying the operation.

WM_ERROR_3 cannot modify primary key values for version-enabled table

Cause: A DML operation which modifies one or more values in columns in the primary key constraint was performed on a version-enabled table.

Action: Do not perform DML operations on columns in the primary key constraints of version-enabled tables.

WM_ERROR_4 There are open short transactions on this table.

Cause: DisableVersioning failed because there were open short transactions on the table to be version-disabled.

Action: The user with the open short transaction should issue a standard database commit or rollback.

WM_ERROR_5 integrity constraint (*'string'.'string'*) violated - child record found

Cause: An attempt was made to delete/update a record in a parent table of a referential integrity constraint with restrict option and there was a matching

record in the child table of the integrity constraint. Restrict is a default property of a referential integrity constraint, the other being "on delete cascade", where the dependent rows in the child tables are deleted if corresponding rows in the parent table are deleted. The "cascade" option applies only to a delete from the parent table. An update of the parent table always follows the restrict option.

Action: Delete all matching records from the child table first.

WM_ERROR_6 integrity constraint (*'string'.'string'*) violated - parent key not found

Cause: An attempt was made to insert/update a record in a child table of a referential integrity constraint and there was no matching record in the parent table of the integrity constraint.

Action: Insert a matching record in the parent table first.

WM_ERROR_7 WM not found on the import platform

Cause: Import of a version-enabled database failed because the import platform did not have Workspace Manager installed.

Action: Install Workspace Manager on the import platform and retry.

WM_ERROR_8 the import platform cannot have any versioned tables

Cause: Import of a version-enabled database failed because the import platform already had one or more version-enabled tables.

Action: The import platform may not have any version-enabled tables. A clean install of Workspace Manager is needed on the import platform.

WM_ERROR_9 the import platform has non-"LIVE" workspaces or explicit savepoints

Cause: Import of a version-enabled database failed because the import platform had either non-LIVE workspaces in the workspace hierarchy, or explicit savepoints in the LIVE workspace.

Action: The import platform may have only the LIVE workspace and there may be no explicit savepoints. A clean install of Workspace Manager is needed on the import platform.

WM_ERROR_10 unique key violation

Cause: An insert operation failed because it violated the table's primary key constraint.

Action: Ensure that the primary key is not violated by the insert operation in the current workspace.

WM_ERROR_11 need to be on the latest version to delete

Cause: A delete operation failed because the delete was being made in a non-latest version of a workspace.

Action: Ensure that the current session is on the latest version in the workspace by using the GotoWorkspace or GotoSavepoint procedures.

WM_ERROR_12 need to be on the latest version to insert

Cause: An insert operation failed because the insert was being made in a non-latest version of a workspace.

Action: Ensure that the current session is on the latest version in the workspace by using the GotoWorkspace or GotoSavepoint procedures.

WM_ERROR_13 need to be on the latest version to update

Cause: An update operation failed because the update was made in a non-latest version of a workspace.

Action: Ensure that the current session is on the latest version in the workspace by using the GotoWorkspace or GotoSavepoint procedures.

WM_ERROR_14 *'string'.'string'* has not been version enabled

Cause: This operation failed because it can only be invoked on a version-enabled table.

Action: Verify that the table is version-enabled.

WM_ERROR_15 "/" is not allowed in a workspace name

Cause: CreateWorkspace failed because the workspace name contained a "/".

Action: Choose another workspace name that does not contain a "/".

WM_ERROR_16 "WM_ADMIN_ROLE" is required to version disable a table in another schema

Cause: DisableVersioning failed because only a user with `WM_ADMIN_ROLE` role can version-disable a table in another schema.

Action: Ensure that the invoking user has the required privileges before attempting to version-disable this table. Otherwise, have the owner of the table version-disable it.

WM_ERROR_17 "WM_ADMIN_ROLE" is required to version enable a table in another schema

Cause: EnableVersioning failed because only a user with `WM_ADMIN_ROLE` can version-enable a table in another schema.

Action: Ensure that the invoking user has the required privileges before attempting to version-enable this table. Otherwise, have the owner of the table version-enable it.

WM_ERROR_18 "WM_ADMIN_ROLE" or ownership is required to alter workspace attributes

Cause: AlterWorkspace failed because only a user with `WM_ADMIN_ROLE` or the owner of the workspace can alter workspace attributes.

Action: Ensure that the invoking user has the required privileges before attempting to alter the workspace attributes. Otherwise, have the owner of the workspace alter the workspace attributes.

WM_ERROR_19 "WM_ADMIN_ROLE" or ownership is required to freeze a workspace

Cause: FreezeWorkspace failed because only a user with `WM_ADMIN_ROLE` or the owner of the workspace can freeze a workspace.

Action: Ensure that the invoking user has the required privileges before attempting to freeze the workspace. Otherwise, have the owner of the workspace freeze it.

WM_ERROR_20 "WM_ADMIN_ROLE" or ownership is required to set workspace lock mode

Cause: SetWorkspaceLockModeOn failed because only a user with `WM_ADMIN_ROLE` role or the owner of the workspace can set the workspace lock mode.

Action: Ensure that the invoking user has the required privileges before attempting to set the workspace lock mode. Otherwise, have the owner of the workspace set the workspace lock mode.

WM_ERROR_21 insufficient privileges to change savepoint attributes

Cause: AlterSavepoint failed because only a user with `WM_ADMIN_ROLE` role or the owner of the workspace or savepoint can alter the savepoint attributes.

Action: Ensure that the invoking user has the required privileges before attempting to alter the savepoint attributes. Otherwise, have the workspace owner or the savepoint owner alter the savepoint attributes.

WM_ERROR_22 insufficient privileges to delete savepoint

Cause: DeleteSavepoint failed because only a user with `WM_ADMIN_ROLE` role or the owner of the workspace or savepoint can delete the savepoint.

Action: Ensure that the invoking user has the required privileges before attempting to delete the savepoint. Otherwise, have the workspace owner or the savepoint owner delete the savepoint.

WM_ERROR_23 a workspace already exists with the name: *'string'*

Cause: CreateWorkspace failed because a workspace with the same name already existed in the system. Workspace Manager requires that workspace names be unique across the database.

Action: Choose another workspace name and retry.

WM_ERROR_24 a workspace cannot be rolled back over an implicit savepoint

Cause: A RollbackWorkspace operation was invoked on a non-leaf workspace across an implicit savepoint.

Action: Do not rollback over an implicit savepoint. To remove the implicit savepoint, merge or remove the descendant workspace.

WM_ERROR_25 a table cannot be merged from the "LIVE" workspace

Cause: MergeTable was invoked with the input workspace specified as the `LIVE` workspace. The `LIVE` workspace is the root workspace in the workspace hierarchy tree.

Action: Do not invoke MergeTable with the workspace parameter `LIVE`.

WM_ERROR_27 a table cannot be refreshed to the "LIVE" workspace

Cause: RefreshTable was invoked with the input workspace specified as the `LIVE` workspace. The `LIVE` workspace is the root workspace in the workspace hierarchy tree.

Action: Do not invoke RefreshTable with the workspace parameter `LIVE`.

WM_ERROR_28 a table cannot be rolled back over an implicit savepoint

Cause: A RollbackTable operation was invoked on a non-leaf workspace across an implicit savepoint.

Action: Do not rollback over an implicit savepoint. To remove the implicit savepoint, merge or remove the descendant workspace.

WM_ERROR_29 cannot rollback this table using RollbackTable

Cause: RollbackTable failed because the table to be rolled back is part of a referential integrity constraint.

Action: Use RollbackWorkspace or RollbackToSP instead.

WM_ERROR_30 cannot merge this table using MergeTable

Cause: MergeTable failed because the table to be merged is part of a referential integrity constraint.

Action: Use MergeWorkspace instead.

WM_ERROR_31 All version enabled tables owned by *'string'* must be disabled first.

Cause: An attempt was made to drop a database user who owns one or more version-enabled tables.

Action: Ensure that all the version-enabled tables owned by the user have been explicitly disabled before attempting to drop the database user.

WM_ERROR_32 An index-organized table cannot be version enabled.

Cause: Workspace Manager does not support index-organized tables.

Action: Ensure the table to be version-enabled is not index-organized.

WM_ERROR_33 attempt to *'string'* a row locked by: *'string'* in workspace *'string'*

Cause: A DML operation failed because the row was previously locked.

Action: Wait for the lock on the row to be released or have the lock owner use the UnlockRows operation to unlock the row. Consult the table's _LOCK view to see which rows in this table are locked.

WM_ERROR_34 attempt to *'string'* a row locked by *'string'* in workspace: *'string'*

Cause: A DML operation failed because the row was previously locked.

Action: Wait for the lock on the row to be released or have the lock owner use the UnlockRows operation to unlock the row. Consult the table's _LOCK view to see which rows in this table are locked.

WM_ERROR_35 attempt to lock a row locked in workspace: *'string'*

Cause: The operation failed because a lock could not be obtained on the row, since it was already locked.

Action: Wait for the lock on the row to be released or have the lock owner use the UnlockRows operation to unlock the row. Consult the table's _LOCK view to see which rows in this table are locked.

WM_ERROR_36 attempt to lock a row locked by *'string'*

Cause: The operation failed because a lock could not be obtained on the row, since it was already locked.

Action: Wait for the lock on the row to be released or have the lock owner use the UnlockRows operation to unlock the row. Consult the table's _LOCK view to see which rows in this table are locked.

WM_ERROR_37 attempt to modify a WM generated procedure

Cause: An attempt to drop or re-create a database procedure failed because that procedure was created by Workspace Manager.

Action: Do not drop or re-create this procedure.

WM_ERROR_38 cannot disable version a table modified in non-LIVE workspaces

Cause: DisableVersioning failed because the table had been modified in non-LIVE workspaces.

Action: Remove or merge all workspaces that have modified this table. Otherwise, use the `FORCE` option of DisableVersioning.

WM_ERROR_39 cannot drop tables involved in foreign key relationships

Cause: An attempt to drop a database table failed because it was involved in a foreign key relationship with a version-enabled table.

Action: Consult the `WM_RIC_INFO` view and version-disable the table that is involved in the foreign key relationship before attempting to drop the table.

WM_ERROR_40 only grantor of a privilege may revoke it

Cause: An attempt was made to revoke a privilege that was not granted by the current user.

Action: Do not attempt to revoke this privilege.

WM_ERROR_41 unable to set workspace lock mode

Cause: SetWorkspaceLockModeOn failed because the workspace contained modifications from one or more version-enabled tables.

Action: Use SetLockingOn to set the session's lock mode. Use SetWorkspaceLockModeOn only for those workspaces that have not yet modified any version-enabled tables.

WM_ERROR_42 cannot version enable tables owned by SYS

Cause: EnableVersioning failed because Workspace Manager can only version-enable tables owned by users other than SYS.

Action: Do not invoke EnableVersioning on tables owned by SYS.

WM_ERROR_43 A continually refreshed workspace must be a leaf workspace.

Cause: CreateWorkspace failed because the workspace to be created was to be a child of a continually refreshed workspace. Continually refreshed workspaces carry with them the restriction that they must be leaf workspaces.

Action: Do not create a workspace off of a continually refreshed workspace.

WM_ERROR_44 merge operation requires ACCESS and MERGE privileges on the workspace

Cause: The operation invoked failed because it required both ACCESS and MERGE privileges on the workspace on which it was invoked.

Action: Use the function GetPrivs to ensure that the user invoking this operation has the required privileges on the workspace.

WM_ERROR_45 merge operation requires ACCESS privileges on the parent workspace

Cause: The operation invoked failed because it required ACCESS privileges on the parent workspace of the workspace it was invoked on.

Action: Use the function GetPrivs to ensure that the user invoking this operation has the required privileges on the parent workspace.

WM_ERROR_46 commit/rollback open short transactions before calling CommitResolve

Cause: CommitResolve failed because open short transactions existed.

Action: The user with the open short transaction should issue a standard database commit or rollback.

WM_ERROR_47 commit/rollback open short transactions before calling CompressWorkspace

Cause: CompressWorkspace failed because open short transactions existed.

Action: The user with the open short transaction should issue a standard database commit or rollback.

WM_ERROR_48 commit/rollback open short transactions before calling CompressWorkspaceTree

Cause: CompressWorkspaceTree failed because open short transactions existed.

Action: The user with the open short transaction should issue a standard database commit or rollback.

WM_ERROR_49 commit/rollback open short transactions before calling DeleteSavepoint

Cause: DeleteSavepoint failed because open short transactions existed.

Action: The user with the open short transaction should issue a standard data

WM_ERROR_50 commit/rollback open short transactions before calling GotoWorkspace

Cause: GotoWorkspace failed because open short transactions existed.

Action: The user with the open short transaction should issue a standard database commit or rollback.

WM_ERROR_51 commit/rollback open short transactions before calling RollbackResolve

Cause: RollbackResolve failed because open short transactions existed.

Action: The user with the open short transaction should issue a standard database commit or rollback.

WM_ERROR_52 CommitResolve can be called only after BeginResolve has been invoked

Cause: CommitResolve failed because BeginResolve was not previously invoked.

Action: To resolve conflicts, first issue a BeginResolve, then issue ResolveConflicts, and finally issue CommitResolve.

WM_ERROR_53 CompressWorkspace operation requires ACCESS and MERGE privileges on the workspace

Cause: The operation invoked failed because it required both ACCESS and MERGE privileges on the workspace on which it was invoked.

Action: Use the function GetPrivs to ensure that the user invoking this operation has the required privileges on the workspace.

WM_ERROR_54 CompressWorkspace operation requires ACCESS privilege on the workspace

Cause: The operation invoked failed because it required ACCESS privileges on the workspace on which it was invoked.

Action: Use the function GetPrivs to ensure that the user invoking this operation has the required privileges on the workspace.

WM_ERROR_55 conflicts detected for workspace: *'string'* in table: *'string'*

Cause: An operation failed because there were conflicts detected for the table.

Action: To resolve conflicts, first issue a BeginResolve, then issue ResolveConflicts, and finally issue CommitResolve. Otherwise, refrain from calling this operation.

WM_ERROR_56 conflicts detected for workspace: *'string'* in table: *'string'.'string'*

Cause: An operation failed because there were conflicts detected for the table.

Action: To resolve conflicts, first issue a BeginResolve, then issue ResolveConflicts, and finally issue CommitResolve. Otherwise, refrain from calling this operation.

WM_ERROR_57 CreateSavepoint operation requires ACCESS privileges on the workspace

Cause: The operation invoked failed because it required ACCESS privileges on the workspace on which it was invoked.

Action: Use the function GetPrivs to ensure that the user invoking this operation has the required privileges on the workspace.

WM_ERROR_58 RemoveWorkspace operation requires ACCESS and REMOVE privileges on the workspace

Cause: The operation invoked failed because it required both ACCESS and REMOVE privileges on the workspace on which it was invoked.

Action: Use the function GetPrivs to ensure that the user invoking this operation has the required privileges on the workspace.

WM_ERROR_59 entry already exists in spatial metadata table for *'string'*_WM

Cause: EnableVersioning of the spatial table failed because the spatial metadata table already contained an entry for the table.

Action: Contact Oracle Support

WM_ERROR_60 user must call BeginResolve or have WM_ADMIN_ROLE to invoke RollbackResolve

Cause: RollbackResolve can be successful only if the user invoking it also invoked BeginResolve, or if the user invoking it had the WM_ADMIN_ROLE role.

Action: Ensure that the invoking user has the required privileges before attempting to invoke RollbackResolve. Otherwise, have the user that issued the BeginResolve operation invoke RollbackResolve.

WM_ERROR_61 versioned objects have to be version disabled before being dropped

Cause: An attempt to drop a database table or view failed because it was associated with a version-enabled table.

Action: version-disable the table first. In the case of a view, version-disable the table associated with the view.

WM_ERROR_62 versioned table: *'string'* does not exist

Cause: The operation failed because the table passed in as input did not exist or was not version-enabled.

Action: Pass in an existing, version-enabled table as input.

WM_ERROR_63 need to be on the latest version to create a continually refreshed workspace.

Cause: CreateWorkspace failed because the session was in a non-latest version of the workspace.

Action: Ensure that the current session is on the latest version in the workspace by using the GotoWorkspace or GotoSavepoint procedures.

WM_ERROR_64 need to be on the latest version to create a savepoint

Cause: CreateSavepoint failed because the session was in a non-latest version of the workspace.

Action: Ensure that the current session is on the latest version in the workspace by using the GotoWorkspace or GotoSavepoint procedures.

WM_ERROR_65 grantor and grantee may not be the same user

Cause: An attempt was made to grant or revoke a privilege from / to the same user.

Action: Do not attempt to grant or revoke privileges from / to the same user. Privileges can only be granted or revoked between different users.

WM_ERROR_66 unable to version enable this table with history option

Cause: An attempt was made to version-enable a table with `VIEW_WO_OVERWRITE` or `VIEW_W_OVERWRITE` option and the cumulative length of the names of the primary key columns was greater than 600.

Action: Rename the primary key columns.

WM_ERROR_67 grantee must be an existing user, an existing role or PUBLIC

Cause: A grant operation was attempted with an invalid grantee parameter.

Action: The grantee may only be an existing user, role, or PUBLIC. Verify correct spelling of the grantee parameter.

WM_ERROR_68 input parameter grant_option must be "YES" or "NO"

Cause: An attempt was made to invoke a GrantWorkspacePriv or GrantSystemPriv operation with an invalid input parameter.

Action: Ensure that the valid parameters are passed to the GrantWorkspacePriv or GrantSystemPriv operation. The grant_option parameter may only be YES or NO.

WM_ERROR_69 invalid in_date time for GotoDate

Cause: GotoDate was invoked with an in_date time less than the create time of the current workspace.

Action: The in_date parameter for GotoDate must be greater than or equal to the create time for the current workspace.

WM_ERROR_70 insufficient privileges on *'string'* to lock rows

Cause: An attempt was made to invoke a LockRows operation on a versioned table without the required privileges on the table.

Action: Ensure that the invoking user has the required privileges before invoking the operation. A lockRows operation requires the invoking user to have SELECT, INSERT, UPDATE and DELETE privileges on the versioned table.

WM_ERROR_71 insufficient privileges on *'string'* to unlock rows

Cause: An attempt was made to invoke an UnlockRows operation on a versioned table without the required privileges on the table.

Action: Ensure that the invoking user has the required privileges before invoking the operation. An UnlockRows operation requires the invoking user to have SELECT, INSERT, UPDATE and DELETE privileges on the versioned table.

WM_ERROR_72 insufficient privileges on *'string'.'string'*

Cause: An attempt was made to invoke a ResolveConflicts operation on a versioned table without the required privileges on the table.

Action: Ensure that the invoking user has the required privileges before invoking the operation. A ResolveConflicts operation requires the invoking user to have SELECT, INSERT, UPDATE and DELETE privileges on the versioned table being conflict resolved.

WM_ERROR_73 insufficient privileges to ACCESS the workspace: *'string'*

Cause: An attempt was made to invoke an operation that required the specified privileges on the input workspace.

Action: Ensure that the invoking user has the required privileges before invoking the operation. Privileges can be granted using the GrantWorkspacePriv or the GrantSystemPriv procedures. Use the function GetPrivs to see which privileges you have on a workspace.

WM_ERROR_74 insufficient privileges to ACCESS the parent workspace: *'string'*

Cause: An attempt was made to invoke an operation that required the specified privileges on the parent workspace of the input workspace.

Action: Ensure that the invoking user has the required privileges before invoking the operation. Privileges can be granted using the grantWorkspacePriv or the grantSystemPriv procedures. Use the function GetPrivs to see which privileges you have on a workspace.

WM_ERROR_75 insufficient privileges to create a child workspace of: *'string'*

Cause: An attempt was made to invoke a CreateWorkspace operation from a workspace without the required privileges on the workspace.

Action: Ensure that the invoking user has the required privileges before invoking the operation. The invoking user must have CREATE privileges on a workspace to be allowed to create a workspace off of it. Privileges can be granted using the grantWorkspacePriv or the grantSystemPriv procedures. Use the function GetPrivs to see which privileges you have on a workspace.

WM_ERROR_76 insufficient privileges to grant *'string'*

Cause: An attempt was made to invoke the GrantWorkspacePriv or GrantSystemPriv operation without the required privileges to do so.

Action: Ensure that the invoking user has the required privileges to grant the privilege. A user needs to have been granted a privilege with the grant option to be able to grant it to others.

WM_ERROR_77 insufficient privileges on the versioned table *'string'*

Cause: An attempt was made to invoke a Workspace Manager operation without the required privileges on the versioned table.

Action: Ensure that the invoking user has the required privileges before invoking the operation. All Workspace Manager workspace wide operations require the invoking user to have SELECT, INSERT, UPDATE and DELETE privileges on all versioned tables that were modified in the input workspace.

WM_ERROR_78 insufficient privileges on the versioned table: *'string'.'string'*

Cause: An attempt was made to invoke a Workspace Manager operation without the required privileges on the versioned table.

Action: Ensure that the invoking user has the required privileges before invoking the operation. All Workspace Manager workspace wide operations require the invoking user to have SELECT, INSERT, UPDATE and DELETE privileges on all versioned tables that were modified in the input workspace.

WM_ERROR_79 WM internal error ['*string*']

Cause: An Workspace Manager operation resulted in an internal error.

Action: Contact Oracle support to resolve the issue.

WM_ERROR_80 invalid "hist" parameter for EnableVersioning

Cause: An invalid value was specified for the hist parameter of procedure EnableVersioning.

Action: Valid values for the hist parameter are NONE, VIEW_W_OVERWRITE, and VIEW_WO_OVERWRITE.

WM_ERROR_81 invalid column name specified in the where-clause

Cause: An attempt was made to invoke a Workspace Manager operation with an invalid where-clause parameter as input.

Action: Ensure that the input where-clause contains only valid column names and has proper syntax.

WM_ERROR_82 invalid privilege type: *'string'* was specified as input

Cause: An attempt was made to invoke a Grant/Revoke Privilege operation with an invalid priv_type parameter.

Action: Ensure that the valid parameters are passed to the Grant/Revoke Privilege operation. The valid privilege types are: ACCESS_WORKSPACE, MERGE_WORKSPACE, ROLLBACK_WORKSPACE, REVOKE_WORKSPACE, and CREATE_WORKSPACE.

WM_ERROR_83 invalid user specified for the freezewriter parameter

Cause: The FreezeWorkspace procedure was called with an invalid freezewriter parameter.

Action: Ensure that the freezewriter parameter passed in as input to the FreezeWorkspace procedure is an existing database user.

WM_ERROR_84 invalid value for lock_mode - "E" or "S" expected

Cause: An invalid value was specified for the lock_mode parameter of procedure LockRows.

Action: Specify a valid value for lock_mode. The valid values for lock_mode are E and S (default is E).

WM_ERROR_85 invalid value for the lock_mode argument - "E", "S" or "ES" expected

Cause: An invalid value has been specified for the lock_mode parameter (fifth parameter) of procedure UnlockRows.

Action: Specify a valid value for lock_mode. The valid values for lock_mode are E, S, and ES (default is ES).

WM_ERROR_86 invalid value for the all_or_user argument - "ALL" or "USER" expected

Cause: An invalid value has been specified for the all_or_user parameter (fourth parameter) of procedure UnlockRows.

Action: Specify a valid value for all_or_user. The valid values for all_or_user are ALL and USER (default is USER).

WM_ERROR_87 IsWorkspaceOccupied cannot be used for "LIVE" workspace

Cause: A user attempted to invoke IsWorkspaceOccupied on the LIVE workspace.

Action: Workspace Manager allows IsWorkspaceOccupied to be invoked only on workspaces other than LIVE. The LIVE workspace is the default workspace for any session that is connected and Workspace Manager does not monitor users in the LIVE workspace. Do not invoke this method on the LIVE workspace.

WM_ERROR_88 IsWorkspaceOccupied requires ACCESS privilege on the workspace

Cause: IsWorkspaceOccupied was invoked for a workspace on which the user did not have ACCESS privilege.

Action: IsWorkspaceOccupied can only be invoked for a workspace on which the user has ACCESS privilege.

WM_ERROR_89 "LIVE" workspace can be frozen only in (READ_ONLY, 1WRITER, WM_ONLY) modes

Cause: An attempt was made to Freeze the LIVE workspace in NO_ACCESS mode. Workspace Manager does not support this mode for the LIVE workspace.

Action: Use one of (READ_ONLY, 1WRITER, WM_ONLY) modes to freeze the LIVE workspace.

WM_ERROR_90 lock operation requires ACCESS privilege on the parent workspace

Cause: LockRows was invoked for a workspace whose parent workspace was not accessible to the user.

Action: The user requires ACCESS privilege on the parent workspace of the workspace for which lockRows in invoked.

WM_ERROR_91 lock operation requires ACCESS privilege on the workspace

Cause: LockRows was invoked for a workspace on which the user did not have ACCESS privilege.

Action: The user requires ACCESS privilege on the workspace for which Lock-Rows in invoked.

WM_ERROR_92 cannot *'string'* because locking is on and row is already versioned

Cause: An attempt to place a shared / exclusive lock on a row in a versioned table failed because the row was already versioned in some other workspace.

Action: To update / delete / insert a row that was already versioned in some other workspace, the current session must turn locking off. Consult the table's _ LOCK view to see which rows in this table are locked.

WM_ERROR_93 The multi-workspace view requires ACCESS privilege on the workspace : *'string'*

Cause: SetMultiWorkspaces was invoked with the name of a workspace on which the user did not have ACCESS privilege.

Action: Names of only those workspaces for which the user has ACCESS privilege can be passed to SetMultiWorkspaces.

WM_ERROR_94 non-existent versioned table: *'string'.'string'*

Cause: This operation was invoked on a non-version-enabled table.

Action: This operation can only be invoked on a version-enabled table. Verify that the table is version-enabled. The xxx_VERSIONED_TABLES views show all the versioned tables in the database.

WM_ERROR_95 null savepoint name passed in as input

Cause: An attempt was made to invoke on Workspace Manager operation with a null savepoint name parameter.

Action: User must pass in a non-null savepoint parameter for this operation to succeed

WM_ERROR_96 null workspace name passed in as input

Cause: A null value was passed in as input to a Workspace Manager operation

Action: A user must pass in a non-null workspace parameter for this operation to succeed

WM_ERROR_97 null table name parameter passed in

Cause: MergeTable was invoked with a null table name.

Action: Specify name of the version-enabled table to be merged.

WM_ERROR_98 Number of workspaces in the multi-workspace view cannot be greater than 8.

Cause: SetMultiWorkspaces was invoked with more than 8 workspace names.

Action: Invoke SetMultiWorkspaces with 8 or fewer workspace names.

WM_ERROR_99 WM failed to install - system triggers not properly created

Cause: One of the Workspace Manager generated database triggers was not created properly.

Action: Contact Oracle support to resolve the issue.

WM_ERROR_100 *'string'* is both parent and child tables of referential integrity constraints

Cause: An attempt was made to version-enable a table that was both parent and child tables of referential integrity constraints. Workspace Manager does not support (except for self referential integrity constraints) such transitive referential integrity constraints.

Action: Those referential integrity constraints (except for self referential integrity constraints) for which the table to be version-enabled is the child table should be dropped before version-enabling it.

WM_ERROR_101 child table must be version enabled

Cause: An attempt was made to version-enable the parent table of a referential integrity constraint whose child table was not version-enabled.

Action: Before version-enabling a table T, all tables that are child tables of referential integrity constraints (excluding self referential integrity constraints) that have T as the parent table, must be version-enabled.

WM_ERROR_102 cannot version enable this table

Cause: An attempt was made to version-enable a table which was the child table of a non-self referential integrity constraint with cascade option and which had a self referential integrity constraint defined on it.

Action: If application semantics permit, change the cascade option to the restrict option.

WM_ERROR_103 cannot version disable this table with force option

Cause: Force option was specified while version-disabling a table which was the parent table of a referential integrity constraint.

Action: Force option cannot be specified while version-disabling a table which is the parent table of a referential integrity constraint. Commit/rollback all changes done on this table in non-LIVE workspaces and then version-disable the table without the force option.

WM_ERROR_104 cannot version disable this table

Cause: An attempt has been made to version-disable the child table of a referential integrity constraint whose parent table was still version-enabled.

Action: User must first disable the parent table before successfully version-disabling this table.

WM_ERROR_105 owner of constraint (*'string'.'string'*) must have select privilege on the parent

Cause: An attempt was made to version-enable a table which was the child table of a referential integrity constraint with another table and the owner of the table to be version-enabled did not have select privilege on the parent table.

Action: Workspace Manager requires that before version-enabling the child table of a integrity constraint, the child table owner must have select privilege on the parent table. Grant the required privilege before version-enabling.

WM_ERROR_106 select and delete privileges needed on the child of constraint (*'string'.'string'*)

Cause: An attempt was made to version-enable a table which was the parent table of a referential integrity constraint with another table and the owner of the table to be version-enabled did not have select/delete privilege on the child table.

Action: Workspace Manager requires that before version-enabling the parent table of a referential integrity constraint, the parent table owner must have select and delete privileges on the child table. Grant select and delete privileges on the child table to the owner of the table being version-enabled.

WM_ERROR_107 select privilege needed on the child of constraint ('*string*'.'*string*')

Cause: An attempt was made to version-enable a table which was the parent table of a referential integrity constraint with another table and the owner of the table to be version-enabled did not have select privilege on the child table.

Action: Workspace Manager requires that before version-enabling the parent table of a referential integrity constraint, the parent table owner must have select and delete privileges on the child table. Grant select and delete privileges on the child table to the owner of the table being version-enabled.

WM_ERROR_108 triggering event '*string*' not allowed

Cause: A triggering event of the form "insert OR update OR delete" was specified.

Action: Drop the trigger and re-create separate triggers (with identical bodies) for insert, update and delete.

WM_ERROR_109 a table with unique constraints cannot be version enabled

Cause: An attempt was made to version-enable a table which had unique constraints defined on it.

Action: Drop the unique constraint on this table before version-enabling it. If the table needs to have a index for performance reasons, create a non-unique index on the relevant set of columns. Oracle will use the created index to optimize queries on the version-enabled table whenever appropriate.

WM_ERROR_112 refresh operation requires ACCESS and MERGE privileges on the workspace

Cause: An attempt was made to invoke RefreshTable or RefreshWorkspace and the user did not have ACCESS/MERGE privilege on the (child) workspace.

Action: Ensure that the invoking user has `ACCESS` and `MERGE` privileges on the (child) workspace before invoking RefreshTable or RefreshWorkspace. Privileges can be granted using the GrantWorkspacePriv or the GrantSystemPriv procedures. Use the function GetPrivs to see which privileges the current user has on a workspace.

WM_ERROR_113 refresh operation requires ACCESS privileges on the parent workspace

Cause: An attempt was made to invoke RefreshTable or RefreshWorkspace and the user did not have ACCESS privilege on the parent workspace.

Action: Ensure that the invoking user has ACCESS privilege on the parent workspace before invoking RefreshTable or RefreshWorkspace. Privileges can be granted using the GrantWorkspacePriv or the GrantSystemPriv procedures. Use the function GetPrivs to see which privileges the current user has on a workspace.

WM_ERROR_114 Continually refreshed workspaces can be created only off of the "LIVE" workspace

Cause: An attempt was made to create a continually refreshed workspace off a non-LIVE workspace.

Action: Workspace Manager only supports creation of continually refreshed workspaces off the LIVE workspace. The user needs to be in the LIVE workspace before invoking CreateWorkspace for creating a continually refreshed workspace.

WM_ERROR_115 ResolveConflicts can be called only after BeginResolve is invoked

Cause: The ResolveConflicts procedure was invoked without calling the BeginResolve procedure first.

Action: Ensure that BeginResolve is invoked by the current user on a workspace before invoking ResolveConflicts for a version-enabled table in that workspace. (See the Resolving Conflicts section of the User Guide for details on the process of resolving conflicts for version-enabled tables.)

WM_ERROR_116 rollback operation requires ACCESS and ROLLBACK privileges on the workspace

Cause: An attempt was made to invoke RollbackTable or RollbackWorkspace and the user did not have ACCESS/ROLLBACK privilege on the workspace.

Action: Ensure that the invoking user has ACCESS and ROLLBACK privileges on the workspace before invoking RollbackTable or RollbackWorkspace. Privileges can be granted using the GrantWorkspacePriv or the GrantSystemPriv procedures. Use the function GetPrivs to see which privileges the current user has on a workspace.

WM_ERROR_117 RollbackResolve can be called only after BeginResolve has been invoked

Cause: RollbackResolve procedure was invoked without calling the BeginResolve procedure first.

Action: Ensure that BeginResolve is invoked first before invoking RollbackResolve. (See the Resolving Conflicts section of the User Guide for details on the process of resolving conflicts for version-enabled tables.)

WM_ERROR_118 savepoint names may not be longer than 30 characters

Cause: An attempt was made to create a savepoint whose name had more than 30 characters.

Action: Please choose a shorter savepoint name.

WM_ERROR_119 savepoint names may not begin with \"ICP-\"

Cause: An attempt was made to create a savepoint whose name began with the string "ICP-".

Action: Choose a savepoint name that does not begin with the string "ICP-". Workspace Manager reserves names starting with "ICP-" for naming implicit savepoints.

WM_ERROR_120 savepoint: *'string'* already exists in workspace: *'string'*

Cause: An attempt was made to create a savepoint with the same name as an existing savepoint. Workspace Manager savepoint names must be unique within a workspace.

Action: Choose another savepoint name.

WM_ERROR_121 savepoint: *'string'* does not exist in workspace: *'string'*

Cause: An attempt was made to invoke a Workspace Manager operation on a savepoint that did not exist in the specified workspace.

Action: Verify that the savepoint name is spelled correctly and that it exists in the specified workspace. Workspace names and savepoint names are case sensitive.

WM_ERROR_122 workspace *'string'* does not exist

Cause: An attempt was made to invoke a Workspace Manager operation on a workspace that did not exist.

Action: Pass in an existing workspace name as input. Workspace names and savepoint names are case sensitive.

WM_ERROR_123 workspace *'string'* is currently frozen in *'string'* mode

Cause: The user invoked a Workspace Manager operation which cannot proceed as the specified workspace has been frozen in the specified mode.

Action: Wait for the database session that holds the lock to release the lock. Refer to the User Guide for a description of the Workspace Manager operations allowed for different workspace freeze modes. Consult the xxx_WM_WORKSPACES view to see which workspaces are currently frozen.

WM_ERROR_124 workspace name may not be "BASE"

Cause: A user attempted to CreateWorkspace with the name BASE.

Action: Workspace Manager considers "BASE" to be a reserved keyword. Therefore, Workspace Manager does not allow the workspace to be named BASE. Choose another workspace name.

WM_ERROR_125 workspace name may not be "LIVE"

Cause: A user attempted to CreateWorkspace with the name LIVE.

Action: Workspace Manager considers "LIVE" to be a reserved keyword. Therefore, Workspace Manager does not allow new workspaces to be named LIVE. Choose another workspace name.

WM_ERROR_126 workspace name may not exceed 30 characters

Cause: A user attempted to create a workspace with the workspace name length greater than 30 characters.

Action: Workspace Manager limits workspace names to 30 characters. Choose a shorter workspace name.

WM_ERROR_127 workspace: *'string'* is already being conflict resolved by user: *'string'*

Cause: A user attempted to invoke BeginResolve on a workspace which was already being conflict resolved by some other user.

Action: Workspace Manager allows only one user to resolve conflicts for a workspace at the same time. Wait until the user is finished resolving conflicts in the workspace and verify that the conflicts you are attempting to resolve still exist. Use the xxx_WORKSPACES views to check on the current resolve status of the workspace.

WM_ERROR_128 workspace: *'string'* is temporarily frozen in an internal mode for a *'string'* operation

Cause: A user attempted to invoke a Workspace Manager operation on a workspace that was frozen internally for another Workspace Manager operation.

Action: Workspace Manager acquires internal freezes on workspaces for the duration of various Workspace Manager operations. Wait until Workspace Manager releases the internal freeze on the workspace. Refer to the User Guide for details on the freezes that Workspace Manager acquires for various workspace-wide operations. Use the xxx_WORKSPACES views to check on the current freeze status of the workspace.

WM_ERROR_129 table *'string'* does not exist

Cause: An attempt was made to invoke a Workspace Manager operation on a table that did not exist.

Action: Verify that the table exists.

WM_ERROR_130 table *'string'* has been modified in an open transaction

Cause: An attempt was made to execute a Workspace Manager operation that required that there be no open short transactions on the table.

Action: Ensure that all open short transactions on the specified table have completed before invoking the Workspace Manager operation.

WM_ERROR_131 table *'string'* is already version enabled

Cause: The specified table is already version-enabled.

Action: To version-disable it, execute the DisableVersioning procedure. The xxx_VERSIONED_TABLES views show all the versioned tables in the database.

WM_ERROR_132 table *'string'* is not version enabled

Cause: This operation can only be invoked on a version-enabled table.

Action: Verify that the specified table is version-enabled. The xxx_VERSIONED_TABLES views show all the versioned tables in the database.

WM_ERROR_133 table *'string'* needs to have a primary key

Cause: An attempt was made to version-enable a table that did not have any primary key defined on it. Workspace Manager requires that a primary key exist on a version-enabled table.

Action: Add a primary key constraint on this table before version-enabling it.

WM_ERROR_134 table *'string'* is already being version disabled

Cause: An attempt was made to version-disable a table which another transaction was in the process of version-disabling.

Action: Wait until the other transaction finishes version-disabling the specified table. The xxx_VERSIONED_TABLES views show all the versioned tables in the database.

WM_ERROR_135 table *'string'* is being version enabled

Cause: An attempt was made to version-enable a table which another transaction was in the process of version-enabling.

Action: Wait until the other transaction finishes version-enabling the specified table. The xxx_VERSIONED_TABLES views show all the versioned tables in the database.

WM_ERROR_136 table names are limited to 25 characters

Cause: An attempt was made to version-enable a table whose name was longer than 25 characters.

Action: Rename the table to a shorter table name.

WM_ERROR_138 table: *'string'* is in use in other sessions

Cause: An attempt to disable version a table has failed due to the existence of short transaction locks on the table.

Action: To successfully disable version this table, verify that there are no short transaction locks on the table.

WM_ERROR_140 invalid value for FreezeMode parameter

Cause: An attempt was made to invoke the FreezeWorkspace procedure with an invalid `freezemode` parameter.

Action: The freezemode parameter for the FreezeWorkspace procedure must be one of (`NO_ACCESS`, `READ_ONLY`, `1WRITER`, `WM_ONLY`). Ensure that Freeze-Workspace is invoked with the correct parameters.

WM_ERROR_141 the parameter freezewriter can be non-null only for the 1WRITER mode

Cause: An attempt was made to invoke the FreezeWorkspace procedure with an invalid `freezewriter` parameter.

Action: The `freezewriter` parameter for the FreezeWorkspace procedure can be non-null only when the `freezemode` parameter is `1WRITER`. Ensure that FreezeWorkspace is invoked with the correct parameters.

WM_ERROR_142 the keep parameter must be one of ("PARENT","CHILD","BASE")

Cause: The ResolveConflicts procedure was called with an invalid keep parameter.

Action: Ensure that the keep parameter to the ResolveConflicts procedure is one of (CHILD,PARENT,BASE). This parameter is not case sensitive. Refer to the Resolving Conflicts section of the User Guide for details on the process of conflict resolution.

WM_ERROR_143 the "LIVE" workspace can only be rolled back to a savepoint

Cause: An attempt was made to rollback the entire LIVE workspace. Workspace Manager only supports the RollbackToSP operation for the LIVE workspace.

Action: Use RollbackToSP to achieve the desired result.

WM_ERROR_144 the "LIVE" workspace cannot be merged

Cause: A user attempted to invoke MergeWorkspace on the LIVE workspace.

Action: Workspace Manager disallows commit of the LIVE workspace. Do not invoke MergeWorkspace on the LIVE workspace.

WM_ERROR_145 the "LIVE" workspace cannot be removed

Cause: A user attempted to invoke RemoveWorkspace on the LIVE workspace.

Action: To rollback changes in the LIVE workspace, use the RollbackToSP operation. To remove descendants to the LIVE workspace, use the RemoveWorkspace operation on the child workspaces.

WM_ERROR_147 the "LIVE" workspace cannot be refreshed

Cause: A user attempted to invoke RefreshWorkspace on the LIVE workspace.

Action: Workspace Manager disallows the Refresh operation on the LIVE workspace. Do not invoke RefreshWorkspace on the LIVE workspace.

WM_ERROR_148 the lock mode is currently not set for this session

Cause: The user invoked a SetLockingOFF operation without having called SetLockingON earlier in the current session.

Action: A user can only execute SetLockingOff if the user had called SetLockingOn in the session. To see what the current lock mode is, use the GetLockMode function.

WM_ERROR_149 the lock mode must be one of ("C","E","S")

Cause: The user invoked a SetLockingON operation with an invalid lockMode parameter.

Action: Use a lockmode that Workspace Manager currently supports: E, or exclusive, and S, or shared. For a discussion of the differences and similarities between these two modes, refer to the Workspace Manager Guide.

WM_ERROR_150 the lock mode is already set for workspace: *'string'*

Cause: An attempt was made to invoke the SetWorkspaceLockModeON operation for a workspace whose lock mode has already been set.

Action: To change the lock mode for a workspace, use the SetWorkspaceLock-ModeOFF procedure to first unset the lock mode.

WM_ERROR_151 the parent workspace *'string'* is currently frozen in *'string'*mode

Cause: An attempt was made to invoke a Workspace Manager operation that required the specified parent workspace to be unfrozen.

Action: Wait for the workspace to be unfrozen before invoking the Workspace Manager operation. The workspace can be unfrozen by the owner of the workspace or by a user with the `WM_ADMIN_ROLE` using the UnfreezeWorkspace procedure.

WM_ERROR_152 the workspace *'string'* is not a leaf workspace

A workspace wide operation was invoked on an intermediate workspace. Workspace Manager supports this operation only on leaf workspaces. A leaf workspace is one that does not have any descendants.

Action: Invoke the operation only on leaf workspaces.

WM_ERROR_153 the workspace: *'string'* has savepoints in the branch specified

Cause: A CompressWorkspace or CompressWorkspaceTree operation resulted in this internal error.

Action: Contact Oracle support to resolve the issue.

WM_ERROR_154 the workspaceLockMode for *'string'* has been set to *'string'* without override

Cause: An attempt was made to invoke the SetLockingON or the SetLockingOFF procedure while the current session was in a workspace whose lock mode was set without override.

Action: The lock mode can be changed by the current session only if the session is in a workspace whose lock mode has not been set or if the session is in a workspace whose lock mode has been set with the override option. Privileged

users can change the lock mode for a workspace using the SetWorkspaceLockModeON and the SetWorkspaceLockModeOFF procedures.

WM_ERROR_155 the where-clause can involve only primary key columns

Cause: An attempt was made to invoke a Workspace Manager operation with an invalid `where_clause` parameter as input.

Action: Ensure that the input `where_clause` contains only valid column names and has proper syntax. The `where_clause` for this Workspace Manager operation can contain only columns which are part of the primary key.

WM_ERROR_156 there are active sessions in the workspace: *'string'*

Cause: An attempt was made to invoke a Workspace Manager operation that required that there be no sessions in the specified workspace.

Action: To successfully invoke the Workspace Manager operation on the specified workspace, ensure that there are no sessions in the workspace. Privileged users can view all the sessions in a workspace using the DBA_WORKSPACE_USERS view.

WM_ERROR_157 there are sessions on non-latest versions in the workspace: *'string'*

Cause: An attempt was made to invoke CompressWorkspace with some sessions in the workspace being on non-LATEST savepoints in the workspace. CompressWorkspace requires that all sessions in the specified workspace be on the latest version of the workspace.

Action: All sessions in the specified workspace must either go to another workspace using GotoWorkspace or must go to the LATEST savepoint using GotoSavepoint. Privileged users can view all the sessions in a workspace using the DBA_WORKSPACE_USERS view.

WM_ERROR_158 this procedure cannot be invoked on the "LIVE" workspace

Cause: An attempt was made to invoke a Workspace Manager procedure on the LIVE workspace.

Action: Invoke this Workspace Manager procedure only on non-LIVE workspaces.

WM_ERROR_159 unable to exclusively lock table: *'string'.'string'*

Cause: An attempt to disable version a table failed due to the existence of short transaction locks on the table.

Action: To successfully disable version this table, verify that there are no short transaction locks on the table.

WM_ERROR_160 unable to grant/revoke appropriate privileges

Cause: An attempt to disable version a table failed due to an internal error in granting/revoking appropriate privileges on the table being version-enabled.

Action: Contact Oracle support to resolve the issue.

WM_ERROR_161 unable to lock *'string'*: *'string'* in *'string'* mode

Cause: An attempt was made to invoke a Workspace Manager operation that failed because Workspace Manager was unable to acquire an exclusive lock on the specified resource.

Action: The specified resource may have been locked by some other database session performing a Workspace Manager operation. Wait for the lock on the resource to be released before proceeding with the Workspace Manager operation.

WM_ERROR_162 unlock operation requires ACCESS privilege on the workspace

Cause: The user attempted to invoke the UnlockRows operation on a workspace without ACCESS privileges on the workspace.

Action: The UnlockRows operation requires ACCESS privileges on the workspace. Invoke the UnlockRows operation only on workspaces that you have ACCESS privileges for.

WM_ERROR_163 use Commit/Rollback Resolve to unfreeze workspaces being conflict resolved

Cause: A user attempted to invoke UnfreezeWorkspace on a workspace being conflict resolved. This workspace was frozen due to a user having issued a BeginResolve operation on it.

Action: To unfreeze it, issue a CommitResolve or a RollbackResolve. Only a user with WM_ADMIN_ROLE or the user who initiated the BeginResolve on the workspace can issue a Commit/Rollback Resolve for that workspace.

WM_ERROR_164 use the RemoveWorkspaceTree procedure to drop non-leaf workspaces

Cause: A user attempted to invoke RemoveWorkspace on an intermediate workspace. To prevent the occurrence of orphaned workspaces, RemoveWorkspace can only be invoked on leaf workspaces.

Action: Execute the RemoveWorkspaceTree procedure to remove the workspace and all its descendants.

WM_ERROR_165 use the force parameter to freeze a currently frozen workspace

Cause: An attempt was made to invoke the FreezeWorkspace procedure for a workspace that was already frozen.

Action: To freeze workspaces that are already frozen, use the FreezeWorkspace procedure with the force parameter.

WM_ERROR_166 only a BeginResolve invoker or a WM_ADMIN_ROLE user can call CommitResolve

Cause: A user attempted to invoke CommitResolve without having initiated the BeginResolve operation earlier and without having the `WM_ADMIN_ROLE`.

Action: CommitResolve can be invoked only by the user who initiated the BeginResolve operation or by a user who has the `WM_ADMIN_ROLE`.

WM_ERROR_167 null lockMode parameter passed in

Cause: A user called a procedure that requires that the `lockMode` parameter have a non-null value.

Action: The user must pass in a non-null `lockMode` parameter for this operation to succeed

WM_ERROR_168 Cannot disable workspace lockmode for continually refreshed workspaces

Cause: An attempt was made to set the workspace lock mode off for a continually refreshed workspaces.

Action: Do not attempt to turn off locking for continually refreshed workspaces.

WM_ERROR_169 "WM_ADMIN_ROLE" or ownership is required to UnFreeze a workspace

Cause: UnfreezeWorkspace failed because only a user with `WM_ADMIN_ROLE` or the owner of the workspace can unfreeze a frozen workspace.

Action: Ensure that the invoking user has the required privileges before attempting to unfreeze the workspace. Otherwise, have the owner of the workspace unfreeze it.

WM_ERROR_170 The row to be locked has already been versioned

Cause: LockRows failed because the row specified to be locked was already versioned.

Action: Do not attempt to lock rows that have already been versioned. Use the where_clause parameter of LockRows to specify those rows which have not already been versioned.

WM_ERROR_171 WM error: *'string'*

Cause: A Workspace Manager error occurred.

Action: Refer to the Workspace Manager Guide.

WM_ERROR_172 all version enabled tables have to be disabled before uninstalling

Cause: An attempt was made to uninstall the Workspace Manager product with existing version-enabled tables.

Action: Version-disable all version-enabled tables before attempting to uninstall the workspace manager product. Version-enabled tables can be disabled using the DisableVersioning procedure.

WM_ERROR_173 cannot create workspaces that are more than 30 levels deep

Cause: An attempt was made to create a workspace that is more than 30 levels in depth from the LIVE workspace.

Action: Do not create workspaces that are more than 30 levels in depth from the LIVE workspace.

WM_ERROR_174 table: \"%s\" contains columns with unsupported data types

Cause: An attempt was made to version-enable a table with one or more columns with an unsupported data type.

Action: Ensure that all the columns in the table being version-enabled are of supported data types. The currently unsupported data types for version-enabled tables are: LONG and LONG RAW.

WM_ERROR_175 cannot delete implicit savepoints with dependent child workspaces

Cause: An attempt was made to invoke the DeleteSavepoint procedure on an implicit savepoint with dependent child workspaces.

Action: Ensure that the savepoint being deleted is not implicit or it does not have any child workspaces created off of it. The xxx_WORKSPACES views show the parent savepoints for all the workspaces in the system. Ensure that the savepoint being deleted is not a parent savepoint for some workspace.

WM_ERROR_176 A user trigger defined on \"%s.%s\" has compilation errors.

Cause: An attempt was made to version-enable a table that has a user defined trigger with compilation errors defined on it.

Action: Ensure that all user defined triggers on the table to be version-enabled have no compilation errors.

WM_ERROR_177 sum of length of all column names of \"%s.%s\" exceeds 8250 characters

Cause: An attempt was made to version-enable a table where the sum of the column name lengths exceeded 8250 characters.

Action: Rename some of the table's columns to reduce the sum of the column name lengths.

WM_ERROR_178 user-defined trigger body defined on \"%s.%s\" exceeds 28000 characters

Cause: An attempt was made to version-enable a table that has a user defined trigger with a trigger body length of more than 28000 characters defined on it.

Action: Ensure that all user defined triggers on the table to be version-enabled have trigger body lengths that are less than 28000 characters.

WM_ERROR_179 combination of column name sizes and user-defined trigger lengths too large

Cause: An attempt was made to version-enable a table where the length of all of the column names combined with the length of the largest trigger body defined on the table was too large.

Action: Reduce the length of the largest trigger body defined on this table and/or rename some of the table's columns to reduce the sum of the column name lengths.

WM_ERROR_180 table \"%s.%s\" has too many primary key columns

Cause: An attempt was made to version-enable a table that has more than 31 primary key columns.

Action: Decrease the number of primary key columns on the table to 31, at most.

WM_ERROR_181 attempt to modify a WM generated trigger

Cause: An attempt to drop or re-create a database trigger failed because that trigger was created by Workspace Manager.

Action: Do not drop or re-create this trigger.

WM_ERROR_182 attempt to modify a WM generated view

Cause: An attempt to re-create a database view failed because it was associated with a version-enabled table.

Action: Do not re-create this view. The view will automatically be dropped when the table associated with it is version-disabled.

WM_ERROR_183 reserved column name found

Cause: An attempt to version-enable the table failed because it had a column with the name as one of the following: VERSION, NEXTVER, DELSTATUS, LTLOCK, CREATETIME, or RETIRETIME.

Action: Rename the column to a different name.

WM_ERROR_184 reserved index name found

Cause: An attempt to version-enable the table failed because it had an index on it with the index name being the name of the table (to version-enabled) with the prefix _PKI$ or _TI$.

Action: Re-create the index using a different name.

Glossary

active version

See current version.

child workspace

A workspace created from its parent workspace.

See also parent workspace and workspace hierarchy.

conflicts

Differences in data values resulting from changes to rows in the child and parent workspace. Conflicts are detected at merge time and presented to the user in conflict views.

See also merging (a workspace).

current version

The version in which the changes are currently being made.

exclusive locking

A Workspace Manager lock mode that prevents any other user from changing a locked row.

See also locks.

explicit savepoint

A savepoint that is explicitly created. It can later be used to perform partial rollbacks in workspaces.

See also savepoint and implicit savepoint.

freezing (a workspace)

Causing the condition in which no changes can be made to data in version-enabled rows in a workspace, and access to the workspace is restricted.

implicit savepoint

A savepoint that is created automatically whenever a new workspace is created.

See also savepoint and explicit savepoint.

LATEST

The name of the logical savepoint that refers to the latest version in the workspace.

See also savepoint.

LIVE

The name of the topmost workspace in the workspace hierarchy.

See also workspace hierarchy.

locks

Version locks provided by Workspace Manager, separate from locks provided by conventional Oracle short transactions. These locks are primarily intended to eliminate row conflicts between a parent workspace and a child workspace. Locking is enabled at a session level and is a session property independent of the workspace that the session is in. When locking is enabled for a session, it locks rows in all workspaces in which it participates.

merging (a workspace)

Applying changes made in a workspace to its parent workspace.

parent workspace

A workspace from which another workspace (a child workspace) was created.

See also child workspace and workspace hierarchy.

privileges

A set of privileges for Workspace Manager that are separate from standard Oracle database privileges. Workspace-level privileges (with names in the form xxx_WORKSPACE*)* that allow the user to affect a specified workspace. System-level privileges (with names in the form xxx_ANY_WORKSPACE*)* that allow the user to affect any workspace.

rolling back (a workspace)

Deleting either all changes made in the workspace or all changes made after a savepoint (that is, an explicit savepoint).

savepoint

A point in the workspace to which operations can be rolled back. It is analogous to a firewall, in that by creating a savepoint you can prevent any damage to the "other side" of the wall (that is, operations performed in the workspace before the savepoint was created).

See also explicit savepoint and implicit savepoint.

shared locking

A Workspace Manager lock mode that allows only users in the workspace in which the row was locked to modify the row.

See also locks.

unfreezing (a workspace)

Reversing the effect of a freeze operation.

See also freezing (a workspace).

version-enabled table

A table in the database in which all rows in the table can now support multiple versions of data. The versioning infrastructure is not visible to the database end users. After a table has been version-enabled, users automatically see the correct version of the record in which they are interested.

workspace

A virtual environment that one or more users can share to make changes to the data in the database. Workspace management involves managing one or more workspaces that can be shared by many users.

workspace hierarchy

The hierarchy of workspaces in the database. For example, a workspace can be a parent to one or more workspaces. By default, when a workspace is created, it is created from the topmost, or `LIVE`, database workspace.

workspace management

The ability of the database to hold different versions of the same record (that is, row) in one or more workspaces.

Index

E

F

G

H

I

L

M

O

P

R

S

T

U

V

W