Contents

Preface ........................................................................................................................................................................ ix
  Audience ..................................................................................................................................................................... ix
  Related documents ........................................................................................................................................................ ix
  Conventions ................................................................................................................................................................. ix
  Documentation Accessibility ........................................................................................................................................ xi
  Technical support ......................................................................................................................................................... xi

What’s New .................................................................................................................................................................... xiii
  New features in Release 11.2.1.6.0 ................................................................................................................................ xiii
  New features in Release 11.2.1.5.0 ................................................................................................................................ xiii
  New features in Release 11.2.1.4.0 ................................................................................................................................ xiii
  New features in Release 11.2.1.1.0 ................................................................................................................................ xiii

1 Oracle In-Memory Database Cache Concepts
  Overview of a cache grid ........................................................................................................................................... 1-1
  Overview of cache groups ....................................................................................................................................... 1-3
    Cache instance ......................................................................................................................................................... 1-4
    Cache group types .................................................................................................................................................. 1-5
    Transmitting updates between the TimesTen and Oracle databases ............................................................... 1-5
    Loading data into a cache group: Explicitly loaded and dynamic cache groups .............................................. 1-6
    Sharing data across a cache grid: Local and global cache groups .............................................................. 1-7
    Summary of cache group types .......................................................................................................................... 1-7
  High availability caching solution .......................................................................................................................... 1-8

2 Getting Started
  Setting up the Oracle and TimesTen systems ......................................................................................................... 2-1
    Create users in the Oracle database .................................................................................................................... 2-1
    Create a DSN for the TimesTen database ............................................................................................................ 2-2
    Create users in the TimesTen database ................................................................................................................ 2-3
    Set the cache administration user name and password in the TimesTen database ........................................... 2-4
  Creating a cache grid .............................................................................................................................................. 2-5
  Creating cache groups .......................................................................................................................................... 2-5
    Create the Oracle tables to be cached .................................................................................................................. 2-6
    Start the cache agent ......................................................................................................................................... 2-7
    Create the cache groups .................................................................................................................................... 2-8
Start the replication agent for the AWT cache group .......................................................... 2-9
Attaching the TimesTen database to the cache grid ............................................................. 2-10
Performing operations on the read-only cache group ......................................................... 2-10
Manually load the cache group ............................................................................................ 2-10
Update the cached Oracle table ........................................................................................... 2-11
Performing operations on the dynamic updatable global cache group ............................ 2-11
Dynamically load the cache group ....................................................................................... 2-12
Update the TimesTen cache table ....................................................................................... 2-12
Cleaning up the TimesTen and Oracle systems .................................................................. 2-13
Detach the TimesTen database from the cache grid ............................................................. 2-14
Stop the replication agent .................................................................................................... 2-14
Drop the cache groups ........................................................................................................ 2-14
Destroy the cache grid ......................................................................................................... 2-14
Stop the cache agent and destroy the TimesTen database ................................................ 2-15
Drop the Oracle users and their objects .............................................................................. 2-15
Procedure for caching Oracle data in TimesTen ............................................................... 2-16

3 Setting Up a Caching Infrastructure

Configuring your system to cache Oracle data in TimesTen .............................................. 3-1
Oracle In-Memory Database Cache environment variables for UNIX ............................ 3-1
Oracle In-Memory Database Cache environment variables for Microsoft Windows .... 3-2
Configuring the Oracle database to cache data in TimesTen ............................................ 3-2
Create the Oracle users ....................................................................................................... 3-2
Grant privileges to the Oracle users ................................................................................... 3-3
Automatically create Oracle objects used to manage caching of Oracle data ................. 3-9
Manually create Oracle objects used to manage caching of Oracle data ......................... 3-10
Configuring a TimesTen database to cache Oracle data ................................................ 3-11
Define a DSN for the TimesTen database .......................................................................... 3-12
Create the TimesTen users ................................................................................................. 3-13
Grant privileges to the TimesTen users ............................................................................ 3-13
Set the cache administration user name and password .................................................... 3-14
Configuring a cache grid ................................................................................................... 3-15
Modify the PROCESSES Oracle system parameter for ten or more grid nodes .......... 3-15
Create a cache grid ............................................................................................................ 3-16
Associate a TimesTen database with a cache grid ......................................................... 3-17
Testing the connectivity between the TimesTen and Oracle databases ....................... 3-17
Managing the cache agent ................................................................................................. 3-17
Set a cache agent start policy ............................................................................................ 3-18

4 Defining Cache Groups

Cache groups and cache tables ......................................................................................... 4-1
Single-table cache group ................................................................................................. 4-2
Multiple-table cache group .............................................................................................. 4-2
Creating a cache group .................................................................................................... 4-5
Read-only cache group ...................................................................................................... 4-6
Restrictions with read-only cache groups ....................................................................... 4-8
Asynchronous writethrough (AWT) cache group ............................................................ 4-9
5 Cache Group Operations

Transmitting updates between the TimesTen and Oracle databases .......................... 5-1
Loading and refreshing a cache group ............................................................... 5-2
Loading and refreshing an explicitly loaded cache group with automatic refresh ...... 5-4
Loading and refreshing a dynamic cache group with automatic refresh ............... 5-4
Loading and refreshing a cache group using a WITH ID clause ......................... 5-5
Initiating an immediate automatic refresh ......................................................... 5-6
Loading and refreshing a multiple-table cache group ........................................ 5-6
Improving the performance of loading or refreshing a large number of cache instances ... 5-7
Example of manually loading and refreshing an explicitly loaded cache group ........ 5-7
Example of manually loading and refreshing a dynamic cache group .................. 5-8
Dynamically loading a cache group .................................................................... 5-10
Types of SQL statements for which dynamic load is available ......................... 5-10
Example of dynamically loading a cache group .............................................. 5-11
Disabling dynamic loading .............................................................................. 5-12
Displaying dynamic load errors ...................................................................... 5-13
Flushing a user managed cache group ............................................................ 5-14
Unloading a cache group ................................................................................. 5-14
Unloading a cache group across all grid members .......................................... 5-15
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determining the number of cache instances affected by an operation</td>
<td>5-15</td>
</tr>
<tr>
<td>Setting a passthrough level</td>
<td>5-15</td>
</tr>
<tr>
<td>Changing the passthrough level for a connection or transaction</td>
<td>5-17</td>
</tr>
<tr>
<td>Determining the passthrough setting for a statement</td>
<td>5-17</td>
</tr>
<tr>
<td>Cache performance</td>
<td></td>
</tr>
<tr>
<td>Dynamic load performance</td>
<td>5-18</td>
</tr>
<tr>
<td>Improving AWT throughput for mixed transactions and network latency</td>
<td>5-18</td>
</tr>
<tr>
<td>Creating Other Cache Grid Members</td>
<td></td>
</tr>
<tr>
<td>Creating and configuring a subsequent standalone TimesTen database</td>
<td>6-1</td>
</tr>
<tr>
<td>Replicating cache tables</td>
<td>6-2</td>
</tr>
<tr>
<td>Create and configure the active master database</td>
<td>6-3</td>
</tr>
<tr>
<td>Create and configure the standby master database</td>
<td>6-5</td>
</tr>
<tr>
<td>Create and configure the read-only subscriber database</td>
<td>6-6</td>
</tr>
<tr>
<td>Example of data sharing among the grid members</td>
<td>6-7</td>
</tr>
<tr>
<td>Performing global queries on a cache grid</td>
<td>6-8</td>
</tr>
<tr>
<td>Restrictions on global queries</td>
<td>6-9</td>
</tr>
<tr>
<td>Adding other elements to a cache grid or grid member</td>
<td>6-9</td>
</tr>
<tr>
<td>Managing a Caching Environment</td>
<td></td>
</tr>
<tr>
<td>Checking the status of the cache and replication agents</td>
<td>7-1</td>
</tr>
<tr>
<td>Monitoring cache groups and cache grids</td>
<td>7-3</td>
</tr>
<tr>
<td>Using the ttlsq1 utility’s cachegroups command</td>
<td>7-3</td>
</tr>
<tr>
<td>Monitoring automatic refresh operations on cache groups</td>
<td>7-4</td>
</tr>
<tr>
<td>Monitoring AWT cache group operations</td>
<td>7-6</td>
</tr>
<tr>
<td>Configuring a transaction log file threshold for AWT cache groups</td>
<td>7-6</td>
</tr>
<tr>
<td>Obtaining information about cache grids</td>
<td>7-7</td>
</tr>
<tr>
<td>Suspending global AWT cache group operations</td>
<td>7-8</td>
</tr>
<tr>
<td>Tracking DDL statements issued on cached Oracle tables</td>
<td>7-8</td>
</tr>
<tr>
<td>Oracle objects used to manage a caching environment</td>
<td>7-10</td>
</tr>
<tr>
<td>Impact of failed automatic refresh operations on TimesTen databases</td>
<td>7-12</td>
</tr>
<tr>
<td>Dropping Oracle objects used by automatic refresh cache groups</td>
<td>7-16</td>
</tr>
<tr>
<td>Monitoring the cache administration user's tablespace</td>
<td>7-17</td>
</tr>
<tr>
<td>Cleaning up the Caching Environment</td>
<td></td>
</tr>
<tr>
<td>Detaching a TimesTen database from a cache grid</td>
<td>8-1</td>
</tr>
<tr>
<td>Stopping the replication agent</td>
<td>8-2</td>
</tr>
<tr>
<td>Dropping a cache group</td>
<td>8-2</td>
</tr>
<tr>
<td>Destroying a cache grid</td>
<td>8-4</td>
</tr>
<tr>
<td>Stopping the cache grid</td>
<td>8-4</td>
</tr>
<tr>
<td>Destroying the TimesTen databases</td>
<td>8-4</td>
</tr>
<tr>
<td>Dropping the Oracle users and objects</td>
<td>8-4</td>
</tr>
<tr>
<td>Using Oracle In-Memory Database Cache in an Oracle RAC Environment</td>
<td></td>
</tr>
<tr>
<td>How Oracle In-Memory Database Cache works in an Oracle RAC environment</td>
<td>9-1</td>
</tr>
<tr>
<td>Restrictions on using IMDB Cache in an Oracle RAC environment</td>
<td>9-4</td>
</tr>
</tbody>
</table>
Setting up IMDB Cache in an Oracle RAC environment ................................................................. 9-4

10 Using Oracle In-Memory Database Cache with Data Guard

Components of MAA for Oracle In-Memory Database Cache .................................................. 10-1
How Oracle In-Memory Database Cache works with Data Guard ............................................. 10-2
Configuring the Oracle databases .............................................................................................. 10-2
Configuring the TimesTen database ............................................................................................ 10-4

11 Compatibility Between TimesTen and Oracle

Summary of compatibility issues .................................................................................................. 11-1
Transaction semantics .................................................................................................................. 11-1
API compatibility ........................................................................................................................ 11-2
SQL compatibility ...................................................................................................................... 11-2
Schema objects ............................................................................................................................ 11-2
Differences between Oracle and TimesTen tables ...................................................................... 11-3
Data type support ....................................................................................................................... 11-3
SQL operators ............................................................................................................................. 11-4
SQL functions .............................................................................................................................. 11-4
SQL expressions .......................................................................................................................... 11-5
SQL subqueries ............................................................................................................................ 11-5
SQL queries .................................................................................................................................. 11-6
  INSERT/DELETE/UPDATE statements .................................................................................... 11-6
TimesTen-only SQL and built-in procedures ............................................................................. 11-6
PL/SQL constructs ....................................................................................................................... 11-7
Mappings between Oracle and TimesTen data types ................................................................. 11-7

12 SQL*Plus Scripts for Oracle In-Memory Database Cache

Installed SQL*Plus scripts .......................................................................................................... 12-1

Glossary

Index
Oracle In-Memory Database Cache is an Oracle Database product option that is ideal for caching performance-critical subsets of an Oracle database into cache tables within TimesTen databases for improved response time in the application tier. Cache tables can be read-only or updatable. Applications read and update the cache tables using standard Structured Query Language (SQL) while data synchronization between the TimesTen database and the Oracle database is performed automatically.

Oracle In-Memory Database Cache offers applications the full generality and functionality of a relational database, the transparent maintenance of cache consistency with the Oracle database, and the real-time performance of an in-memory database.

**Audience**

This guide is for application developers who use and administer TimesTen, and for system administrators who configure and manage TimesTen databases that cache data from Oracle databases.

To work with this guide, you should understand how relational database systems work. You should also have knowledge of SQL, and either Open Database Connectivity (ODBC), Java Database Connectivity (JDBC), or Oracle Call Interface (OCI).

**Related documents**

TimesTen documentation is available on the product distribution media and on the Oracle Technology Network (OTN):

http://www.oracle.com/technology/documentation/timesten_doc.html

**Conventions**

This document uses the following text conventions:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>boldface</td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text.</td>
</tr>
<tr>
<td>italic</td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td>monospace</td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
<tr>
<td>italic monospace</td>
<td>Italic monospace type indicates a variable in a code example that you must replace. For example:</td>
</tr>
<tr>
<td></td>
<td>Driver=TimesTen_install_dir/lib/libtten.so</td>
</tr>
<tr>
<td></td>
<td>Replace TimesTen_install_dir with the path of your TimesTen installation directory.</td>
</tr>
<tr>
<td>[]</td>
<td>Square brackets indicate that an item in a command line is optional.</td>
</tr>
<tr>
<td>{}</td>
<td>Curly braces indicate that you must choose one of the items separated by a vertical bar (</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>An ellipsis (…) after an argument indicates that you may use more than one argument on a single command line.</td>
</tr>
<tr>
<td>%</td>
<td>The percent sign indicates the UNIX shell prompt.</td>
</tr>
<tr>
<td>#</td>
<td>The number (or pound) sign indicates the prompt for the UNIX root user.</td>
</tr>
</tbody>
</table>

TimesTen documentation uses these variables to identify path, file and user names:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimesTen_install_dir</td>
<td>The path that represents the directory where the current release of TimesTen is installed.</td>
</tr>
<tr>
<td>TTinstance</td>
<td>The instance name for your specific installation of TimesTen. Each installation of TimesTen must be identified at install time with a unique alphanumeric instance name. This name appears in the install path.</td>
</tr>
<tr>
<td>bits or bb</td>
<td>Two digits, 32 or 64, that represent either the 32-bit or 64-bit version of the operating system.</td>
</tr>
<tr>
<td>release or rr</td>
<td>Digits that represent the TimesTen release number, with or without dots. For example, 1121 or 11.2.1 represents TimesTen Release 11.2.1.</td>
</tr>
<tr>
<td>jdk_version</td>
<td>One or two digits that represent the major version number of the Java Development Kit (JDK) release. For example, 5 represents JDK 5.</td>
</tr>
<tr>
<td>DSN</td>
<td>The data source name.</td>
</tr>
</tbody>
</table>
Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. 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. Accessibility 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 more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/.

Accessibility of Code Examples in Documentation
Screen readers 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, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation
This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Access to Oracle Support
Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/support/contact.html or visit http://www.oracle.com/accessibility/support.html if you are hearing impaired.

Technical support
For information about obtaining technical support for TimesTen products, go to the following Web address:

http://www.oracle.com/support/contact.html
What's New

This section summarizes the new features of Oracle In-Memory Database Cache release 11.2.1 that are documented in this guide and provides links to more information.

New features in Release 11.2.1.6.0

You can perform a global query on cache tables and noncache tables across all nodes in a cache grid. See "Performing global queries on a cache grid" on page 6-8.

You can unload a cache group on all grid members by specifying a global unload operation. See "Unloading a cache group across all grid members" on page 5-15.

New features in Release 11.2.1.5.0

You can cache Oracle CLOB, BLOB and NCLOB data as TimesTen VARCHAR2, VARBINARY or NVARCHAR2 data. See "Caching Oracle LOB data" on page 4-29.

New features in Release 11.2.1.4.0

This section summarizes the new features of Oracle In-Memory Database Cache that are documented in this guide.

- You can specify PL/SQL execution mode for AWT cache groups. See "Improving AWT throughput for mixed transactions and network latency" on page 5-18.
- You can specify a time unit of seconds for cache group aging. See "Implementing aging on a cache group" on page 4-30.
- You can initiate an immediate automatic refresh operation. See "Initiating an immediate automatic refresh" on page 5-6.
- You can suspend and resume operations on global AWT cache groups. See "Suspending global AWT cache group operations" on page 7-8.
- You can detach all of the cache grid members at once. See "Detaching a TimesTen database from a cache grid" on page 8-1.

New features in Release 11.2.1.1.0

This section summarizes the new features of Oracle In-Memory Database Cache that are documented in this guide.
Cache grid
By default, you must create a cache grid to cache Oracle data in TimesTen databases. A cache grid provides read and write consistency on the cache tables among the TimesTen databases within the grid.
See "Configuring a cache grid" on page 3-15 for information about defining a cache grid on one or more TimesTen databases.

Global cache group
Creating a global cache group allows data in its cache tables to be shared across multiple TimesTen databases within a cache grid. Only a dynamic asynchronous writethrough (AWT) cache group can be defined as a global cache group.
See "Global cache groups" on page 4-37 for details about creating a global cache group.
With a local cache group, data in its cache tables are not shared across TimesTen databases regardless of whether the databases are members of the same cache grid. Any cache group type and category can be defined as a local cache group.

Dynamic cache group
In a dynamic cache group, the data in its cache tables can be loaded on demand or manually. You can create a dynamic cache group for nearly any cache group type and classification.
See "Dynamic cache groups" on page 4-36 for details about creating a dynamic cache group.
See "Dynamically loading a cache group" on page 5-10 for details about loading data into a dynamic cache group’s cache tables.
With an explicitly loaded cache group, data is manually or automatically loaded into its cache tables. Any cache group type and classification can be defined as an explicitly loaded cache group.

Monitoring DDL operations on Oracle tables
You can enable tracking of DDL operations performed on Oracle tables that are cached in a TimesTen database. Tracking DDL operations on cached Oracle tables can be useful for monitoring or troubleshooting purposes.
See "Tracking DDL statements issued on cached Oracle tables" on page 7-8 for details about enabling tracking of DDL operations on Oracle tables.

Recovering cache groups after failed automatic refresh operations
If a TimesTen database contains automatic refresh cache groups and the cache agent is not running on the database, automatic refresh operations are attempted, by default, but fail on those cache groups within that database. The failed automatic refresh attempts impacts the performance of automatic refresh operations on cache groups in other TimesTen databases that cache data from the same Oracle database.

Note: An automatic refresh cache group refers to a read-only cache group or a user managed cache group that uses the AUTOREFRESH MODE INCREMENTAL cache group attribute.

See "Impact of failed automatic refresh operations on TimesTen databases" on page 7-12 for details about reducing the performance degradation in this situation and recovering the cache groups which failed to be automatically refreshed.
Monitoring the cache administration user's tablespace usage

You can configure an action to occur when the cache administration user's tablespace becomes full. For automatic refresh cache groups, change log tables are created in the cache administration user's tablespace. When an update operation is issued on an Oracle table that is cached in one of these cache groups, the configured action is performed when there is no space available in the cache administration user's tablespace to insert a new row into the change log table.

See "Monitoring the cache administration user's tablespace" on page 7-17 for details about the configured actions, and how to return a warning to the application when the cache administration user's tablespace usage exceeds a configured threshold.
Oracle In-Memory Database Cache Concepts

Oracle In-Memory Database Cache is an Oracle Database product option that includes the Oracle TimesTen In-Memory Database. It is used as a database cache at the application tier to cache Oracle data and reduce the workload on the Oracle database. It also provides the connection and transfer of data between an Oracle database and a TimesTen database, as well as facilitating the capture and processing of high-volume event flows into a TimesTen database and subsequent transfer of data into an Oracle database.

You can cache Oracle data in a TimesTen database by defining a cache grid and then creating cache groups. A cache group in a TimesTen database can cache a single Oracle table or a group of related Oracle tables.

This chapter includes the following topics:

- Overview of a cache grid
- Overview of cache groups
- High availability caching solution

Overview of a cache grid

A cache grid is a set of distributed TimesTen in-memory databases that work together to cache data from an Oracle database and guarantee cache coherence among the TimesTen databases. A grid consists of one or more in-memory database grid members that collectively manage the application data using the relational data model. The members of a grid cache data from a single Oracle database. Each grid member is backed by either a standalone TimesTen database or an active standby pair.

A grid node is the database of a grid member. A node is one of the following:

- A standalone TimesTen database
- The active master database in an active standby pair
- The standby master database in an active standby pair

Thus a grid member that is a standalone database consists of one node. A grid member that is an active standby pair consists of two nodes.

Figure 1–1 shows a cache grid containing three members: two standalone TimesTen databases and an active standby pair. The grid has four nodes: the two standalone TimesTen databases, the active master database and the standby master database of the active standby pair. The read-only subscriber database is not part of the cache grid because it has no connectivity with the Oracle database. The read-only subscriber receives replicated updates from the standby master database.
In a cache grid, cached data is dynamically distributed across multiple grid members without shared storage. This architecture allows the capacity of the cache grid to scale based on the processing needs of the application. When the workload increases or decreases, new grid members attach to the grid or existing grid members detach from the grid. Attaching to or detaching from the grid are online operations that do not interrupt operations on other grid members.

When requests are submitted to the grid members, the cache grid automatically redistributes data based on application access patterns. The location of the data is transparent to the application but the cache grid redistributes data dynamically to minimize access time. The cache grid automatically maintains cache coherence and transactional consistency across the grid members.

You can configure the cache grid to perform global queries without redistributing the data. See "Performing global queries on a cache grid" on page 6-8.

TimesTen databases within a cache grid can contain explicitly loaded and dynamic cache groups, as well as local and global cache groups of any cache group type that is supported for the various cache group classifications and categories.

See "Cache group types" on page 1-5 for details about the different types of cache groups.

See "Loading data into a cache group: Explicitly loaded and dynamic cache groups" on page 1-6 for details about the differences between an explicitly loaded and a dynamic cache group.

See "Sharing data across a cache grid: Local and global cache groups" on page 1-7 for details about the differences between a local and a global cache group.

See "Configuring a cache grid" on page 3-15 for information about creating a cache grid and associating a TimesTen database with a cache grid.
Overview of cache groups

Cache groups define the Oracle data to be cached in a TimesTen database. A cache group can be defined to cache all or part of a single Oracle table, or a set of related Oracle tables.

Figure 1–2 shows a cache group target_customers that caches a subset of a single Oracle table customer.

Figure 1–2  Single-table cache group

You can cache multiple Oracle tables in the same cache group by defining a root table and one or more child tables. A cache group can contain only one root table.

In a cache group with multiple tables, each child table must reference the root table or another child table in the same cache group using a foreign key constraint. Although tables in a multiple-table cache group must be related to each other in the TimesTen database through foreign key constraints, the corresponding tables do not necessarily need to be related to each other in the Oracle database. The root table does not reference any table with a foreign key constraint. See "Multiple-table cache group" on page 4-2 for more details about the characteristics of a multiple-table cache group.
Cache instance

Data is loaded from an Oracle database into a cache group within a TimesTen database in units called cache instances. A cache instance is defined as a single row in the cache group’s root table together with the set of related rows in the child tables.

**Figure 1–3** shows three tables in the `customer_orders` cache group. The root table is `customer`, and `orders` and `order_item` are child tables. The cache instance identified by the row with the value 122 in the `cust_num` primary key column of the customer table includes:

- The two rows with the value 122 in the `cust_num` column of the `orders` table (whose value in the `ord_num` primary key column is 44325 or 65432), and
- The three rows with the value 44325 or 65432 in the `ord_num` column of the `order_item` table
Cache group types

The most commonly used types of cache groups are:

- **Read-only cache group**
  
  A read-only cache group enforces a caching behavior in which committed updates on cached tables in the Oracle database are automatically refreshed to the cache tables in the TimesTen database. Using a read-only cache group is suitable for reference data that is heavily accessed by applications.
  
  See "Read-only cache group" on page 4-6 for details about read-only cache groups.

- **Asynchronous writethrough (AWT) cache group**
  
  An AWT cache group enforces a caching behavior in which committed updates on cache tables in the TimesTen database are automatically propagated to the cached tables in the Oracle database in asynchronous fashion. Using an AWT cache group is suitable for high speed data capture and online transaction processing.
  
  See "Asynchronous writethrough (AWT) cache group" on page 4-9 for details about AWT cache groups.

Other types of cache groups include:

- **Synchronous writethrough (SWT) cache group**
  
  An SWT cache group enforces a caching behavior in which committed updates on cache tables in the TimesTen database are automatically propagated to the cached tables in the Oracle database in synchronous fashion.
  
  See "Synchronous writethrough (SWT) cache group" on page 4-13 for details about SWT cache groups.

- **User managed cache group**
  
  A user managed cache group defines customized caching behavior.
  
  For example, you can define a cache group that does not use automatic refresh nor automatic propagation where committed updates on the cache tables are manually propagated or flushed to the cached Oracle tables.
  
  You can also define a cache group that uses both automatic propagation in synchronous fashion on every table and automatic refresh.
  
  See "User managed cache group" on page 4-15 for details about user managed cache groups.

Transmitting updates between the TimesTen and Oracle databases

Transmitting committed updates between the TimesTen cache tables and the cached Oracle tables keeps these tables in the two databases synchronized.

As shown in Figure 1–4, propagate or flush is the process of transmitting committed updates on cache tables in the TimesTen database to the cached tables in the Oracle database. Flush is a manual operation and propagate is an automatic operation.

Load, refresh or automatic refresh is the process of transmitting committed updates on cached tables in the Oracle database to the cache tables in the TimesTen database. Load and refresh are manual operations, and automatic refresh is an automatic operation.

See "Flushing a user managed cache group" on page 5-14 for information about the FLUSH CACHE GROUP statement which can only be issued on a user managed cache group.
Overview of cache groups

See "Asynchronous writethrough (AWT) cache group" on page 4-9 and "Synchronous writethrough (SWT) cache group" on page 4-13 for details about how a propagate operation is processed on AWT and SWT cache groups, respectively.

See "Loading and refreshing a cache group" on page 5-2 for information about the LOAD CACHE GROUP and REFRESH CACHE GROUP statements.

See "AUTOREFRESH cache group attribute" on page 4-21 for details about an automatic refresh operation.

Figure 1–4 Transmitting committed updates between the TimesTen and Oracle databases

Loading data into a cache group: Explicitly loaded and dynamic cache groups

Cache groups are categorized as either explicitly loaded or dynamic.

In an explicitly loaded cache group, cache instances are loaded manually into the TimesTen cache tables from Oracle using a load or refresh operation, or automatically using an automatic refresh operation. The cache tables are loaded before operations such as queries are performed on the tables. An explicitly loaded cache group is appropriate when the set of data to cache is static and can be predetermined before applications begin performing operations on the cache tables. By default, cache groups are explicitly loaded unless they are defined as dynamic.

In a dynamic cache group, cache instances are loaded into the TimesTen cache tables on demand from Oracle using a dynamic load operation, or manually using a load operation. A manual refresh or an automatic refresh operation on a dynamic cache group can result in existing cache instances being updated or deleted, but committed updates on Oracle data that are not being cached do not result in new cache instances being loaded into its cache tables. A dynamic cache group is appropriate when the set...
of data to cache is small and should not be preloaded from Oracle before applications begin performing operations on the cache tables.

See "Transmitting updates between the TimesTen and Oracle databases" on page 1-5 for details about cache group load and refresh operations.

See "Loading and refreshing a cache group" on page 5-2 for more details about the differences between performing a load and a refresh operation on an explicitly loaded cache group, and performing the same operations on a dynamic cache group.

See "Dynamically loading a cache group" on page 5-10 for details about a dynamic load operation.

Any cache group type (read-only, AWT, SWT, user managed) can be defined as an explicitly loaded cache group. All cache group types except a user managed cache group that uses both the AUTOREFRESH cache group attribute and the PROPAGATE cache table attribute can be defined as a dynamic cache group.

See "Dynamic cache groups" on page 4-36 for more information about dynamic cache groups.

Sharing data across a cache grid: Local and global cache groups

In addition to being explicitly loaded or dynamic, cache groups are also classified as either local or global.

In a local cache group, data in its cache tables are not shared across TimesTen databases even if the databases are members of the same cache grid. Therefore, the databases can have overlapping data because the cache instances are local to a specific grid member. Committed updates on the TimesTen cache tables are propagated to the cached Oracle tables without coordination with other grid members. Any cache group type can be defined as a local cache group. A local cache group can be defined either as explicitly loaded or dynamic. Using a local cache group is suitable for reference data that is read frequently and can be present in all grid members, and for disjoint data that is logically partitioned for optimal concurrency and throughput. By default, cache groups are local unless they are defined as global.

In a global cache group, data in its cache tables are shared across TimesTen databases that are members of the same cache grid. Committed updates to the same data on different grid members are propagated to Oracle in the order in which they were issued within the grid to ensure read/write data consistency across the members of the grid. Only a dynamic AWT cache group can be defined as global cache group. New cache instances are loaded into the cache tables of a global cache group on demand because the cache group must be defined as dynamic. Using a global cache group is suitable for updatable data that can only be accessed by or present in one grid member at a time in order to ensure that the data is consistent among both the members of the grid and the Oracle database.

See "Global cache groups" on page 4-37 for information about creating and using a global cache group.

Summary of cache group types

The table in Figure 1–5 summarizes the valid combinations of cache group types, categories and classifications available to the user at cache group creation time. The cache group categories determine how the data is loaded into the cache group. The cache group classifications determine whether data in the cache group can be shared across a cache grid.
You can create an explicitly loaded local cache group or a dynamic local cache group of any cache group type. You can create a global cache group only for a cache group whose category and type is dynamic AWT.

### Figure 1–5  Summary of cache group types and categories

<table>
<thead>
<tr>
<th>Cache group type</th>
<th>Loading data into a cache group</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Explicitly loaded</td>
<td>Dynamic</td>
<td></td>
</tr>
<tr>
<td>Read-only</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AWT</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>SWT</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>User managed</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Local</td>
<td>Global</td>
<td>Local</td>
</tr>
</tbody>
</table>

### High availability caching solution

You can configure Oracle In-Memory Database Cache to achieve high availability of cache tables, and facilitate failover and recovery while maintaining connectivity to the Oracle database. A TimesTen database that is a participant in an active standby pair replication scheme can provide high availability for cache tables in a read-only or an AWT cache group.

An active standby pair provides for high availability of a TimesTen database. Multiple grid members provide for high availability of a TimesTen cache grid. Oracle Real Application Clusters (Oracle RAC) and Data Guard provides for high availability of an Oracle database.

See "Replicating cache tables" on page 6-2 for information on configuring replication of cache tables.

See "Using Oracle In-Memory Database Cache in an Oracle RAC Environment" on page 9-1 for more information on Oracle In-Memory Database Cache and Oracle RAC.

See "Using Oracle In-Memory Database Cache with Data Guard" on page 10-1 for more information on Oracle In-Memory Database Cache and Data Guard.
This chapter describes how to create a cache grid. To illustrate the creation and use of cache groups, the chapter describes how to create an explicitly loaded read-only local cache group, and a dynamic updatable global cache group. The chapter also describes how to populate the cache tables, and how to observe the transfer of updates between the cache tables in the TimesTen database and the cached tables in the Oracle database.

This chapter includes the following topics:

- Setting up the Oracle and TimesTen systems
- Creating a cache grid
- Creating cache groups
- Attaching the TimesTen database to the cache grid
- Performing operations on the read-only cache group
- Performing operations on the dynamic updatable global cache group
- Cleaning up the TimesTen and Oracle systems
- Procedure for caching Oracle data in TimesTen

## Setting up the Oracle and TimesTen systems

Before you can create a cache grid or a cache group, you must first install TimesTen and then configure the Oracle and TimesTen systems. See Oracle TimesTen In-Memory Database Installation Guide for information about installing TimesTen.

Complete the following tasks:

1. Create users in the Oracle database
2. Create a DSN for the TimesTen database
3. Create users in the TimesTen database
4. Set the cache administration user name and password in the TimesTen database

## Create users in the Oracle database

Before you can use Oracle In-Memory Database Cache, you must create some Oracle users:

- A user `timesten` owns Oracle tables that store information about cache grids.
- One or more schema users own the Oracle tables to be cached in a TimesTen database. These may be existing users or new users.
A cache administration user creates and maintains Oracle objects that store information used to manage cache grids and enforce predefined behaviors of particular cache group types.

Start SQL*Plus on the Oracle system from an operating system shell or command prompt, and connect to the Oracle database instance as the sys user:

```
% cd TimesTen_install_dir/oraclescripts
% sqlplus sys as sysdba
Enter password: password
```

Use SQL*Plus to create a default tablespace that will be used by both the timesten user and the cache administration user. This tablespace should only be used to store objects for Oracle In-Memory Database Cache and should not be shared with other applications. Then run the SQL*Plus script `TimesTen_install_dir/oraclescripts/initCacheGlobalSchema.sql` to create the following elements:

- The timesten user
- The Oracle tables owned by the timesten user to store information about cache grids
- The TT_CACHE_ADMIN_ROLE role that defines privileges on these Oracle tables

Pass the default tablespace as an argument to the `initCacheGlobalSchema.sql` script. In the following example, the name of the default tablespace is `cachetblsp`:

```
SQL> CREATE TABLESPACE cachetblsp DATAFILE 'datfttuser.dbf' SIZE 100M;
SQL> @initCacheGlobalSchema "cachetblsp"
```

Next use SQL*Plus to create a schema user. Grant this user the minimum set of privileges required to create tables in the Oracle database to be cached in a TimesTen database. In the following example, the schema user is `oratt`:

```
SQL> CREATE USER oratt IDENTIFIED BY oracle;
SQL> GRANT CREATE SESSION, RESOURCE TO oratt;
```

Then use SQL*Plus to perform the following operations:

- Create a cache administration user.
- Run the SQL*Plus script `TimesTen_install_dir/oraclescripts/grantCacheAdminPrivileges.sql` to grant the cache administration user the minimum set of privileges required to perform cache grid and cache group operations.

Pass the cache administration user name as an argument to the `grantCacheAdminPrivileges.sql` script. In the following example, the cache administration user name is `cacheuser` and the name of its default tablespace is `cachetblsp`:

```
SQL> CREATE USER cacheuser IDENTIFIED BY oracle
2  DEFAULT TABLESPACE cachetblsp QUOTA UNLIMITED ON cachetblsp;
SQL> @grantCacheAdminPrivileges 'cacheuser'
SQL> exit
```

The privileges that the cache administration user requires depend on the types of cache groups you create and the operations that you perform on the cache groups. See "Create the Oracle users" on page 3-2 for more information about the timesten user, the schema users, and the cache administration user.
Create a DSN for the TimesTen database

In the following data source name (DSN) examples, the net service name of the Oracle database instance is `oracledb` and its database character set is AL32UTF8. The TimesTen database character set must match the Oracle database character set. You can determine the Oracle database character set by executing the following query in SQL*Plus as any user:

```sql
SQL> SELECT value FROM nls_database_parameters WHERE parameter='NLS_CHARACTERSET';
```

On UNIX, in the `.odbc.ini` file that resides in your home directory or the `TimesTen_install_dir/info/sys.odbc.ini` file, create a TimesTen DSN `cachealone1` and set the following attributes:

```
[cachealone1]
DataStore=/users/OracleCache/alone1
PermSize=64
OracleNetServiceName=oracledb
DatabaseCharacterSet=AL32UTF8
```

On Windows, create a TimesTen user DSN or system DSN `cachealone1` and set the following attributes:

- Data Store Path + Name: `c:\temp\alone1`
- Permanent Data Size: 64
- Oracle Net Service Name: `oracledb`
- Database Character Set: `AL32UTF8`

Use the default settings for all the other DSN attributes.

See "Define a DSN for the TimesTen database" on page 3-12 for more information about defining a DSN for a TimesTen database that is used to cache data from an Oracle database.

See "Managing TimesTen Data Stores" in Oracle TimesTen In-Memory Database Operations Guide for more information about TimesTen DSNs.

---

**Note:** The term "data store" is used interchangeably with "TimesTen database".

Create users in the TimesTen database

In addition to the Oracle users, you must create some TimesTen users before you can use Oracle In-Memory Database Cache:

- A **cache manager user** performs cache grid and cache group operations. The TimesTen cache manager user must have the same name as an Oracle user that can access the cached Oracle tables. For example, the Oracle user must have privileges to select from and update the cached Oracle tables. The Oracle user can be the cache administration user, a schema user, or some other existing user. The password of the cache manager user can be different than the password of the Oracle user with the same name.

  The cache manager user creates and configures the cache grid and creates the cache groups. It may perform operations such as loading or refreshing a cache group although these operations can be performed by any TimesTen user that has sufficient privileges. The cache manager user can also monitor various aspects of
the caching environment, such as the grid itself or asynchronous operations that are performed on cache groups such as automatic refresh.

- One or more cache table users own the cache tables. You must create a TimesTen cache table user with the same name as an Oracle schema user for each schema user who owns or will own Oracle tables to be cached in the TimesTen database. The password of a cache table user can be different than the password of the Oracle schema user with the same name.

  The owner and name of a TimesTen cache table is the same as the owner and name of the corresponding cached Oracle table.

Start the ttIsql utility on the TimesTen system from an operating system shell or command prompt as the instance administrator, and connect to the cachealone1 DSN to create the TimesTen database that will be used to cache data from an Oracle database:

  % ttIsql cachealone1

Use ttIsql to create a cache manager user. Grant this user the minimum set of privileges required to create a cache grid and cache groups, and perform operations on the cache groups. In the following example, the cache manager user name is cacheuser, which is the same name as the Oracle cache administration user that was created earlier:

  Command> CREATE USER cacheuser IDENTIFIED BY timesten;
  Command> GRANT CREATE SESSION, CACHE_MANAGER, CREATE ANY TABLE TO cacheuser;

Then use ttIsql to create a cache table user. In the following example, the cache table user name is oratt, which is the same name as the Oracle schema user that was created earlier:

  Command> CREATE USER oratt IDENTIFIED BY timesten;
  Command> exit

The privileges that the cache manager user requires depend on the types of cache groups you create and the operations that you perform on the cache groups. See "Create the TimesTen users" on page 3-13 for more information about the cache manager user and the cache table users.

See "Managing Access Control" in Oracle TimesTen In-Memory Database Operations Guide for more information about TimesTen users and privileges.

**Set the cache administration user name and password in the TimesTen database**

Start the ttIsql utility and connect to the cachealone1 DSN as the cache manager user. In the connection string, specify the cache manager user name in the UID attribute. (In this example, the TimesTen cache manager user name is the same as the Oracle cache administration user name.) Specify the cache manager user's password in the PWD attribute and the cache administration user's password in the OraclePWD attribute within the connection string.

  % ttIsql "DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"

Use ttIsql to call the ttCacheUidPwdSet built-in procedure to set the Oracle cache administration user name and password:

  Command> call ttCacheUidPwdSet('cacheuser','oracle');
The cache administration user name and password need to be set only once in a TimesTen database. See "Set the cache administration user name and password" on page 3-14 for information about how this setting is used by the TimesTen database.

Creating a cache grid

After you have created the Oracle users, the TimesTen database, and the TimesTen users, and set the Oracle cache administration user name and password in the TimesTen database, you need to create a cache grid to define a framework for TimesTen databases that cache tables from an Oracle database.

As the cache manager user, use the ttIsql utility to call the ttGridCreate built-in procedure to create a cache grid myGrid:

```
Command> call ttGridCreate('myGrid');
```

Then use ttIsql to call the ttGridNameSet built-in procedure to associate the TimesTen database with the myGrid cache grid:

```
Command> call ttGridNameSet('myGrid');
```

See "Configuring a cache grid" on page 3-15 for more information about the contents and functionality of a cache grid.

Creating cache groups

After you have created a cache grid and associated the TimesTen database with the grid, you are ready to create cache groups. You create a read-only cache group as shown in Figure 2–1. Then you create an asynchronous writethrough (AWT) cache group as shown in Figure 2–2.

*Figure 2–1  Single-table read-only cache group*
Complete the following tasks to create a read-only cache group and an AWT cache group:

1. Create the Oracle tables to be cached
2. Start the cache agent
3. Create the cache groups
4. Start the replication agent for the AWT cache group

Create the Oracle tables to be cached

Start SQL*Plus and connect to the Oracle database as the schema user:

$ sqlplus oratt/oracle

Use SQL*Plus to create a table `readtab` as shown in Figure 2–3, and a table `writetab` as shown in Figure 2–4:

```
SQL> CREATE TABLE readtab (keyval NUMBER NOT NULL PRIMARY KEY, str VARCHAR2(32));
SQL> CREATE TABLE writetab (pk NUMBER NOT NULL PRIMARY KEY, attr VARCHAR2(40));
```

Create the Oracle tables to be cached

Start SQL*Plus and connect to the Oracle database as the schema user:

$ sqlplus oratt/oracle

Use SQL*Plus to create a table `readtab` as shown in Figure 2–3, and a table `writetab` as shown in Figure 2–4:
Creating cache groups

Getting Started

Figure 2–4 Creating an Oracle table to be cached in an AWT cache group

Then use SQL*Plus to insert some rows into the readtab and writetab tables, and commit the changes:

```
SQL> INSERT INTO readtab VALUES (1, 'Hello');
SQL> INSERT INTO readtab VALUES (2, 'World');
SQL> INSERT INTO writetab VALUES (100, 'TimesTen');
SQL> INSERT INTO writetab VALUES (101, 'IMDB');
SQL> COMMIT;
```

Next use SQL*Plus to grant the SELECT privilege on the readtab table, and the SELECT, INSERT, UPDATE and DELETE privileges on the writetab table to the cache administration user:

```
SQL> GRANT SELECT ON readtab TO cacheuser;
SQL> GRANT SELECT ON writetab TO cacheuser;
SQL> GRANT INSERT ON writetab TO cacheuser;
SQL> GRANT UPDATE ON writetab TO cacheuser;
SQL> GRANT DELETE ON writetab TO cacheuser;
```

The SELECT privilege on the readtab table is required to create a read-only cache group that caches this table and to perform automatic refresh operations from the cached Oracle table to the TimesTen cache table.

The SELECT privilege on the writetab table is required to create an AWT cache group that caches this table. The INSERT, UPDATE and DELETE privileges on the writetab table are required to perform writethrough operations from the TimesTen cache table to the cached Oracle table.

See “Grant privileges to the Oracle users” on page 3-3 for more information about the privileges required for the cache administration user to create and perform operations on a read-only cache group and an AWT cache group.

Start the cache agent

As the cache manager user, use the ttIsq1 utility to call the ttCacheStart built-in procedure to start the cache agent on the TimesTen database:

```
```
Command> call ttCacheStart;

See "Managing the cache agent" on page 3-17 for more information about starting the cache agent.

Create the cache groups

As the cache manager user, use the ttIsql utility to create a read-only cache group readcache that caches the Oracle oratt.readtab table as shown in Figure 2–5 and a dynamic AWT global cache group writecache that caches the Oracle oratt.writetab table as shown in Figure 2–6:

Command> CREATE READONLY CACHE GROUP readcache
    > AUTOREFRESH INTERVAL 5 SECONDS
    > FROM oratt.readtab
    > (keyval NUMBER NOT NULL PRIMARY KEY, str VARCHAR2(32));

Command> CREATE DYNAMIC ASYNCHRONOUS WRITETHROUGH GLOBAL CACHE GROUP writecache
    > FROM oratt.writetab
    > (pk NUMBER NOT NULL PRIMARY KEY, attr VARCHAR2(40));

Figure 2–5  Creating a read-only cache group

Figure 2–6  Creating an asynchronous writethrough cache group
The cache groups readcache and writecache, and their respective cache tables oratt.readtab and oratt.writetab, whose owners and names are identical to the cached Oracle tables, are created in the TimesTen database.

Use the ttIsql cachegroups command to view the definition of the readcache and writecache cache groups:

```
Command> cachegroups;
```

**Cache Group CACHEUSER.READCACHE:**

- Cache Group Type: Read Only
- Autorefresh: Yes
- Autorefresh Mode: Incremental
- Autorefresh State: Paused
- Autorefresh Interval: 5 Seconds
- Autorefresh Status: ok
- Aging: No aging defined

  Root Table: ORATT.READTAB
  Table Type: Read Only

**Cache Group CACHEUSER.WRITECACHE:**

- Cache Group Type: Asynchronous Writethrough global (Dynamic)
- Autorefresh: No
- Aging: LRU on

  Root Table: ORATT.WRITETAB
  Table Type: Propagate

2 cache groups found.

See "Read-only cache group" on page 4-6 for more information about read-only cache groups.

See "Asynchronous writethrough (AWT) cache group" on page 4-9 for more information about AWT cache groups.

See "Dynamic cache groups" on page 4-36 for more information about dynamic cache groups.

See "Global cache groups" on page 4-37 for more information about global cache groups.

### Start the replication agent for the AWT cache group

As the cache manager user, use the ttIsql utility to call the ttRepStart built-in procedure to start the replication agent on the TimesTen database:

```
Command> call ttRepStart;
```

The replication agent propagates committed updates on TimesTen cache tables in AWT cache groups to the cached Oracle tables.

See "Managing the replication agent" on page 4-10 for more information about starting the replication agent.
Attaching the TimesTen database to the cache grid

Before you can perform operations on a global cache group or on its cache tables, you must attach the TimesTen database to the cache grid that it associated with.

As the cache manager user, use the ttIsq1 utility to call the ttGridAttach built-in procedure to attach the TimesTen database to the myGrid cache grid:

Command> call ttGridAttach(1,'alone1','mysys',5001);

In this example, alone1 is a name that is used to uniquely identify the grid member, mysys is the host name of the TimesTen system, and 5001 is the TCP/IP port for the cache agent.

Calling the ttGridAttach built-in procedure automatically starts the cache agent if it is not already running.

Although the example in this chapter contains only one standalone TimesTen database as the sole grid member, it can be extended to include additional grid members such as active standby pairs and other standalone TimesTen databases. See Chapter 6, "Creating Other Cache Grid Members", for details on how to create and add other members to an existing cache grid, and how data in a global cache group is shared among the grid members.

Performing operations on the read-only cache group

This section shows how to manually load the read-only cache group. Then it shows the TimesTen cache table being automatically refreshed with committed updates on the cached Oracle table.

Complete the following tasks to perform operations on the read-only cache group:

1. Manually load the cache group
2. Update the cached Oracle table

Manually load the cache group

As the cache manager user, use the ttIsq1 utility to load the contents of the Oracle oratt.readtab table into the TimesTen oratt.readtab cache table in the readcache cache group:

Command> LOAD CACHE GROUP readcache COMMIT EVERY 256 ROWS;
2 cache instances affected.
Command> exit

Start the ttIsq1 utility and connect to the cachealone1 DSN as the instance administrator. Use ttIsq1 to grant the SELECT privilege on the oratt.readtab cache table to the cache manager user so that this user can issue a SELECT query on this table.

% ttIsq1 cachealone1
Command> GRANT SELECT ON oratt.readtab TO cacheuser;
Command> exit

Start the ttIsq1 utility and connect to the cachealone1 DSN as the cache manager user. Use ttIsq1 to query the contents of oratt.readtab cache table.

% ttIsq1 "DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> SELECT * FROM oratt.readtab;
< 1, Hello >
Performing operations on the dynamic updatable global cache group

Getting Started
2-11

< 2, World >
2 rows found.

See "Loading and refreshing a cache group" on page 5-2 for more information about manually loading a cache group.

Update the cached Oracle table

Use SQL*Plus, as the Oracle schema user, to insert a new row, delete an existing row, and update an existing row in the Oracle readtab table, and commit the changes:

SQL> INSERT INTO readtab VALUES (3, 'Welcome');
SQL> DELETE FROM readtab WHERE keyval=2;
SQL> UPDATE readtab SET str='Hi' WHERE keyval=1;
SQL> COMMIT;

After 5 seconds, the oratt.readtab cache table in the readcache cache group is automatically refreshed with the committed updates on the cached Oracle oratt.readtab table as shown in Figure 2–7.

Figure 2–7  Automatically refresh the TimesTen cache table with Oracle updates

As the cache manager user, use the ttIsq1 utility to query the contents of the oratt.readtab cache table after the readcache cache group has been automatically refreshed with the committed updates on the cached Oracle table:

Command> SELECT * FROM oratt.readtab;
< 1, Hi >
< 3, Welcome >
2 rows found.
Command> exit

See "AUTOREFRESH cache group attribute" on page 4-21 for more information about automatically refreshing cache groups.

Performing operations on the dynamic updatable global cache group

This section shows how to dynamically load the AWT cache group. Then it shows committed updates on the TimesTen cache table being automatically propagated to the cached Oracle table.
Performing operations on the dynamic updatable global cache group

Complete the following tasks to perform operations on the AWT cache group:

1. **Dynamically load the cache group**
2. **Update the TimesTen cache table**

### Dynamically load the cache group

Start the `ttIsql` utility and connect to the `cachealone1` DSN as the instance administrator. Use `ttIsql` to grant the SELECT privilege on the `oratt.writetab` cache table to the cache manager user so that this user can issue a dynamic load SELECT statement on this table.

```bash
% ttIsql cachealone1
Command> GRANT SELECT ON oratt.writetab TO cacheuser;
Command> exit
```

Start the `ttIsql` utility and connect to the `cachealone1` DSN as the cache manager user. Use `ttIsql` to load a cache instance on demand from the Oracle `oratt.writetab` table to the TimesTen `oratt.writetab` cache table in the `writecache` cache group.

```bash
% ttIsql "DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> SELECT * FROM oratt.writetab WHERE pk=100;
< 100, TimesTen >
1 row found.
Command> exit
```

In a dynamic cache group, a cache instance can be loaded into its cache tables on demand with a dynamic load statement. A SELECT, UPDATE, DELETE or INSERT statement issued on a TimesTen cache table that uniquely identifies a cache instance results in the cache instance being automatically loaded from the cached Oracle table if the data is not found in the cache table. A dynamically loaded cache instance consists of a single row in the root table of the cache group, and all the related rows in the child tables.

See "**Dynamically loading a cache group**" on page 5-10 for more information about a dynamic load operation.

Data can also be manually loaded into the cache tables of a dynamic cache group using a LOAD CACHE GROUP statement.

### Update the TimesTen cache table

Start the `ttIsql` utility and connect to the `cachealone1` DSN as the instance administrator. Use `ttIsql` to grant the INSERT, DELETE and UPDATE privileges on the `oratt.writetab` cache table to the cache manager user so that this user can perform updates on this table.

```bash
% ttIsql cachealone1
Command> GRANT INSERT ON oratt.writetab TO cacheuser;
Command> GRANT DELETE ON oratt.writetab TO cacheuser;
Command> GRANT UPDATE ON oratt.writetab TO cacheuser;
Command> exit
```

Start the `ttIsql` utility and connect to the `cachealone1` DSN as the cache manager user. Use `ttIsql` to insert a new row, delete an existing row, and update an existing row in the `oratt.writetab` cache table, and commit the changes.

```bash
% ttIsql "DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> INSERT INTO oratt.writetab VALUES (102, 'Cache');
```
Cleaning up the TimesTen and Oracle systems

```
Command> DELETE FROM oratt.writetab WHERE pk=101;
Command> UPDATE oratt.writetab SET attr='Oracle' WHERE pk=100;
Command> COMMIT;
Command> exit
```

The committed updates on the `oratt.writetab` cache table in the `writecache` cache group are automatically propagated to the Oracle `oratt.writetab` table as shown in Figure 2–8.

**Figure 2–8  Automatically propagate TimesTen cache table updates to Oracle**

As the Oracle schema user, use SQL*Plus to query the contents of the `writetab` table:
```
SQL> SELECT * FROM writetab;
PK    ATTR
---------- -------------------------------
100    Oracle
102    Cache
```

```
SQL> exit
```

**Cleaning up the TimesTen and Oracle systems**

Complete the following tasks to restore the TimesTen and Oracle systems to their original state prior to creating a cache grid and cache groups:

1. Detach the TimesTen database from the cache grid
2. Stop the replication agent
3. Drop the cache groups
4. Destroy the cache grid
5. Stop the cache agent and destroy the TimesTen database
6. Drop the Oracle users and their objects
Cleaning up the TimesTen and Oracle systems

Detach the TimesTen database from the cache grid

Start the ttIsql utility and connect to the cachealone1 DSN as the cache manager user. Use ttIsql to call the ttGridDetach built-in procedure to detach the TimesTen database from the myGrid cache grid.

% ttIsql "DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> call ttGridDetach;

See "Detaching a TimesTen database from a cache grid" on page 8-1 for information about the effects of detaching a TimesTen database from a cache grid.

Stop the replication agent

As the cache manager user, use the ttIsql utility to call the ttRepStop built-in procedure to stop the replication agent on the TimesTen database:

Command> call ttRepStop;
Command> exit

See "Managing the replication agent" on page 4-10 for more information about stopping the replication agent.

Drop the cache groups

Start the ttIsql utility and connect to the cachealone1 DSN as the instance administrator. Use ttIsql to grant the DROP ANY TABLE privilege to the cache manager user so that this user can drop the underlying cache tables when dropping the cache groups.

% ttIsql cachealone1
Command> GRANT DROP ANY TABLE TO cacheuser;
Command> exit

Start the ttIsql utility and connect to the cachealone1 DSN as the cache manager user. Use ttIsql to drop the readcache read-only cache group and the writecache AWT cache group.

% ttIsql "DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> DROP CACHE GROUP readcache;
Command> DROP CACHE GROUP writecache;

The cache groups readcache and writecache, and their respective cache tables oratt.readtab and oratt.writetab are dropped from the TimesTen database.

See "Dropping a cache group" on page 8-2 for more information about dropping cache groups.

Destroy the cache grid

As the cache manager user, use the ttIsql utility to call the ttGridDestroy built-in procedure to destroy the myGrid cache grid:

Command> call ttGridDestroy('myGrid');

See "Destroying a cache grid" on page 8-4 for more information about destroying a cache grid.
Stop the cache agent and destroy the TimesTen database

As the cache manager user, use the `ttIsql` utility to call the `ttCacheStop` built-in procedure to stop the cache agent on the TimesTen database:

```
Command> call ttCacheStop;
Command> exit
```

See "Managing the cache agent" on page 3-17 for more information about stopping the cache agent.

Then use the `ttDestroy` utility to connect to the `cachealone1` DSN and destroy the TimesTen database:

```
% ttDestroy cachealone1
```

Drop the Oracle users and their objects

Start SQL*Plus and connect to the Oracle database as the `sys` user. Use SQL*Plus to drop the `timesten` user, the schema user `oratt`, and the cache administration user `cacheuser`.

```
% sqlplus sys as sysdba
Enter password: password
SQL> DROP USER timesten CASCADE;
SQL> DROP USER oratt CASCADE;
SQL> DROP USER cacheuser CASCADE;
```

Specifying `CASCADE` in a `DROP USER` statement drops all objects such as tables and triggers owned by the user before dropping the user itself.

Next use SQL*Plus to drop the `TT_CACHE_ADMIN_ROLE` role:

```
SQL> DROP ROLE tt_cache_admin_role;
```

Then use SQL*Plus to drop the default tablespace `cachetblsp` used by the `timesten` user and cache administration user including the contents of the tablespace and its data file:

```
SQL> DROP TABLESPACE cachetblsp INCLUDING CONTENTS AND DATAFILES;
SQL> exit
```
Procedure for caching Oracle data in TimesTen

Table 2–1 Instructions for caching Oracle data in a TimesTen database

<table>
<thead>
<tr>
<th>Task number</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create the following users in the Oracle database:</td>
</tr>
<tr>
<td></td>
<td>■ timesten user</td>
</tr>
<tr>
<td></td>
<td>User is created by running the SQL*Plus script TimesTen_install_dir/oraclescripts/initCacheGlobalSchema.sql as the sys user.</td>
</tr>
<tr>
<td></td>
<td>■ One or more schema users who own the cached Oracle tables (may be existing users)</td>
</tr>
<tr>
<td></td>
<td>■ Cache administration user</td>
</tr>
<tr>
<td></td>
<td>A default tablespace must be created for the timesten user and the cache administration user. The same tablespace can be designated for both users.</td>
</tr>
<tr>
<td></td>
<td>Execute CREATE USER statements as the sys user.</td>
</tr>
<tr>
<td></td>
<td>See “Create the Oracle users” on page 3-2 for more information about the Oracle users.</td>
</tr>
<tr>
<td></td>
<td>Grant the cache administration user the privileges required to perform the cache grid operations, create the desired types of cache groups, and perform operations on the cache groups. Privileges are granted by running either the TimesTen_install_dir/oraclescripts/grantCacheAdminPrivileges.sql or the TimesTen_install_dir/oraclescripts/initCacheAdminSchema.sql script as the sys user.</td>
</tr>
<tr>
<td></td>
<td>See “Automatically create Oracle objects used to manage caching of Oracle data” on page 3-9 or “Manually create Oracle objects used to manage caching of Oracle data” on page 3-10 to determine the appropriate script to run.</td>
</tr>
<tr>
<td></td>
<td>If you are manually creating the Oracle objects, you also need to run the TimesTen_install_dir/oraclescripts/initCacheGridSchema.sql script to create the Oracle tables used to store information about TimesTen databases that are associated with a particular cache grid.</td>
</tr>
<tr>
<td></td>
<td>Some privileges cannot be granted until the cached Oracle tables have been created. To grant these privileges, execute GRANT statements as the sys user.</td>
</tr>
<tr>
<td></td>
<td>See “Grant privileges to the Oracle users” on page 3-3 for more information about the privileges that must be granted to the cache administration user to perform particular cache operations.</td>
</tr>
<tr>
<td>2</td>
<td>Define a DSN that references the TimesTen database that will be used to cache data from an Oracle database.</td>
</tr>
<tr>
<td></td>
<td>Set the OracleNetServiceName attribute to the Oracle net service name that references the Oracle database instance.</td>
</tr>
<tr>
<td></td>
<td>Set the DatabaseCharacterSet attribute to the Oracle database character set. The TimesTen database character set must match the Oracle database character set.</td>
</tr>
<tr>
<td></td>
<td>Then connect to the DSN to create the database if this is a standalone database or will be an active master database of an active standby pair.</td>
</tr>
<tr>
<td></td>
<td>See “Define a DSN for the TimesTen database” on page 3-12 for more information about defining a DSN for a TimesTen database that will be used to cache data from an Oracle database.</td>
</tr>
</tbody>
</table>
### Table 2–1 (Cont.) Instructions for caching Oracle data in a TimesTen database

<table>
<thead>
<tr>
<th>Task number</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Create the following users in the TimesTen database:</td>
</tr>
<tr>
<td></td>
<td>■ Cache manager user</td>
</tr>
<tr>
<td></td>
<td>- This user must have the same name as an Oracle user that can access the cached Oracle tables. The Oracle user can be the cache administration user, a schema user, or some other existing user. The password of the cache manager user and the Oracle user with the same name can be different.</td>
</tr>
<tr>
<td></td>
<td>■ One or more cache table users who own the TimesTen cache tables</td>
</tr>
<tr>
<td></td>
<td>- These users must have the same name as the Oracle schema users who own the cached Oracle tables. The password of a cache table user and the Oracle user with the same name can be different.</td>
</tr>
<tr>
<td></td>
<td>Execute CREATE USER statements as the instance administrator.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Create the TimesTen users&quot; on page 3-13 for more information about the TimesTen users.</td>
</tr>
<tr>
<td></td>
<td>Grant the cache manager user the privileges required to perform the cache grid operations, create the desired types of cache groups, and perform operations on the cache groups. Execute GRANT statements as the instance administrator.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Grant privileges to the TimesTen users&quot; on page 3-13 for more information about the privileges that must be granted to the cache manager user to perform particular cache operations.</td>
</tr>
<tr>
<td>4</td>
<td>Set the cache administration user name and password in the TimesTen database either by calling the ttCacheUidPwdSet built-in procedure as the cache manager user or running a ttAdmin -cacheUidPwdSet utility command as a TimesTen external user with the CACHE_MANAGER privilege.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Set the cache administration user name and password&quot; on page 3-14 for more information about setting the cache administration user name and password in a TimesTen database.</td>
</tr>
<tr>
<td>5</td>
<td>Create a cache grid by calling the ttGridCreate built-in procedure in the TimesTen database as the cache manager user.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Create a cache grid&quot; on page 3-16 for more information about creating a cache grid.</td>
</tr>
<tr>
<td>6</td>
<td>Associate the TimesTen database with the cache grid by calling the ttGridNameSet built-in procedure in the TimesTen database as the cache manager user.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Associate a TimesTen database with a cache grid&quot; on page 3-17 for more information about associating a TimesTen database with a cache grid.</td>
</tr>
<tr>
<td>7</td>
<td>Start the cache agent on the TimesTen database either by calling the ttCacheStart built-in procedure as the cache manager user or running a ttAdmin -cacheStart utility command as a TimesTen external user with the CACHE_MANAGER privilege.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Managing the cache agent&quot; on page 3-17 for more information about starting a cache agent on a TimesTen database.</td>
</tr>
</tbody>
</table>
Procedure for caching Oracle data in TimesTen

8 Design the schema for the cache groups by determining which Oracle tables to cache and within those tables, which columns and rows to cache. For multiple table cache groups, determine the relationship between the tables by defining which table is the root table, which tables are direct child tables of the root table, and which tables are the child tables of other child tables. For each cached column, determine the TimesTen data type to which the Oracle data type should be mapped.

See “Mappings between Oracle and TimesTen data types” on page 11-7 for a list of valid data type mappings between the Oracle and TimesTen databases.

For each cache group, determine what type to create (read-only, SWT, AWT, user managed) based on the application requirements and objectives. Also, determine whether each cache group will be explicitly loaded or dynamic, and local or global.

Then create the cache groups.

See “Creating a cache group” on page 4-5 for more information about creating a cache group.

9 If this TimesTen database is intended to be an active master database of an active standby pair, create an active standby pair replication scheme in the database.

10 If the TimesTen database contains an active standby pair replication scheme or at least one AWT cache group, start the replication agent on the database either by calling the ttRepStart built-in procedure as the cache manager user or running a ttAdmin -repStart utility command as a TimesTen external user with the CACHE_MANAGER privilege.

See “Managing the replication agent” on page 4-10 for more information about starting a replication agent on a TimesTen database.

11 If the TimesTen database contains at least one global cache group, attach the TimesTen database to the cache grid that the database associated with by calling the ttGridAttach built-in procedure as the cache manager user.

See “Attach a TimesTen database to a cache grid” on page 4-39 for more information about attaching a TimesTen database to a cache grid.

12 Manually load the cache tables in explicitly loaded cache groups using LOAD CACHE GROUP statements, and load the cache tables in dynamic cache groups using proper SELECT, UPDATE or INSERT statements.

See “Loading and refreshing a cache group” on page 5-2 for more information about manually loading cache tables in a cache group.

See “Dynamically loading a cache group” on page 5-10 for more information about dynamically loading cache tables in a dynamic cache group.

13 Subsequent standalone TimesTen databases can be added as members to an existing cache grid.

To create a standalone database, perform task 2. Then perform tasks 3 to 4, 6 to 8, and 10 to 11 to configure the database and add it as a member to the grid.

See ”Creating and configuring a subsequent standalone TimesTen database” on page 6-1 for details about creating another standalone TimesTen database and adding that database to an existing cache grid.
An active standby pair can be added as a member to an existing cache grid so that the cache tables can be replicated to another TimesTen database for high availability.

To create the active master database perform task 2. Then perform tasks 3 to 4, and 6 to 11 to configure the database and add it as a member to the grid.

See "Create and configure the active master database" on page 6-3 for details about creating an active master database and adding the database to an existing cache grid.

To create the standby master database from the active master database, perform task 2 to create a DSN for the standby master database, and then run a ttRepAdmin -duplicate utility command on the standby master database system as a TimesTen external user with the ADMIN privilege. For the command to succeed, the cache manager user in the active master database must be granted the ADMIN privilege. Then perform tasks 4, 7, 10 and 11 to configure the database and add it as a member to the grid.

See "Create and configure the standby master database" on page 6-5 for details about creating a standby master database and adding the database to an existing cache grid.

To create an optional read-only subscriber database from the standby master database, perform task 2 to create a DSN for the subscriber database. Then run a ttRepAdmin -duplicate utility command on the subscriber database system as a TimesTen external user with the ADMIN privilege. For the command to succeed, the cache manager user in the standby master database must be granted the ADMIN privilege. Then perform task 10 to start the replication agent on the database.

See "Create and configure the read-only subscriber database" on page 6-6 for details about creating a read-only subscriber database for an active standby pair.

---

**Table 2–1 (Cont.) Instructions for caching Oracle data in a TimesTen database**

<table>
<thead>
<tr>
<th>Task number</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>An active standby pair can be added as a member to an existing cache grid so that the cache tables can be replicated to another TimesTen database for high availability. To create the active master database perform task 2. Then perform tasks 3 to 4, and 6 to 11 to configure the database and add it as a member to the grid. See &quot;Create and configure the active master database&quot; on page 6-3 for details about creating an active master database and adding the database to an existing cache grid. To create the standby master database from the active master database, perform task 2 to create a DSN for the standby master database, and then run a ttRepAdmin -duplicate utility command on the standby master database system as a TimesTen external user with the ADMIN privilege. For the command to succeed, the cache manager user in the active master database must be granted the ADMIN privilege. Then perform tasks 4, 7, 10 and 11 to configure the database and add it as a member to the grid. See &quot;Create and configure the standby master database&quot; on page 6-5 for details about creating a standby master database and adding the database to an existing cache grid. To create an optional read-only subscriber database from the standby master database, perform task 2 to create a DSN for the subscriber database. Then run a ttRepAdmin -duplicate utility command on the subscriber database system as a TimesTen external user with the ADMIN privilege. For the command to succeed, the cache manager user in the standby master database must be granted the ADMIN privilege. Then perform task 10 to start the replication agent on the database. See &quot;Create and configure the read-only subscriber database&quot; on page 6-6 for details about creating a read-only subscriber database for an active standby pair.</td>
</tr>
</tbody>
</table>
This chapter describes the tasks for setting up the TimesTen and Oracle systems before you can start caching Oracle data in a TimesTen database. It includes the following topics:

- Configuring your system to cache Oracle data in TimesTen
- Configuring the Oracle database to cache data in TimesTen
- Configuring a TimesTen database to cache Oracle data
- Configuring a cache grid
- Testing the connectivity between the TimesTen and Oracle databases
- Managing the cache agent

### Configuring your system to cache Oracle data in TimesTen

Oracle In-Memory Database Cache supports the following Oracle server releases:

- Oracle 11g Release 2
- Oracle 11g Release 1
- Oracle 10g Release 2 (10.2.0.4.0 or above)

Configure the environment variables for your particular operating system, as described in "Oracle In-Memory Database Cache environment variables for UNIX" on page 3-1 or "Oracle In-Memory Database Cache environment variables for Microsoft Windows" on page 3-2.

Then install TimesTen as described in Oracle TimesTen In-Memory Database Installation Guide.

Note: From a product perspective, "Oracle In-Memory Database Cache" is used interchangeably with "TimesTen" because the Oracle In-Memory Database Cache product option includes the Oracle TimesTen In-Memory Database.

TimesTen does not support Oracle Name Server for Windows clients.

### Oracle In-Memory Database Cache environment variables for UNIX

The shared library search path environment variable such as LD_LIBRARY_PATH or SHLIB_PATH must include the TimesTen_install_dir/lib directory.
For more information, see "Shared library path environment variable" in *Oracle TimesTen In-Memory Database Installation Guide*.

The PATH environment variable must include the `TimesTen_install_dir/bin` directory.

In the following example, TimesTen is installed in the `/timesten/myinstance` directory:

```plaintext
LD_LIBRARY_PATH=/timesten/myinstance/lib
PATH=/timesten/myinstance/bin
```

**Oracle In-Memory Database Cache environment variables for Microsoft Windows**

The PATH system environment variable must include the following directories:

- `Oracle_install_dir\bin`
- `TimesTen_install_dir\lib`
- `TimesTen_install_dir\bin`

In the following example, Oracle is installed in the `C:\oracle\ora112` directory and TimesTen is installed in the `C:\timesten\myinstance` directory:

```plaintext
PATH=C:\oracle\ora112\bin;C:\timesten\myinstance\lib;C:\timesten\myinstance\bin
```

**Configuring the Oracle database to cache data in TimesTen**

This section describes the tasks that must be performed on the Oracle database by the sys user. The topics include:

- Create the Oracle users
- Grant privileges to the Oracle users
- Automatically create Oracle objects used to manage caching of Oracle data
- Manually create Oracle objects used to manage caching of Oracle data

**Create the Oracle users**

First you must create a user `timesten` that will own Oracle tables that store information about cache grids. The SQL*Plus script `TimesTen_install_dir/oraclescripts/initCacheGlobalSchema.sql` is used to create:

- The `timesten` user
- The Oracle tables owned by the `timesten` user to store information about cache grids
- The `TT_CACHE_ADMIN_ROLE` role that defines privileges on these Oracle tables

Create or designate a default tablespace for the `timesten` user and pass this tablespace as an argument to the `initCacheGlobalSchema.sql` script. See "Oracle objects used to manage a caching environment" on page 7-10 for a list of Oracle tables owned by the `timesten` user.

**Example 3-1 Creating the timesten user and its tables**

In the following SQL*Plus example, the default tablespace that is created for the `timesten` user is `cachetblsp`:

```sql
```
% cd TimesTen_install_dir/oraclescripts
% sqlplus sys as sysdba
      Enter password: password
SQL> CREATE TABLESPACE cachetblsp DATAFILE 'datfttuser.dbf' SIZE 100M;
SQL> @initCacheGlobalSchema "cachetblsp"

Then you must create or designate one or more users that will own Oracle tables that
will be cached in a TimesTen database. We refer to these users as the schema users.
These may be existing users or new users. The tables to be cached may or may not
already exist.

Example 3-2 Creating a schema user
As the sys user, create a schema user oratt.

Use SQL*Plus to create the schema user:

SQL> CREATE USER oratt IDENTIFIED BY oracle;

Next you must create a user that will create, own and maintain Oracle objects that
store information used to manage a specific cache grid and enforce predefined
behaviors of particular cache group types. We refer to this user as the cache
administration user.

Designate the tablespace that was created for the timesten user as the default
tablespace for the cache administration user. This user will create tables in this
tablespace that are used to store information about the cache grid and its cache groups.
Other Oracle objects such change log tables, replication metadata tables, and triggers
that are used to enforce the predefined behaviors of automatic refresh cache groups
and AWT cache groups are created in the same tablespace.

See "Oracle objects used to manage a caching environment" on page 7-10 for a list of
Oracle tables and triggers owned by the cache administration user.

Note: An automatic refresh cache group refers to a read-only cache
group or a user managed cache group that uses the AUTOREFRESH
MODE INCREMENTAL cache group attribute.

Example 3-3 Creating the cache administration user
As the sys user, create a cache administration user cacheuser. In the following
example, the default tablespace for the cacheuser user is cachetblsp.

Use SQL*Plus to create the cache administration user:

SQL> CREATE USER cacheuser IDENTIFIED BY oracle
         DEFAULT TABLESPACE cachetblsp QUOTA UNLIMITED ON cachetblsp;

Grant privileges to the Oracle users

The privileges that the Oracle users require depends on the types of cache groups you
create and the operations that you perform on the cache groups. The privileges
required for the Oracle cache administration user and the TimesTen cache manager
user for each cache operation are listed in Table 3-1.
<table>
<thead>
<tr>
<th>Cache operation</th>
<th>Privileges required for Oracle cache administration user¹</th>
<th>Privileges required for TimesTen cache manager user²</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set the cache administration user name and password</td>
<td>• CREATE SESSION</td>
<td>• CACHE_MANAGER</td>
</tr>
<tr>
<td></td>
<td>• RESOURCE³</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Call the ttCacheUidPwdSet built-in procedure</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Run the ttAdmin -cacheUidPwdSet utility command</td>
<td></td>
</tr>
<tr>
<td>Get the cache administration user name</td>
<td>None</td>
<td>• CACHE_MANAGER</td>
</tr>
<tr>
<td></td>
<td>• Call the ttCacheUidGet built-in procedure</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Run the ttAdmin -cacheUidGet utility command</td>
<td></td>
</tr>
<tr>
<td>Create a cache grid</td>
<td>• CREATE SESSION</td>
<td>• CACHE_MANAGER</td>
</tr>
<tr>
<td></td>
<td>• TT_CACHE_ADMIN_ROLE role</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• RESOURCE³</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Call the ttGridCreate built-in procedure</td>
<td></td>
</tr>
<tr>
<td>Associate a TimesTen database with a cache grid</td>
<td>• CREATE SESSION</td>
<td>• CACHE_MANAGER</td>
</tr>
<tr>
<td></td>
<td>• tt_cache_admin_role role</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Call the ttGridNameSet built-in procedure</td>
<td></td>
</tr>
<tr>
<td>Attach a TimesTen database to a cache grid</td>
<td>• CREATE SESSION</td>
<td>• CACHE_MANAGER</td>
</tr>
<tr>
<td></td>
<td>• TT_CACHE_ADMIN_ROLE role</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Call the ttGridAttach built-in procedure</td>
<td></td>
</tr>
<tr>
<td>Detach a TimesTen database from a cache grid</td>
<td>• CREATE SESSION</td>
<td>• CACHE_MANAGER</td>
</tr>
<tr>
<td></td>
<td>• TT_CACHE_ADMIN_ROLE role</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Call the ttGridDetach built-in procedure</td>
<td></td>
</tr>
<tr>
<td>Detach a list of nodes from a cache grid</td>
<td>• CREATE SESSION</td>
<td>• CACHE_MANAGER</td>
</tr>
<tr>
<td></td>
<td>• TT_CACHE_ADMIN_ROLE role</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Call the ttGridDetachList built-in procedure</td>
<td></td>
</tr>
<tr>
<td>Destroy a cache grid</td>
<td>• CREATE SESSION</td>
<td>• CACHE_MANAGER</td>
</tr>
<tr>
<td></td>
<td>• TT_CACHE_ADMIN_ROLE role</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Call the ttGridDestroy built-in procedure</td>
<td></td>
</tr>
<tr>
<td>Start the cache agent</td>
<td>• CREATE SESSION</td>
<td>• CACHE_MANAGER</td>
</tr>
</tbody>
</table>

³ These are generic roles, required for administrative tasks.
### Table 3–1 (Cont.) Oracle and TimesTen user privileges required for cache operations

<table>
<thead>
<tr>
<th>Cache operation</th>
<th>Privileges required for Oracle cache administration user^1</th>
<th>Privileges required for TimesTen cache manager user^2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stop the cache agent</td>
<td>None</td>
<td>CACHE_MANAGER</td>
</tr>
<tr>
<td>■ Call the ttCacheStop built-in procedure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>■ Run the ttAdmin -cacheStop utility command</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set a cache agent start policy</td>
<td>CREATE SESSION^4</td>
<td>CACHE_MANAGER</td>
</tr>
<tr>
<td>■ Call the ttCachePolicySet built-in procedure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>■ Run the ttAdmin -cachePolicy utility command</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return the cache agent start policy setting</td>
<td>CREATE SESSION</td>
<td>None</td>
</tr>
<tr>
<td>■ Call the ttCachePolicyGet built-in procedure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start the replication agent</td>
<td>None</td>
<td>CACHE_MANAGER</td>
</tr>
<tr>
<td>■ Call the ttRepStart built-in procedure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>■ Run the ttAdmin -repStart utility command</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stop the replication agent</td>
<td>None</td>
<td>CACHE_MANAGER</td>
</tr>
<tr>
<td>■ Call the ttRepStop built-in procedure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>■ Run the ttAdmin -repStop utility command</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set a replication agent start policy</td>
<td>None</td>
<td>ADMIN</td>
</tr>
<tr>
<td>■ Call the ttRepPolicySet built-in procedure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>■ Run the ttAdmin -repPolicy utility command</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CREATE [DYNAMIC] READONLY CACHE GROUP with AUTOREFRESH MODE INCREMENTAL</td>
<td>CREATE SESSION</td>
<td>CREATE [ANY] CACHE GROUP^6</td>
</tr>
<tr>
<td>■ SELECT ON table_name^5</td>
<td></td>
<td>CREATE [ANY] TABLE^7</td>
</tr>
<tr>
<td>■ RESOURCE^3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>■ CREATE ANY TRIGGER^3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CREATE [DYNAMIC] READONLY CACHE GROUP with AUTOREFRESH MODE FULL</td>
<td>CREATE SESSION</td>
<td>CREATE [ANY] CACHE GROUP^6</td>
</tr>
<tr>
<td>■ SELECT ON table_name^5</td>
<td></td>
<td>CREATE [ANY] TABLE^7</td>
</tr>
<tr>
<td>■ RESOURCE^3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>■ CREATE ANY TRIGGER^3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 3–1 (Cont.) Oracle and TimesTen user privileges required for cache operations

<table>
<thead>
<tr>
<th>Cache operation</th>
<th>Privileges required for Oracle cache administration user¹</th>
<th>Privileges required for TimesTen cache manager user²</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE [DYNAMIC] SYNCHRONOUS WRITETHROUGH CACHE GROUP</td>
<td>CREATE SESSION, SELECT ON <code>table_name</code>, RESOURCE³</td>
<td>CREATE [ANY] CACHE GROUP, CREATE [ANY] TABLE⁷</td>
</tr>
<tr>
<td>CREATE [DYNAMIC] USERMANAGED CACHE GROUP</td>
<td>CREATE SESSION, SELECT ON <code>table_name</code>, RESOURCE³</td>
<td>CREATE [ANY] CACHE GROUP, CREATE [ANY] TABLE⁷</td>
</tr>
<tr>
<td>(see variants in following rows)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CREATE [DYNAMIC] USERMANAGED CACHE GROUP with AUTOREFRESH MODE INCREMENTAL</td>
<td>CREATE SESSION, SELECT ON <code>table_name</code>, RESOURCE³, CREATE ANY TRIGGER³</td>
<td>CREATE [ANY] CACHE GROUP, CREATE [ANY] TABLE⁷</td>
</tr>
<tr>
<td>CREATE [DYNAMIC] USERMANAGED CACHE GROUP with AUTOREFRESH MODE FULL</td>
<td>CREATE SESSION, SELECT ON <code>table_name</code>, RESOURCE³</td>
<td>CREATE [ANY] CACHE GROUP, CREATE [ANY] TABLE⁷</td>
</tr>
<tr>
<td>CREATE [DYNAMIC] USERMANAGED CACHE GROUP with READONLY</td>
<td>CREATE SESSION, SELECT ON <code>table_name</code>, RESOURCE³</td>
<td>CREATE [ANY] CACHE GROUP, CREATE [ANY] TABLE⁷</td>
</tr>
<tr>
<td>CREATE [DYNAMIC] USERMANAGED CACHE GROUP with PROPAGATE</td>
<td>CREATE SESSION, SELECT ON <code>table_name</code>, RESOURCE³</td>
<td>CREATE [ANY] CACHE GROUP, CREATE [ANY] TABLE⁷</td>
</tr>
<tr>
<td>ALTER CACHE GROUP SET AUTOREFRESH STATE PAUSED</td>
<td>CREATE SESSION, SELECT ON <code>table_name</code>, RESOURCE³, CREATE ANY TRIGGER³</td>
<td>ALTER ANY CACHE GROUP⁹</td>
</tr>
<tr>
<td>ALTER CACHE GROUP SET AUTOREFRESH STATE ON</td>
<td>CREATE SESSION, SELECT ON <code>table_name</code>, RESOURCE³, CREATE ANY TRIGGER³</td>
<td>ALTER ANY CACHE GROUP⁹</td>
</tr>
<tr>
<td>ALTER CACHE GROUP SET AUTOREFRESH STATE OFF</td>
<td>CREATE SESSION, SELECT ON <code>table_name</code>, RESOURCE³, CREATE ANY TRIGGER³</td>
<td>ALTER ANY CACHE GROUP⁹</td>
</tr>
<tr>
<td>ALTER CACHE GROUP SET AUTOREFRESH MODE FULL</td>
<td>CREATE SESSION, SELECT ON <code>table_name</code>, RESOURCE³, CREATE ANY TRIGGER³</td>
<td>ALTER ANY CACHE GROUP⁹</td>
</tr>
</tbody>
</table>
### Table 3–1 (Cont.) Oracle and TimesTen user privileges required for cache operations

<table>
<thead>
<tr>
<th>Cache operation</th>
<th>Privileges required for Oracle cache administration user</th>
<th>Privileges required for TimesTen cache manager user</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER CACHE GROUP SET AUTOREFRESH MODE INCREMENTAL</td>
<td>• CREATE SESSION</td>
<td>• ALTER ANY CACHE GROUP</td>
</tr>
<tr>
<td></td>
<td>• SELECT ON <code>table_name</code>^5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• RESOURCE^3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• CREATE ANY TRIGGER^3</td>
<td></td>
</tr>
<tr>
<td>ALTER CACHE GROUP SET AUTOREFRESH INTERVAL</td>
<td>• CREATE SESSION</td>
<td>• ALTER ANY CACHE GROUP</td>
</tr>
<tr>
<td></td>
<td>• SELECT ON <code>table_name</code>^5,10</td>
<td></td>
</tr>
<tr>
<td>LOAD CACHE GROUP</td>
<td>• CREATE SESSION</td>
<td>• LOAD [ANY CACHE GROUP</td>
</tr>
<tr>
<td></td>
<td>• SELECT ON <code>table_name</code>^5</td>
<td></td>
</tr>
<tr>
<td>REFRESH CACHE GROUP</td>
<td>• CREATE SESSION</td>
<td>• REFRESH [ANY CACHE GROUP</td>
</tr>
<tr>
<td></td>
<td>• SELECT ON <code>table_name</code>^5</td>
<td></td>
</tr>
<tr>
<td>FLUSH CACHE GROUP</td>
<td>• CREATE SESSION</td>
<td>• FLUSH [ANY CACHE GROUP</td>
</tr>
<tr>
<td></td>
<td>• UPDATE ON <code>table_name</code>^5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• INSERT ON <code>table_name</code>^5</td>
<td></td>
</tr>
<tr>
<td>UNLOAD CACHE GROUP</td>
<td>None</td>
<td>• UNLOAD [ANY CACHE GROUP</td>
</tr>
<tr>
<td>DROP CACHE GROUP</td>
<td>• CREATE SESSION</td>
<td>• DROP ANY CACHE GROUP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DROP ANY TABLE^11</td>
</tr>
<tr>
<td>Synchronous writethrough or propagate</td>
<td>• CREATE SESSION</td>
<td>• INSERT ON <code>table_name</code>^13</td>
</tr>
<tr>
<td></td>
<td>• INSERT ON <code>table_name</code>^5,12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• UPDATE ON <code>table_name</code>^5,12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• DELETE ON <code>table_name</code>^5,12</td>
<td></td>
</tr>
<tr>
<td>Asynchronous writethrough</td>
<td>• CREATE SESSION</td>
<td>• INSERT ON <code>table_name</code>^13</td>
</tr>
<tr>
<td></td>
<td>• INSERT ON <code>table_name</code>^5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• UPDATE ON <code>table_name</code>^5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• DELETE ON <code>table_name</code>^5</td>
<td></td>
</tr>
<tr>
<td>Asynchronous writethrough when the CacheAWTMethod attribute is set to 1</td>
<td>• CREATE PROCEDURE</td>
<td>No additional privileges</td>
</tr>
<tr>
<td>Note: This privilege is an addition to the privileges needed for any asynchronous writethrough cache group.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 3–1  (Cont.) Oracle and TimesTen user privileges required for cache operations

<table>
<thead>
<tr>
<th>Cache operation</th>
<th>Privileges required for Oracle cache administration user</th>
<th>Privileges required for TimesTen cache manager user</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asynchronous writethrough cache for Oracle CLOB, BLOB and NCLOB fields when the <code>CacheAWTMethod</code> attribute is set to 1</td>
<td>EXECUTE privilege on the Oracle DBMS_LOB PL/SQL package</td>
<td>No additional privileges</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: This privilege is an addition to the privileges needed for any asynchronous writethrough cache group.</td>
<td></td>
</tr>
<tr>
<td>Incremental automatic refresh</td>
<td>SELECT ON <code>table_name</code></td>
<td>None</td>
</tr>
<tr>
<td>Full automatic refresh</td>
<td>SELECT ON <code>table_name</code></td>
<td>None</td>
</tr>
<tr>
<td>Dynamic load</td>
<td>CREATE SESSION</td>
<td>SELECT ON <code>table_name</code></td>
</tr>
<tr>
<td></td>
<td>SELECT ON <code>table_name</code></td>
<td>UPDATE ON <code>table_name</code></td>
</tr>
<tr>
<td></td>
<td>SELECT ON <code>table_name</code></td>
<td>DELETE ON <code>table_name</code></td>
</tr>
<tr>
<td></td>
<td>SELECT ON <code>table_name</code></td>
<td>INSERT ON <code>table_name</code></td>
</tr>
<tr>
<td>Aging</td>
<td>None</td>
<td>DELETE {ANY TABLE</td>
</tr>
<tr>
<td>Set the LRU aging attributes</td>
<td>None</td>
<td>ADMIN</td>
</tr>
<tr>
<td></td>
<td>Call the <code>ttAgingLRUConfig</code> built-in procedure</td>
<td></td>
</tr>
<tr>
<td>Generate Oracle SQL statements to manually install or uninstall Oracle objects</td>
<td>CREATE SESSION</td>
<td>CACHE_MANAGER</td>
</tr>
<tr>
<td></td>
<td>Run the <code>ttIsql</code> utility's <code>cachesqlget</code> command</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Call the <code>ttCacheSQLGet</code> built-in procedure</td>
<td></td>
</tr>
<tr>
<td>Disable or enable propagation of committed cache table updates to Oracle</td>
<td>None</td>
<td>CACHE_MANAGER</td>
</tr>
<tr>
<td></td>
<td>Call the <code>ttCachePropagateFlagSet</code> built-in procedure</td>
<td></td>
</tr>
<tr>
<td>Configure cache agent timeout and recovery method for automatic refresh cache groups</td>
<td>CREATE SESSION</td>
<td>CACHE_MANAGER</td>
</tr>
<tr>
<td></td>
<td>Call the <code>ttCacheConfig</code> built-in procedure</td>
<td></td>
</tr>
</tbody>
</table>
### Table 3–1 (Cont.) Oracle and TimesTen user privileges required for cache operations

<table>
<thead>
<tr>
<th>Cache operation</th>
<th>Privileges required for Oracle cache administration user¹</th>
<th>Privileges required for TimesTen cache manager user²</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set the AWT transaction log file threshold</td>
<td>None</td>
<td>CACHE_MANAGER</td>
</tr>
<tr>
<td>Call the <code>ttCacheAWTThresholdSet</code> built-in procedure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enable or disable monitoring of AWT cache groups</td>
<td>None</td>
<td>CACHE_MANAGER</td>
</tr>
<tr>
<td>Call the <code>ttCacheAWTMonitorConfig</code> built-in procedure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enable or disable tracking of DDL statements issued on cached Oracle tables</td>
<td>CREATE SESSION</td>
<td>CACHE_MANAGER</td>
</tr>
<tr>
<td>Call the <code>ttCacheDDLTrackingConfig</code> built-in procedure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return information about cache grids</td>
<td>CREATE SESSION</td>
<td>CACHE_MANAGER</td>
</tr>
<tr>
<td>Call the <code>ttGridInfo</code> built-in procedure</td>
<td>TT_CACHE_ADMIN_ROLE role</td>
<td></td>
</tr>
<tr>
<td>Return information about cache grid nodes</td>
<td>CREATE SESSION</td>
<td>CACHE_MANAGER</td>
</tr>
<tr>
<td>Call the <code>ttGridNodeStatus</code> built-in procedure</td>
<td>TT_CACHE_ADMIN_ROLE role</td>
<td></td>
</tr>
</tbody>
</table>

¹ At minimum, the cache administration user must have the CREATE ANY TYPE privilege.
² At minimum, the cache manager user must have the CREATE SESSION privilege.
³ Not required if the Oracle objects used to manage the caching of Oracle data are manually created.
⁴ Required if the cache agent start policy is being set to always or norestart.
⁵ Required on all Oracle tables cached in the TimesTen cache group except for tables owned by the cache administration user.
⁶ The CACHE_MANAGER privilege includes the CREATE [ANY] CACHE GROUP privilege. ANY is required if the cache manager user creates cache groups owned by a user other than itself.
⁷ ANY is required if any of the cache tables are owned by a user other than the cache manager user.
⁸ Required if the cache group’s automatic refresh mode is incremental and initial automatic refresh state is OFF, and the Oracle objects used to manage the caching of Oracle data are automatically created.
⁹ Required if the TimesTen user accessing the cache group does not own the cache group.
¹⁰ Required if the cache group’s automatic refresh mode is incremental.
¹¹ Required if the TimesTen user accessing the cache group does not own all its cache tables.
¹² The privilege must be granted to the Oracle user with the same name as the TimesTen cache manager user if the Oracle user is not the cache administration user.
¹³ Required if the TimesTen user accessing the cache table does not own the table.

### Automatically create Oracle objects used to manage caching of Oracle data

TimesTen can automatically create Oracle objects owned by the cache administration user, such as cache and replication metadata tables, change log tables, and triggers when particular cache grid and cache group operations are performed. Some of these objects are used to store information about TimesTen databases that are associated
with a particular cache grid. Other objects are used to enforce the predefined behaviors of automatic refresh cache groups and AWT cache groups.

These Oracle objects are automatically created if the cache administration user has been granted the required privileges by running the SQL*Plus script `TimesTen_install_dir/oraclescripts/grantCacheAdminPrivileges.sql` as the `sys` user. The set of required privileges include CREATE SESSION, RESOURCE, CREATE ANY TRIGGER, and the TT_CACHE_ADMIN_ROLE role. The cache administration user name is passed as an argument to the `grantCacheAdminPrivileges.sql` script.

In addition to the privileges granted to the cache administration user by running the `grantCacheAdminPrivileges.sql` script, this user may also need to be granted privileges such as SELECT or INSERT on the cached Oracle tables depending on the types of cache groups you create, and the operations that you perform on the cache groups and their cache tables. See Table 3–1 for a complete list of privileges that need to be granted to the cache administration user in order to perform particular cache grid, cache group, and cache table operations.

Example 3–4  Granting privileges to automatically create Oracle objects

As the `sys` user, run the `grantCacheAdminPrivileges.sql` script to grant privileges to the cache administration user to automatically create Oracle objects used to manage caching of Oracle data in a TimesTen database. In the following example, the cache administration user name is `cacheuser`.

Use SQL*Plus to run the `grantCacheAdminPrivileges.sql` script:

```sql
SQL> @grantCacheAdminPrivileges 'cacheuser'
SQL> exit
```

For example, with automatic refresh cache groups, the Oracle objects used to enforce the predefined behaviors of these cache group types are automatically created if the objects do not already exist and one of the following occurs:

- The cache group is created with its automatic refresh state set to PAUSED or ON
- The cache group is created with its automatic refresh state set to OFF and then altered to either PAUSED or ON

Manually create Oracle objects used to manage caching of Oracle data

The cache administration user requires the RESOURCE privilege to automatically create the Oracle objects used to:

- Store information about TimesTen databases that are associated with a particular cache grid
- Enforce the predefined behaviors of automatic refresh cache groups. In this case, the cache administration user also requires the CREATE ANY TRIGGER privilege to automatically create these Oracle objects.
- Enforce the predefined behaviors of AWT cache groups

For security purposes, if you do not want to grant the RESOURCE and CREATE ANY TRIGGER privileges to the cache administration user required to automatically create the Oracle objects, you can manually create these objects.

To manually create the Oracle tables and triggers used to enforce the predefined behaviors of particular cache group types, run the SQL*Plus script `TimesTen_install_dir/oraclescripts/initCacheAdminSchema.sql` as the
sys user. These objects must be created before you can create automatic refresh cache groups and AWT cache groups. The cache administration user name is passed as an argument to the initCacheAdminSchema.sql script.

The initCacheAdminSchema.sql script also grants a minimal set of required privileges including CREATE SESSION and the TT_CACHE_ADMIN_ROLE role to the cache administration user. In addition to the privileges granted to the cache administration user by running the initCacheAdminSchema.sql script, this user may also need to be granted privileges such as SELECT or INSERT on the cached Oracle tables depending on the types of cache groups you create and the operations that you perform on the cache groups and their cache tables. See Table 3–1 for a complete list of privileges that need to be granted to the cache administration user in order to perform particular cache grid, cache group, and cache table operations.

To manually create the Oracle tables used to store information about TimesTen databases that are associated with a particular cache grid, run the SQL*Plus script TimesTen_install_dir/oraclescripts/initCacheGridSchema.sql as the sys user. These tables must be created before you can create a cache grid. The cache administration user name and the name of the cache grid that you create are passed as arguments to the initCacheGridSchema.sql script.

Example 3–5 Manually creating Oracle objects used to manage caching of Oracle data

As the sys user, run the initCacheAdminSchema.sql script to manually create Oracle objects used to enforce the predefined behaviors of automatic refresh cache groups and AWT cache groups, and grant a limited set of privileges to the cache administration user. Then run the initCacheGridSchema.sql script to manually create Oracle objects used to store information about TimesTen databases associated with a particular cache grid. In the following example, the cache administration user name is cacheuser and the cache grid name is ttGrid.

Use SQL*Plus to run the initCacheAdminSchema.sql and initCacheGridSchema.sql scripts:

```
SQL> @initCacheAdminSchema "cacheuser"
SQL> @initCacheGridSchema "cacheuser" "ttGrid"
SQL> exit
```

Other Oracle objects associated with Oracle tables that are cached in an automatic refresh cache group are needed to enforce the predefined behaviors of these cache group types. See “Manually creating Oracle objects for automatic refresh cache groups” on page 4-24 for details about how to create these additional objects after you create the cache group.

To view a list of the Oracle objects created and used by TimesTen to manage the caching of Oracle data, execute the following query in SQL*Plus as the sys user:

```
SQL> SELECT owner, object_name, object_type FROM all_objects WHERE object_name 2  LIKE 'TT\___%' ESCAPE '\';
```

The query returns a list of tables, indexes and triggers owned by either the timesten user or the cache administration user.

**Configuring a TimesTen database to cache Oracle data**

This section describes the operations that must be performed on the TimesTen database by the instance administrator or the cache manager user. The topics include:

- Define a DSN for the TimesTen database
Create the TimesTen users

Grant privileges to the TimesTen users

Set the cache administration user name and password

Define a DSN for the TimesTen database

A TimesTen database that caches data from an Oracle database can be referenced by either a system DSN or a user DSN. See “Managing TimesTen Data Stores” in the Oracle TimesTen In-Memory Database Operations Guide for more information about creating TimesTen DSNs.

When creating a DSN for a TimesTen database that caches data from an Oracle database, pay special attention to the settings of the following attributes. All of these attributes can be set in a Data Manager DSN or a connection string, unless otherwise stated.

- **PermSize** specifies the allocated size of the database’s permanent partition in MB. Set this value to at least 32 MB.
- **OracleNetServiceName** must be set to the net service name of the Oracle database instance.
  
  On Microsoft Windows systems, the net service name of the Oracle database instance is specified in the Oracle Net Service Name field of the IMDB Cache tab within the TimesTen ODBC Setup dialog box.
- **DatabaseCharacterSet** must be set to the Oracle database character set.
  
  You can determine the Oracle database character set by executing the following query in SQL*Plus as any user:
  
  ```sql
  SQL> SELECT value FROM nls_database_parameters WHERE parameter='NLS_CHARACTERSET';
  ```
- **UID** specifies the name of a cache user, such as the cache manager user, that has the same name as an Oracle user who can access the cached Oracle tables. The **UID** attribute can be specified in a Data Manager DSN, a Client DSN, or a connection string.
- **PWD** specifies the password of the TimesTen user specified in the **UID** attribute. The **PWD** attribute can be specified in a Data Manager DSN, a Client DSN, or a connection string.
- **OraclePWD** specifies the password of the Oracle user that has the same name as the TimesTen user specified in the **UID** attribute and can access the cached Oracle tables.
- **PassThrough** can be set to control whether statements are to be executed in the TimesTen database or passed through to be executed in the Oracle database, as described in "Setting a passthrough level" on page 5-15.
- **LockLevel** must be set to its default of 0 (row-level locking) because Oracle In-Memory Database Cache does not support database-level locking.
- **TypeMode** must be set to its default of 0 (Oracle type mode).

**Example 3–6  DSN for a TimesTen database that caches data from an Oracle database**

The following example is the definition of the `cachealone1` DSN that references the first standalone TimesTen database that will become a member of the `ttGrid` cache grid:
Create the TimesTen users

First you must create a user who performs cache grid and cache group operations. We refer to this user as the cache manager user. This user must have the same name as an Oracle user that can select from and update the cached Oracle tables. The Oracle user can be the cache administration user, a schema user, or some other existing user. The password of the cache manager user can be different than the password of the Oracle user with the same name.

The cache manager user is responsible for creating and configuring the cache grid and creating the cache groups. This user can also monitor the grid itself and various operations that are performed on the cache groups.

Then you must create a user with the same name as an Oracle schema user for each schema user who owns or will own Oracle tables to be cached in the TimesTen database. We refer to these users as cache table users because the TimesTen cache tables will be owned by these users. Therefore, the owner and name of a TimesTen cache table is the same as the owner and name of the corresponding cached Oracle table. The password of a cache table user can be different than the password of the Oracle schema user with the same name.

Operations on a cache group or a cache table such as loading a cache group or updating a cache table can be performed by any TimesTen user that has sufficient privileges. In the examples throughout this guide, the cache manager user performs these types of operations although these operations can be performed by another user, such as a cache table user, that has the required privileges. If these operations are to be performed by a TimesTen user other than the cache manager user, the other user must have the same name as an Oracle user that can select from and update the cached Oracle tables. Connect to the TimesTen database specifying that user's name in the UID attribute, and supply the corresponding TimesTen and Oracle passwords in the PWD and OraclePWD attributes, respectively, to perform operations on a cache group or cache table.

**Example 3–7  Creating the TimesTen users**

In the following ttIsq1 utility example, create the TimesTen database by connecting to the cachealone1 DSN as the instance administrator. Then create the cache manager user cacheuser whose name, in this example, is the same as the Oracle cache administration user. Then create a cache table user oratt whose name is the same as the Oracle schema user who will own the Oracle tables to be cached in the TimesTen database.

% ttIsq1 cachealone1
Command> CREATE USER cacheuser IDENTIFIED BY timesten;
Command> CREATE USER oratt IDENTIFIED BY timesten;

Grant privileges to the TimesTen users

The privileges that the TimesTen users require depend on the types of cache groups you create and the operations that you perform on the cache groups. The privileges required for the TimesTen cache manager user and the Oracle cache administration user for each cache operation are listed in Table 3–1.
**Example 3–8  Granting privileges to the cache manager user**

The cache manager user cacheuser requires privileges to perform the following operations:

- Set the cache administration user and password (CACHE_MANAGER)
- Create and associate the TimesTen database with a cache grid (CACHE_MANAGER)
- Start the cache agent and replication agent processes on the TimesTen database (CACHE_MANAGER)
- Attach the TimesTen database to the cache grid (CACHE_MANAGER)
- Create cache groups to be owned by the cache administration user (CREATE CACHE GROUP, inherited by the CACHE_MANAGER privilege; CREATE ANY TABLE to create the underlying cache tables which will be owned by the oratt cache table user)

As the instance administrator, use the ttIsq1 utility to grant the cache manager user cacheuser the required privileges:

Command> GRANT CREATE SESSION, CACHE_MANAGER, CREATE ANY TABLE TO cacheuser;
Command> exit

**Set the cache administration user name and password**

You must set the cache administration user name and password in the TimesTen database before any cache grid or cache group operation can be issued. The cache agent connects to the Oracle database as this user to create and maintain Oracle objects that store information used to manage a cache grid and enforce predefined behaviors of particular cache group types.

The cache administration user name and password need to be set only once in each TimesTen database that will cache Oracle data unless it needs to be changed. For example, if the TimesTen database is destroyed and re-created, or the cache administration user name is dropped and re-created in the Oracle database, the cache administration user name and password must be set again.

The cache administration user name and password cannot be changed if the cache agent is running on the TimesTen database or there are cache groups in the database. The cache groups must be dropped before you can change the cache administration user name and password. You must also stop the cache agent before you change the cache administration user name and password, and then restart the cache agent after the user name and password have been changed.

**Example 3–9  Setting the cache administration user name and password**

The cache administration user name and password can be set programmatically by calling the ttCacheUidPwdSet built-in procedure as the cache manager user:

% ttIsq1 "DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> call ttCacheUidPwdSet('cacheuser','oracle');

It can also be set from a command line by running a ttAdmin -cacheUidPwdSet utility command as a TimesTen external user with the CACHE_MANAGER privilege:

% ttAdmin -cacheUidPwdSet -cacheUid cacheuser -cachePwd oracle cachealone1

If you do not specify the -cachePwd option, the ttAdmin utility prompts for the cache administration user's password.
Configuring a cache grid

An Oracle table cannot be cached in separate cache groups within the same TimesTen database. However, the table can be cached in separate cache groups within different TimesTen databases.

A TimesTen cache grid provides users with Oracle databases a means to horizontally scale out cache groups across multiple systems with read and write data consistency across the TimesTen databases and predictable latency for database transactions. A cache grid contains one or more grid members that collectively manage application data using the relational data model. A grid member is either a standalone TimesTen database or an active standby pair that consists of at least two replicated TimesTen databases.

Each database of a grid member is called a grid node. A node is a single TimesTen database that is either a standalone database, or the active master database or standby master database of an active standby pair. Therefore, a grid member is composed of one or two nodes.

See “Administering an Active Standby Pair with Cache Groups” in Oracle TimesTen In-Memory Database TimesTen to TimesTen Replication Guide for more information about replicating cache tables.

Grid members can reside on the same system or on different systems. If the grid members reside across different systems, the systems must be connected to each other in a TCP/IP private network. Each system must have the same machine architecture, operating system version, platform and bit version. The TimesTen major release number of all grid members must be the same.

A TimesTen database that is or is part of a grid member can contain local and global cache groups as well as explicitly loaded and dynamic cache groups.

See "Dynamic cache groups" on page 4-36 for more information about dynamic cache groups.

See “Global cache groups” on page 4-37 for more information about global cache groups.

A cache grid can be associated with only one Oracle database. A TimesTen database can be a member of only one cache grid. An Oracle database can be associated with more than one cache grid and each grid can be administered by a different cache administration user. A cache grid has no association with other cache grids.

This section describes the operations that must be performed on the TimesTen database by the cache manager user. The topics include:

- Modify the PROCESSES Oracle system parameter for ten or more grid nodes
- Create a cache grid
- Associate a TimesTen database with a cache grid

Modify the PROCESSES Oracle system parameter for ten or more grid nodes

If you are planning a grid with ten or more nodes, modify the PROCESSES Oracle system parameter. Use this guideline:

\[
\text{PROCESSES} \geq (10 \times \text{GridMembers} + \text{TPLConnections} + \text{OraBackgroundProcesses})
\]
Configuring a cache grid

where

\[ \text{GridMembers} = \text{number of grid members} \]
\[ \text{TPLConnections} = \text{number of transparent load connections} \]
\[ \text{OraBackgroundProcesses} = \text{number of Oracle background processes} \]

The number of transparent load connections is determined by how many sessions will have dynamic cache group operations.

For more information about modifying an Oracle system parameter, see "Changing Parameter Values in a Parameter File" in Oracle Database Reference. For more information about Oracle background processes, see "Background Processes" in Oracle Database Reference.

Create a cache grid

In the examples used throughout the rest of this guide, you will create a cache grid \texttt{ttGrid} that contains three grid members: two standalone TimesTen databases and an active standby pair consisting of three TimesTen databases. This chapter shows how to associate one of the standalone databases with the cache grid. Subsequent chapters show how to create the other standalone database and the active standby pair, and how to associate those members with the grid.

See Example 3-6 for the DSN definition of the first standalone TimesTen database.

You can create a cache grid from any of the standalone databases, or from either the active or standby master database of the active standby pair. A cache grid is created only once from any one of the grid members.

Example 3-10  Creating a cache grid

Create the \texttt{ttGrid} cache grid from the first standalone database by calling the \texttt{ttGridCreate} built-in procedure as the cache manager user:

\begin{verbatim}
Command> call ttGridCreate('ttGrid');
\end{verbatim}

All the databases in these examples, except for the read-only subscriber database of the active standby pair, will be associated with the \texttt{ttGrid} cache grid.

If you manually created the Oracle objects used to store information about TimesTen databases that are associated with a particular cache grid as described in "Manually create Oracle objects used to manage caching of Oracle data" on page 3-10, you do not need to call \texttt{ttGridCreate} because the grid, in effect, was created by running the \texttt{initCacheGridSchema.sql} script.

By default, you must associate a TimesTen database with a cache grid before you can create cache groups in that database. For backward compatibility, you can set the \texttt{CacheGridEnable} DSN attribute to 0 so that you do not have to create a cache grid and associate the TimesTen database with the grid before cache groups can be created within that database. However, regardless of the setting of \texttt{CacheGridEnable}, you must create a cache grid and associate the TimesTen database with the grid before you can create global cache groups within that database. See "Global cache groups" on page 4-37 for more information about global cache groups.

\texttt{CacheGridEnable} is set to 1 by default.
**Associate a TimesTen database with a cache grid**

All standalone databases, and the active and standby master databases of the active standby pair must be associated with a cache grid before you can create cache groups within those databases.

*Example 3–11  Associating a TimesTen database with a cache grid*

Associate the first standalone database to the `ttGrid` cache grid by calling the `ttGridNameSet` built-in procedure as the cache manager user:

```
Command> call ttGridNameSet('ttGrid');
```

**Testing the connectivity between the TimesTen and Oracle databases**

To test the connectivity between the TimesTen and Oracle databases, set the passthrough level to 3 and execute the following query, to be processed on the Oracle database, as the cache manager user:

```
Command> passthrough 3;
Command> SELECT * FROM V$VERSION;
Command> passthrough 0;
```

If connectivity has been successfully established, the query returns the version of the Oracle database. If it does not, check the following for correctness:

- The Oracle net service name set in the `OracleNetServiceName` DSN attribute and the state of the Oracle server
- The settings of the shared library search path environment variable such as `LD_LIBRARY_PATH` or `SHLIB_PATH`
- The setting of the cache administration user name in the TimesTen database

*Example 3–12  Determining the cache administration user name setting*

The cache administration user name setting can be returned programatically by calling the `ttCacheUidGet` built-in procedure as the cache manager user:

```
Command> call ttCacheUidGet;
```

It can also be returned from a command line by running a `ttAdmin -cacheUidGet` utility command as a TimesTen external user with the CACHE_MANAGER privilege:

```
% ttAdmin -cacheUidGet cachealone1
```

**Managing the cache agent**

The cache agent is a TimesTen process that performs cache operations such as loading a cache group and automatic refresh, as well as manages Oracle objects used to enforce the predefined behaviors of particular cache group types.

*Example 3–13  Starting the cache agent*

The cache agent can be manually started programatically by calling the `ttCacheStart` built-in procedure as the cache manager user:

```
Command> call ttCacheStart;
```

It can also be started from a command line by running a `ttAdmin -cacheStart` utility command as a TimesTen external user with the CACHE_MANAGER privilege:
Managing the cache agent

Example 3–14  Stopping the cache agent

The cache agent can be manually stopped programmatically by calling the `ttCacheStop` built-in procedure as the cache manager user:

```sql
Command> call ttCacheStop;
```

It can also be stopped from a command line by running a `ttAdmin -cacheStop` utility command as a TimesTen external user with the CACHE_MANAGER privilege:

```shell
% ttAdmin -cacheStop cachealone1
```

The `ttCacheStop` built-in procedure has an optional parameter and the `ttAdmin -cacheStop` utility command has an option `-stopTimeout` that specifies how long the TimesTen main daemon process waits for the cache agent to stop. If the cache agent does not stop within the specified timeout period, the TimesTen daemon stops the cache agent. The default cache agent stop timeout is 100 seconds. A value of 0 specifies to wait indefinitely.

Do not stop the cache agent immediately after you have dropped or altered an automatic refresh cache group. Instead, wait for at least two minutes to allow the cache agent to clean up Oracle objects such as change log tables and triggers that were created and used to manage the cache group.

---

**Note:** The TimesTen X/Open XA and Java Transaction API (JTA) implementations do not work with Oracle In-Memory Database Cache. The start of any XA or JTA transaction fails if the cache agent is running.

---

Set a cache agent start policy

A cache agent start policy determines how and when the cache agent process starts on a TimesTen database.

The default start policy is `manual` which means the cache agent must be started manually by calling the `ttCacheStart` built-in procedure or running a `ttAdmin -cacheStart` utility command. To manually stop a running cache agent process, call the `ttCacheStop` built-in procedure or run a `ttAdmin -cacheStop` utility command.

The start policy can be set to `always` so that the cache agent starts automatically when the TimesTen main daemon process starts. With the always start policy, the cache agent cannot be stopped when the main daemon is running unless the start policy is changed to either manual or norestart and then a manual stop is issued by calling the `ttCacheStop` built-in procedure or running a `ttAdmin -cacheStop` utility command.

With the `manual` and `always` start policies, the cache agent automatically restarts when the database recovers after a failure such as a database invalidation. If the database was attached to a cache grid when the failure occurred, it is automatically reattached to the grid when the database recovers.

The start policy can be set to `norestart` which means the cache agent must be started manually by calling the `ttCacheStart` built-in procedure or running a `ttAdmin -cacheStart` utility command, and stopped manually by calling the `ttCacheStop` built-in procedure or running a `ttAdmin -cacheStop` utility command.
With the norestart start policy, the cache agent does not automatically restart when the database recovers after a failure such as a database invalidation. You must restart the cache agent manually by calling the `ttCacheStart` built-in procedure or running a `ttAdmin -cacheStart` utility command. If the database was attached to a cache grid when the failure occurred, it is not automatically reattached to the grid when the database recovers. You must call the `ttGridAttach` built-in procedure to reattach the database to the grid.

**Example 3–15 Setting a cache agent start policy**

The cache agent start policy can be set programmatically by calling the `ttCachePolicySet` built-in procedure as the cache manager user:

```plaintext
Command> call ttCachePolicySet('always');
```

It can also be set from a command line by running a `ttAdmin -cachePolicy` utility command as a TimesTen external user with the CACHE_MANAGER privilege:

```plaintext
% ttAdmin -cachePolicy norestart cachealone1
```
This chapter describes the different types of cache groups and how to define them. It includes the following topics:

- Cache groups and cache tables
- Creating a cache group
- Caching Oracle synonyms
- Caching Oracle LOB data
- Implementing aging on a cache group
- Dynamic cache groups
- Global cache groups

### Cache groups and cache tables

A cache group defines the Oracle data to cache in the TimesTen database. When you create a cache group, cache tables are created in the TimesTen database that correspond to the Oracle tables being cached.

A separate table definition must be specified in the cache group definition for each Oracle table that is being cached. The owner, table name and cached column names of a TimesTen cache table must match the owner, table name and column names of the corresponding cached Oracle table. The cache table can contain all or a subset of the columns and rows of the cached Oracle table.

Each Oracle table to be cached in TimesTen must have a primary key or a unique index defined on non-nullable columns. Also, the primary key or unique index that is defined on each TimesTen cache table must match that of the cached Oracle table. For example, if the cached Oracle table has a composite primary key on columns C1, C2 and C3, the TimesTen cache table must also have a composite primary key on columns C1, C2 and C3. Therefore, a cache table must have a primary key or a unique index defined on non-nullable columns just like the cached Oracle table.

You can also create non-unique indexes on a TimesTen cache table. Creating indexes on a cache table can help speed up particular queries issued on the table in the same fashion as on a TimesTen regular table.

Do not create unique indexes on a cache table that do not match any unique index on the cached Oracle table. Otherwise, it can cause unique constraint failures in the cache table that do not occur in the cached Oracle table, and result in these tables in the two databases being no longer synchronized with each other when automatic refresh operations are performed.
A TimesTen database can contain multiple cache groups. A cache group can contain one or more cache tables. An Oracle table cannot be cached in more than one cache group within the same TimesTen database.

**Single-table cache group**

The simplest cache group is one that caches a single Oracle table. In a single-table cache group, there is a root table but no child tables.

Figure 4–1 shows a single-table cache group **target_customers** that caches the **customer** table.

![Figure 4–1 Cache group with a single table](image)

**Multiple-table cache group**

A multiple-table cache group is one that defines a root table and one or more child tables. A cache group can only contain one root table. Each child table must reference the primary key or a unique index of the root table or of another child table in the cache group using a foreign key constraint. Although tables in a multiple-table cache group must be related to each other in the TimesTen database through foreign key constraints, it is not required that the tables be related to each other in the Oracle database. The root table does not reference any table in the cache group with a foreign key constraint.
Figure 4–2 shows a multiple-table cache group `customer_orders` that caches the `customer`, `orders` and `order_item` tables. Each parent table in the `customer_orders` cache group has a primary key that is referenced by a child table through a foreign key constraint. The `customer` table is the root table of the cache group because it does not reference any table in the cache group with a foreign key constraint. The primary key of the root table is considered the primary key of the cache group. The `orders` table is a child table of the `customer` root table. The `order_item` table is a child table of the `orders` child table.

The table hierarchy in a multiple-table cache group can designate child tables to be parents of other child tables. A child table cannot reference more than one parent table. However, a parent table can be referenced by more than one child table.

Figure 4–3 shows an improper cache table hierarchy. Neither the `customer` nor the `product` table references a table in the cache group with a foreign key constraint. This results in the cache group having two root tables which is invalid.

<table>
<thead>
<tr>
<th>ord_num</th>
<th>prod_num</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>44325</td>
<td>SD07</td>
<td>1</td>
</tr>
<tr>
<td>44325</td>
<td>TR3A</td>
<td>5</td>
</tr>
<tr>
<td>65432</td>
<td>FT094</td>
<td>1</td>
</tr>
<tr>
<td>76543</td>
<td>SD07</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ord_num</th>
<th>when_placed</th>
<th>when_shipped</th>
</tr>
</thead>
<tbody>
<tr>
<td>44325</td>
<td>10/7/07</td>
<td>10/7/07</td>
</tr>
<tr>
<td>65432</td>
<td>8/24/08</td>
<td>8/27/08</td>
</tr>
<tr>
<td>76543</td>
<td>4/2/09</td>
<td>4/8/09</td>
</tr>
</tbody>
</table>
Figure 4–3  Problem: Cache group contains two root tables

TimesTen

Cache group customer_orders

<table>
<thead>
<tr>
<th>cust_num</th>
<th>region</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>122</td>
<td>West</td>
<td>Jim Johnston</td>
<td>231 Main, Needles CA 92363</td>
</tr>
<tr>
<td>342</td>
<td>West</td>
<td>Jane Stone</td>
<td>4223 Cowper, Palo Alto 94302</td>
</tr>
<tr>
<td>663</td>
<td>Midwest</td>
<td>Mary J. Warren</td>
<td>673 State, Madison WI 53787</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ord_num</th>
<th>cust_num</th>
<th>when_placed</th>
<th>when_shipped</th>
</tr>
</thead>
<tbody>
<tr>
<td>44325</td>
<td>122</td>
<td>10/7/07</td>
<td>10/7/07</td>
</tr>
<tr>
<td>65432</td>
<td>122</td>
<td>8/24/08</td>
<td>8/27/08</td>
</tr>
<tr>
<td>76543</td>
<td>663</td>
<td>4/2/09</td>
<td>4/8/09</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>prod_num</th>
<th>name</th>
<th>price</th>
<th>ship_weight</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD07</td>
<td>TR3A</td>
<td>$4.50</td>
<td>2 lbs</td>
<td>1” brad</td>
</tr>
<tr>
<td>FT094</td>
<td>FT133</td>
<td>$1.94</td>
<td>5.4 lbs</td>
<td>.3” washer</td>
</tr>
<tr>
<td>FT133</td>
<td>SD07</td>
<td>$2.76</td>
<td>7.5 lbs</td>
<td>.4” washer</td>
</tr>
<tr>
<td>FT094</td>
<td>SD07</td>
<td>$1.50</td>
<td>2.5 lbs</td>
<td>.5” washer</td>
</tr>
</tbody>
</table>

Order Item

<table>
<thead>
<tr>
<th>ord_num</th>
<th>prod_num</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>44325</td>
<td>SD07</td>
<td>1</td>
</tr>
<tr>
<td>44325</td>
<td>TR3A</td>
<td>5</td>
</tr>
<tr>
<td>65432</td>
<td>FT094</td>
<td>1</td>
</tr>
<tr>
<td>76543</td>
<td>SD07</td>
<td>2</td>
</tr>
</tbody>
</table>

To resolve this problem and cache all the tables, create a cache group which contains the customer, orders, and order_item tables, and a second cache group which contains the product and the inventory tables as shown in Figure 4–4.
Creating a cache group

You create cache groups by using a CREATE CACHE GROUP SQL statement or by using Oracle SQL Developer, a graphical tool. For more information about SQL Developer, see Oracle SQL Developer TimesTen In-Memory Database Support User’s Guide.

Cache groups are identified as either system managed or user managed. System managed cache groups enforce specific behaviors, while the behavior of a user managed cache group can be customized. System managed cache group types include:

- Read-only cache group
- Asynchronous writethrough (AWT) cache group
- Synchronous writethrough (SWT) cache group

See "User managed cache group" on page 4-15 for information about user managed cache groups.

The following topics also apply to creating a cache group:

- AUTOREFRESH cache group attribute
- Using a WHERE clause
- ON DELETE CASCADE cache table attribute
- **UNIQUE HASH ON cache table attribute**

Cache groups must be created by and are owned by the cache manager user.

**Read-only cache group**

A read-only cache group enforces a caching behavior where the TimesTen cache tables cannot be updated directly, and committed updates on the cached Oracle tables are automatically refreshed to the cache tables as shown in Figure 4–5.

*Depending on the PassThrough attribute setting*

If the TimesTen database is unavailable for whatever reason, you can still update the Oracle tables that are cached in a read-only cache group. When the TimesTen database returns to operation, updates that were committed on the cached Oracle tables while the TimesTen database was unavailable are automatically refreshed to the TimesTen cache tables.

The following are the definitions of the Oracle tables that will be cached in the read-only cache groups that are defined in Example 4–1, Example 4–10, Example 4–11, Example 4–19 and Example 4–20. The Oracle tables are owned by the schema user `oratt`. The `oratt` user must be granted the CREATE SESSION and RESOURCE privileges before it can create tables.

```sql
CREATE TABLE customer
    (cust_num NUMBER(6) NOT NULL PRIMARY KEY,
     region VARCHAR2(10),
     name VARCHAR2(50),
    ...
```
Creating a cache group

Defining Cache Groups

4-7

CREATE TABLE orders
(ord_num NUMBER(10) NOT NULL PRIMARY KEY,
cust_num NUMBER(6) NOT NULL,
when_placed DATE NOT NULL,
when_shipped DATE NOT NULL)

The Oracle user with the same name as the TimesTen cache manager user must be
granted the SELECT privilege on the oratt.customer and oratt.orders tables in
order for the cache manager user to create a read-only cache group that caches these
tables, and for automatic refresh operations to occur from the cached Oracle tables to
the TimesTen cache tables.

Use the CREATE READONLY CACHE GROUP statement to create a read-only cache
group.

Example 4–1  Creating a read-only cache group

The following statement creates a read-only cache group customer_orders that
caches the tables oratt.customer (root table) and oratt.orders (child table):

CREATE READONLY CACHE GROUP customer_orders
FROM oratt.customer
{cust_num NUMBER(6) NOT NULL,
region VARCHAR2(10),
name VARCHAR2(50),
adress VARCHAR2(100),
PRIMARY KEY(cust_num)},
oratt.orders
{ord_num NUMBER(10) NOT NULL,
cust_num NUMBER(6) NOT NULL,
when_placed DATE NOT NULL,
when_shipped DATE NOT NULL,
PRIMARY KEY(ord_num),
FOREIGN KEY(cust_num) REFERENCES oratt.customer(cust_num)}

The cache tables in a read-only cache group cannot be updated directly. However, you
can set the passthrough level to 2 to allow committed update operations issued on a
TimesTen cache table to be passed through and processed on the cached Oracle table,
and then have the updates be automatically refreshed into the cache table.

See "Setting a passthrough level" on page 5-15 for information about how to set a
passthrough level.

The effects of a passed through statement on cache tables in a read-only cache group
do not occur in the transaction in which the update operation was issued. Instead, they
are seen after the passed through update operation has been committed on the Oracle
database and the next automatic refresh of the cache group has occurred. The Oracle
user with the same name as the TimesTen cache manager user must be granted the
INSERT, UPDATE and DELETE privileges on the Oracle tables that are cached in the
read-only cache group in order for the passed through update operations to be
processed on the cached Oracle tables.

If you manually created the Oracle objects used to enforce the predefined behaviors of
an automatic refresh cache group as described in "Manually create Oracle objects used
to manage caching of Oracle data" on page 3-10, you need to set the automatic refresh
state to OFF when creating the cache group.
Then you need to run the `ttIsq1` utility's `cachesqlget` command to generate a SQL*Plus script used to create a log table and a trigger in the Oracle database for each Oracle table that is cached in the read-only cache group.

See "Manually creating Oracle objects for automatic refresh cache groups" on page 4-24 for information about how to create these objects.

**Restrictions with read-only cache groups**

The following restrictions apply when using a read-only cache group:

- The cache tables cannot be updated directly.

- Only the ON DELETE CASCADE and UNIQUE HASH ON cache table attributes can be used in the cache table definitions.
  
  See "ON DELETE CASCADE cache table attribute" on page 4-27 for more information about the ON DELETE CASCADE cache table attribute.

  See "UNIQUE HASH ON cache table attribute" on page 4-28 for more information about the UNIQUE HASH ON cache table attribute.

- A FLUSH CACHE GROUP statement cannot be issued on the cache group.
  
  See "Flushing a user managed cache group" on page 5-14 for more information about the FLUSH CACHE GROUP statement.

- A TRUNCATE TABLE statement issued on a cached Oracle table is not automatically refreshed to the TimesTen cache table.

- A LOAD CACHE GROUP statement can only be issued on the cache group if the cache tables are empty, unless the cache group is dynamic.
  
  See "Loading and refreshing a cache group" on page 5-2 for more information about the LOAD CACHE GROUP statement.

  See "Dynamic cache groups" on page 4-36 for more information about dynamic cache groups.

- The automatic refresh state must be PAUSED before you can issue a LOAD CACHE GROUP statement on the cache group, unless the cache group is dynamic, in which case the automatic refresh state must be PAUSED or ON. The LOAD CACHE GROUP statement cannot contain a WHERE clause, unless the cache group is dynamic, in which case the WHERE clause must be followed by a COMMIT EVERY n ROWS clause.
  
  See "AUTOREFRESH cache group attribute" on page 4-21 for more information about automatic refresh states.

See "Using a WHERE clause" on page 4-24 for more information about WHERE clauses in cache group definitions and operations.

- The automatic refresh state must be PAUSED before you can issue a REFRESH CACHE GROUP statement on the cache group. The REFRESH CACHE GROUP statement cannot contain a WHERE clause.
  
  See "Loading and refreshing a cache group" on page 5-2 for more information about the REFRESH CACHE GROUP statement.

- All tables and columns referenced in WHERE clauses when creating, loading or unloading the cache group must be fully qualified. For example: `user_name.table_name` and `user_name.table_name.column_name`.

- Least recently used (LRU) aging cannot be specified on the cache group, unless the cache group is dynamic where LRU aging is defined by default.
See "LRU aging" on page 4-31 for more information about LRU aging.

**Asynchronous writethrough (AWT) cache group**

An asynchronous writethrough (AWT) cache group enforces a caching behavior where committed updates on the TimesTen cache tables are automatically and asynchronously propagated to the cached Oracle tables as shown in Figure 4–6.

*Figure 4–6 Asynchronous writethrough cache group*

The transaction commit on the TimesTen database occurs asynchronously from the commit on the Oracle database. This allows an application to continue issuing transactions on the TimesTen database without having to wait for the Oracle transaction to complete. However, your application cannot ensure when the transactions are completed on the Oracle database.

You can update cache tables in an AWT cache group even if the Oracle database is unavailable. When the Oracle database returns to operation, updates that were committed on the cache tables while the Oracle database was unavailable are automatically propagated to the cached Oracle tables.

The following is the definition of the Oracle table that will be cached in the AWT cache groups that are defined in Example 4–2, Example 4–12 and Example 4–14. The Oracle table is owned by the schema user `oratt`. The `oratt` user must be granted the CREATE SESSION and RESOURCE privileges before it can create tables.

```sql
CREATE TABLE customer
(cust_num NUMBER(6) NOT NULL PRIMARY KEY,
 region VARCHAR2(10),
```

---

**Defining Cache Groups** 4-9
name VARCHAR2(50),
address VARCHAR2(100))

The Oracle user with the same name as the TimesTen cache manager user must be
granted the SELECT privilege on the oratt.customer table in order for the cache
manager user to create an AWT cache group that caches this table. The Oracle cache
administration user must be granted the INSERT, UPDATE and DELETE privileges on
the oratt.customer table for asynchronous writethrough operations to occur from
the TimesTen cache table to the cached Oracle table.

Use the CREATE ASYNCHRONOUS WRITETHROUGH CACHE GROUP statement
to create an AWT cache group.

**Example 4–2 Creating an AWT cache group**

The following statement creates an asynchronous writethrough cache group
new_customers that caches the oratt.customer table:

```
CREATE ASYNCHRONOUS WRITETHROUGH CACHE GROUP new_customers
FROM oratt.customer
(cust_num NUMBER(6) NOT NULL,
  region   VARCHAR2(10),
  name     VARCHAR2(50),
  address  VARCHAR2(100),
  PRIMARY KEY(cust_num))
```

**Managing the replication agent**

Performing asynchronous writethrough operations requires that the replication agent
be running on the TimesTen database that contains AWT cache groups. Executing a
CREATE ASYNCHRONOUS WRITETHROUGH CACHE GROUP statement creates a
replication scheme that enables committed updates on the TimesTen cache tables to be
asynchronously propagated to the cached Oracle tables.

After you have created AWT cache groups, start the replication agent on the TimesTen
database.

**Example 4–3 Starting the replication agent**

The replication agent can be manually started programmatically by calling the
ttRepStart built-in procedure as the cache manager user:

```
% ttIsql "DSN=cachaloner1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> call ttRepStart;
```

It can also be started from a command line by running a ttAdmin -repStart utility
command as a TimesTen external user with the CACHE_MANAGER privilege:

```
% ttAdmin -repStart cachaloner1
```

The replication agent does not start unless there is at least one AWT cache group or
replication scheme in the TimesTen database.

If the replication agent is running, it must be stopped before you can issue another
CREATE ASYNCHRONOUS WRITETHROUGH CACHE GROUP statement or a
DROP CACHE GROUP statement on an AWT cache group.

**Example 4–4 Stopping the replication agent**

The replication agent can be manually stopped programmatically by calling the
ttRepStop built-in procedure as the cache manager user:
Creating a cache group

Defining Cache Groups

4-11

Command> call ttRepStop;

It can also be stopped from a command line by running a `ttAdmin -repStop` utility command as a TimesTen external user with the CACHE_MANAGER privilege:

```
% ttAdmin -repStop cachealone1
```

You can set a replication agent start policy to determine how and when the replication agent process starts on a TimesTen database.

The default start policy is `manual` which means the replication agent must be started manually by calling the `ttRepStart` built-in procedure or running a `ttAdmin -repStart` utility command. To manually stop a running replication agent process, call the `ttRepStop` built-in procedure or run a `ttAdmin -repStop` utility command.

The start policy can be set to `always` so that the replication agent starts automatically when the TimesTen main daemon process starts. With the `always` start policy, the replication agent cannot be stopped when the main daemon is running unless the start policy is changed to either `manual` or `norestart` and then a manual stop is issued by calling the `ttRepStop` built-in procedure or running a `ttAdmin -repStop` utility command.

With the `manual` and `always` start policies, the replication agent automatically restarts after a failure such as a database invalidation.

The start policy can be set to `norestart` which means the replication agent must be started manually by calling the `ttRepStart` built-in procedure or running a `ttAdmin -repStart` utility command, and stopped manually by calling the `ttRepStop` built-in procedure or running a `ttAdmin -repStop` utility command.

With the `norestart` start policy, the replication agent does not automatically restart after a failure such as a database invalidation. You must restart the replication agent manually by calling the `ttRepStart` built-in procedure or running a `ttAdmin -repStart` utility command.

**Example 4–5 Setting a replication agent start policy**

As the instance administrator, grant the ADMIN privilege to the cache manager user:

```
% ttIsql cachealone1
Command> GRANT ADMIN TO cacheuser;
Command> exit
```

The replication agent start policy can be set programmatically by calling the `ttRepPolicySet` built-in procedure as the cache manager user:

```
% ttIsql 'DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle'
Command> call ttRepPolicySet('manual');
Command> exit
```

It can also be set from a command line by running a `ttAdmin -repPolicy` utility command as a TimesTen external user with the ADMIN privilege:

```
% ttAdmin -repPolicy always cachealone1
```

**What an AWT cache group does and does not guarantee**

An AWT cache group *can* guarantee:

- No transactions are lost because of communication failures between the TimesTen and Oracle databases.
■ If the replication agent is not running or loses its connection to the Oracle database, automatic propagation of committed updates on the TimesTen cache tables to the cached Oracle tables resumes after the agent is restarted or is able to reconnect to the Oracle database.

■ Transactions are committed in the Oracle database in the same order they were committed in the TimesTen database.

An AWT cache group cannot guarantee:

■ All transactions committed successfully in the TimesTen database are successfully propagated to and committed in the Oracle database. Execution errors on Oracle cause the transaction in the Oracle database to be rolled back. For example, an update on Oracle may fail because of a unique constraint violation. Transactions that contain execution errors are not retried.

Execution errors are reported to the TimesTenDatabaseFileName.awterrs file that resides in the same directory as the TimesTen database’s checkpoint files. See “Permanent Oracle errors reported by TimesTen” in Oracle TimesTen In-Memory Database Troubleshooting Procedures Guide for information about the contents of this file.

■ The absolute order of Oracle updates is preserved because TimesTen does not resolve update conflicts. The following are some examples:
  
  – An update is committed on a cache table in an AWT cache group. The same update is committed on the cached Oracle table using a passthrough operation. The cache table update, which is automatically and asynchronously propagated to Oracle, may overwrite the passed through update that was processed directly on the cached Oracle table depending on when the propagated update and the passed through update is processed on Oracle.

  – In two separate TimesTen databases (DB1 and DB2), different AWT cache groups cache the same Oracle table. An update is committed on the cache table in DB1. An update is then committed on the cache table in DB2. The two cache tables reside in different TimesTen databases and cache the same Oracle table. Because the writethrough operations are asynchronous, the update from DB2 may get propagated to the Oracle database before the update from DB1, resulting in the update from DB1 overwriting the update from DB2.

Using a dynamic AWT global cache group resolves this write inconsistency. See “Global cache groups” on page 4-37 for more information about global cache groups.

**Restrictions with AWT cache groups**

The following restrictions apply when using an AWT cache group:

■ Only the ON DELETE CASCADE and UNIQUE HASH ON cache table attributes can be used in the cache table definitions.

See “ON DELETE CASCADE cache table attribute” on page 4-27 for more information about the ON DELETE CASCADE cache table attribute.

See “UNIQUE HASH ON cache table attribute” on page 4-28 for more information about the UNIQUE HASH ON cache table attribute.

■ A FLUSH CACHE GROUP statement cannot be issued on the cache group.

See “Flushing a user managed cache group” on page 5-14 for more information about the FLUSH CACHE GROUP statement.

■ The cache table definitions cannot contain a WHERE clause.
Creating a cache group

See "Using a WHERE clause" on page 4-24 for more information about WHERE clauses in cache group definitions and operations.

- A TRUNCATE TABLE statement cannot be issued on the cache tables.
- The replication agent must be stopped before creating or dropping an AWT cache group.
  See "Managing the replication agent" on page 4-10 for information about how to stop and start the replication agent.
- Committed updates on the TimesTen cache tables are not propagated to the cached Oracle tables unless the replication agent is running.
- To create an AWT cache group, the length of the TimesTen database's absolute path name cannot exceed 248 characters.
- TimesTen does not detect or resolve update conflicts that occur on Oracle. Committed updates made directly on a cached Oracle table may be overwritten by a committed update made on the TimesTen cache table when the cache table update is propagated to Oracle.
- TimesTen performs deferred checking when determining whether a single SQL statement causes a constraint violation with a unique index.
  For example, suppose there is a unique index on a cached Oracle table's NUMBER column, and a unique index on the same NUMBER column on the TimesTen cache table. There are five rows in the cached Oracle table and the same five rows in the cache table. The values in the NUMBER column range from 1 to 5.
  An UPDATE statement is issued on the cache table to increment the value in the NUMBER column by 1 for all rows. The operation succeeds on the cache table but fails when it is propagated to the cached Oracle table.
  That's because TimesTen performs the unique index constraint check at the end of the statement's execution after all the rows have been updated. Oracle, however, performs the constraint check each time after a row has been updated.
  Therefore, when the row in the cache table with value 1 in the NUMBER column is changed to 2 and the update is propagated to Oracle, it causes a unique constraint violation with the row that has the value 2 in the NUMBER column of the cached Oracle table.

**Synchronous writethrough (SWT) cache group**

A synchronous writethrough (SWT) cache group enforces a caching behavior where committed updates on the TimesTen cache tables are automatically and synchronously propagated to the cached Oracle tables as shown in Figure 4–7.
The transaction commit on the TimesTen database occurs synchronously with the commit on the Oracle database. When an application commits a transaction in the TimesTen database, the transaction is processed in the Oracle database before it is processed in TimesTen. The application is blocked until the transaction has completed in both the Oracle and TimesTen databases.

If the transaction fails to commit in Oracle, the application must roll back the transaction in TimesTen. If the Oracle transaction commits successfully but the TimesTen transaction fails to commit, the cache tables in the SWT cache group are no longer synchronized with the cached Oracle tables. To manually resynchronize the cache tables with the cached Oracle tables, call the `ttCachePropagateFlagSet` built-in procedure to disable update propagation, and then reissue the transaction in the TimesTen database after correcting the problem that caused the transaction commit to fail in TimesTen. You can also resynchronize the cache tables with the cached Oracle tables by reloading the accompanying cache groups.

The following is the definition of the Oracle table that will be cached in the SWT cache group that is defined in Example 4–6. The Oracle table is owned by the schema user `oratt`. The `oratt` user must be granted the CREATE SESSION and RESOURCE privileges before it can create tables.

```sql
CREATE TABLE product
(prod_num VARCHAR2(6) NOT NULL PRIMARY KEY,
 name VARCHAR2(30),
 price NUMBER(8,2),
 ship_weight NUMBER(4,1))
```

The Oracle user with the same name as the TimesTen cache manager user must be granted the SELECT privilege on the `oratt.product` table in order for the cache manager user to create an SWT cache group that caches this table. This Oracle user
must also be granted the INSERT, UPDATE and DELETE privileges on the oratt.product table for synchronous writethrough operations to occur from the TimesTen cache table to the cached Oracle table.

Use the CREATE SYNCHRONOUS WRITETHROUGH CACHE GROUP statement to create an SWT cache group.

**Example 4–6  Creating a SWT cache group**

The following statement creates a synchronous writethrough cache group top_products that caches the oratt.product table:

```
CREATE SYNCHRONOUS WRITETHROUGH CACHE GROUP top_products
FROM oratt.product
(prod_num    VARCHAR2(6) NOT NULL,
 name        VARCHAR2(30),
 price       NUMBER(8,2),
 ship_weight NUMBER(4,1),
 PRIMARY KEY(prod_num))
```

**Restrictions with SWT cache groups**

The following restrictions apply when using an SWT cache group:

- Only the ON DELETE CASCADE and UNIQUE HASH ON cache table attributes can be used in the cache table definitions.
  
  See "ON DELETE CASCADE cache table attribute" on page 4-27 for more information about the ON DELETE CASCADE cache table attribute.
  
  See "UNIQUE HASH ON cache table attribute" on page 4-28 for more information about the UNIQUE HASH ON cache table attribute.

- A FLUSH CACHE GROUP statement cannot be issued on the cache group.
  
  See "Flushing a user managed cache group" on page 5-14 for more information about the FLUSH CACHE GROUP statement.

- The cache table definitions cannot contain a WHERE clause.
  
  See "Using a WHERE clause" on page 4-24 for more information about WHERE clauses in cache group definitions and operations.

- A TRUNCATE TABLE statement cannot be issued on the cache tables.

**User managed cache group**

If the system managed cache groups (read-only, AWT, SWT) do not satisfy your application's requirements, you can create a user managed cache group that defines customized caching behavior. For example:

- You can define a user managed cache group to automatically refresh and propagate committed updates between the Oracle and TimesTen databases by using the AUTOREFRESH cache group attribute and the PROPAGATE cache table attribute. Using both attributes enables bidirectional transmit, so that committed updates on the TimesTen cache tables or the cached Oracle tables are propagated or refreshed to each other.

- You can use the LOAD CACHE GROUP, REFRESH CACHE GROUP, and FLUSH CACHE GROUP statements to manually control the transmit of committed updates between the Oracle and TimesTen databases.
You can specify the READONLY or the PROPAGATE cache table attribute on individual cache tables in a user managed cache group to define read-only or synchronous writethrough behavior at the table level.

The following are the definitions of the Oracle tables that will be cached in the user managed cache groups that are defined in Example 4–7 and Example 4–8. The Oracle tables are owned by the schema user oratt. The oratt user must be granted the CREATE SESSION and RESOURCE privileges before it can create tables.

```
CREATE TABLE active_customer
(custid NUMBER(6) NOT NULL PRIMARY KEY,
 name VARCHAR2(50),
 addr VARCHAR2(100),
 zip VARCHAR2(12),
 region VARCHAR2(12) DEFAULT 'Unknown')

CREATE TABLE ordertab
(orderid NUMBER(10) NOT NULL PRIMARY KEY,
 custid NUMBER(6) NOT NULL)

CREATE TABLE cust_interests
(custid NUMBER(6) NOT NULL,
 interest VARCHAR2(10) NOT NULL,
 PRIMARY KEY (custid, interest))

CREATE TABLE orderdetails
(orderid NUMBER(10) NOT NULL,
 itemid NUMBER(8) NOT NULL,
 quantity NUMBER(4) NOT NULL,
 PRIMARY KEY (orderid, itemid))
```

Use the CREATE USERMANAGED CACHE GROUP statement to create a user managed cache group.

**Example 4–7 Creating a single-table user managed cache group**

The following statement creates a user managed cache group update_anywhere_customers that caches the oratt.active_customer table as shown in Figure 4–8:

```
CREATE USERMANAGED CACHE GROUP update_anywhere_customers
AUTOREFRESH MODE INCREMENTAL INTERVAL 30 SECONDS
FROM oratt.active_customer
(custid NUMBER(6) NOT NULL,
 name VARCHAR2(50),
 addr VARCHAR2(100),
 zip VARCHAR2(12),
 PRIMARY KEY (custid),
 PROPAGATE)
```
Creating a cache group

Defining Cache Groups

4-17

Figure 4–8  Single-table user managed cache group

All columns except region are cached. Only customers whose customer ID is greater than or equal to 1001 are cached. Committed updates on the `oratt.active_customer` cache table or the `oratt.active_customer` cached Oracle table are transmitted to the corresponding table.

The Oracle user with the same name as the TimesTen cache manager user must be granted the SELECT privilege on the `oratt.active_customer` table in order for the cache manager user to create a user managed cache group that caches this table, and for automatic refresh operations to occur from the cached Oracle table to the TimesTen cache table. This Oracle user must also be granted the INSERT, UPDATE and DELETE privileges on the `oratt.active_customer` table for synchronous writethrough operations to occur from the TimesTen cache table to the cached Oracle table.

In this example, the AUTOREFRESH cache group attribute specifies that committed updates on the `oratt.active_customer` cache table are automatically refreshed to the TimesTen `oratt.active_customer` cached Oracle table every 30 seconds. The PROPAGATE cache table attribute specifies that committed updates on the cache table are automatically and synchronously propagated to the cached Oracle table.

See “AUTOREFRESH cache group attribute” on page 4-21 for more information about defining an automatic refresh mode, interval and state.

If you manually created the Oracle objects used to enforce the predefined behaviors of a user managed cache group that uses the AUTOREFRESH MODE INCREMENTAL cache group attribute as described in "Manually create Oracle objects used to manage caching of Oracle data" on page 3-10, you need to set the automatic refresh state to OFF when creating the cache group.
Then you need to run the `ttIsql` utility's `cachesqlget` command to generate a SQL*Plus script used to create a log table and a trigger in the Oracle database for each Oracle table that is cached in the user managed cache group.

See "Manually creating Oracle objects for automatic refresh cache groups" on page 4-24 for more information.

**Example 4–8 Creating a multiple-table user managed cache group**

The following statement creates a user managed cache group `western_customers` that caches the `oratt.active_customer`, `oratt.ordertab`, `oratt.cust_interests`, and `oratt.orderdetails` tables as shown in Figure 4–9:

```sql
CREATE USERMANAGED CACHE GROUP western_customers
FROM oratt.active_customer
  (custid NUMBER(6) NOT NULL,
    name VARCHAR2(50),
    addr VARCHAR2(100),
    zip VARCHAR2(12),
    region VARCHAR2(12),
    PRIMARY KEY(custid),
    PROPAGATE)
  WHERE (oratt.active_customer.region = 'West'),
FROM oratt.ordertab
  (orderid NUMBER(10) NOT NULL,
    custid NUMBER(6) NOT NULL,
    PRIMARY KEY(orderid),
    FOREIGN KEY(custid) REFERENCES oratt.active_customer(custid),
    PROPAGATE),
FROM oratt.cust_interests
  (custid NUMBER(6) NOT NULL,
    interest VARCHAR2(10) NOT NULL,
    PRIMARY KEY(custid, interest),
    FOREIGN KEY(custid) REFERENCES oratt.active_customer(custid),
    READONLY),
FROM oratt.orderdetails
  (orderid NUMBER(10) NOT NULL,
    itemid NUMBER(8) NOT NULL,
    quantity NUMBER(4) NOT NULL,
    PRIMARY KEY(orderid, itemid),
    FOREIGN KEY(orderid) REFERENCES oratt.ordertab(orderid))
WHERE (oratt.orderdetails.quantity >= 5)
```
Creating a cache group

Defining Cache Groups

Figure 4–9  Multiple-table user managed cache group

TimesTen

User managed cache group western_customers

active_customer (root table)

<table>
<thead>
<tr>
<th>custid</th>
<th>name</th>
<th>addr</th>
<th>zip</th>
<th>region</th>
</tr>
</thead>
</table>

ordertab

<table>
<thead>
<tr>
<th>orderid</th>
<th>custid</th>
</tr>
</thead>
</table>

cust_interests

<table>
<thead>
<tr>
<th>custid</th>
<th>interest</th>
</tr>
</thead>
</table>

orderdetails

<table>
<thead>
<tr>
<th>orderid</th>
<th>itemid</th>
<th>quantity</th>
</tr>
</thead>
</table>

Oracle Database

active_customer

Data for all customers

ordertab

order_details

cust_interests

Only customers in the West region who ordered at least 5 of the same item are cached.

The Oracle user with the same name as the TimesTen cache manager user must be granted the SELECT privilege on the oratt.active_customer, oratt.ordertab, oratt.cust_interests, and oratt.orderdetails tables in order for the cache manager user to create a user managed cache group that caches all of these tables. This Oracle user must also be granted the INSERT, UPDATE and DELETE privileges on the oratt.active_customer and oratt.ordertab tables for synchronous writethrough operations to occur from these TimesTen cache tables to the cached Oracle tables.

Each cache table in the western_customers cache group contains a primary key. Each child table references a parent table with a foreign key constraint. The oratt.active_customer root table and the oratt.orderdetails child table each contain a WHERE clause to restrict the rows to be cached. The oratt.active_customer root table and the oratt.ordertab child table both use the PROPAGATE cache table attribute so that committed updates on these cache tables are automatically propagated to the cached Oracle tables. The oratt.cust_interests child table uses the READONLY cache table attribute so that it cannot be updated directly.
Creating a cache group

PROPAGATE cache table attribute

The PROPAGATE cache table attribute can be specified only for cache tables in a user managed cache group. PROPAGATE specifies that committed updates on the TimesTen cache table are automatically and synchronously propagated to the cached Oracle table such that:

1. The commit is first attempted in the Oracle database. If the commit fails in Oracle, the commit is not attempted in the TimesTen database and the application must roll back the TimesTen transaction. As a result, the Oracle database never misses updates committed in TimesTen.

2. If the commit succeeds in the Oracle database, it is then attempted in the TimesTen database. If the commit fails in TimesTen, an error message is returned from TimesTen indicating the cause of the failure. You then need to manually resynchronize the cache tables with the Oracle tables.

See "Synchronous writethrough (SWT) cache group" on page 4-13 for information on how to resynchronize the cache tables with the Oracle tables.

By default, a cache table in a user managed cache group is created with the NOT PROPAGATE cache table attribute such that committed updates on the cache table are not propagated to the cached Oracle table.

When a cache table uses the PROPAGATE cache table attribute, you may occasionally need to commit updates on the cache table that you do not want propagated to the cached Oracle table. Use the \ttCachePropagateFlagSet built-in procedure to disable automatic propagation so that committed updates on a cache table is not propagated to the cached Oracle table.

The following restrictions apply when using the PROPAGATE cache table attribute:

- If the cache group uses the AUTOREFRESH cache group attribute, the PROPAGATE cache table attribute must be specified on all or none of its cache tables. See "AUTOREFRESH cache group attribute" on page 4-21 for more information about using the AUTOREFRESH cache group attribute.

- If the cache group uses the AUTOREFRESH cache group attribute, the NOT PROPAGATE cache table attribute cannot be explicitly specified on any of its cache tables.

- You cannot use both the PROPAGATE and READONLY cache table attributes on the same cache table. See "READONLY cache table attribute" on page 4-21 for more information about using the READONLY cache table attribute.

- A FLUSH CACHE GROUP statement cannot be issued on the cache group unless one or more of its cache tables use neither the PROPAGATE nor the READONLY cache table attribute. See "Flushing a user managed cache group" on page 5-14 for more information about the FLUSH CACHE GROUP statement.

- After the PROPAGATE cache table attribute has been specified on a cache table, you cannot change this attribute unless you drop the cache group and re-create it.

- TimesTen does not perform a conflict check to prevent a propagate operation from overwriting data that was updated directly on a cached Oracle table. Therefore, updates should only be performed directly on the TimesTen cache tables or the cached Oracle tables, but not both.
In Example 4–7, the oratt.active_customer cache table uses the PROPAGATE cache table attribute.

**READONLY cache table attribute**

The READONLY cache table attribute can be specified only for cache tables in a user managed cache group. READONLY specifies that the cache table cannot be updated directly. By default, a cache table in a user managed cache group is updatable.

Unlike a read-only cache group where all of its cache tables are read-only, in a user managed cache group individual cache tables can be specified as read-only using the READONLY cache table attribute.

The following restrictions apply when using the READONLY cache table attribute:

- If the cache group uses the AUTOREFRESH cache group attribute, the READONLY cache table attribute must be specified on all or none of its cache tables.
  
  See "AUTOREFRESH cache group attribute" on page 4-21 for more information about using the AUTOREFRESH cache group attribute.

- You cannot use both the READONLY and PROPAGATE cache table attributes on the same cache table.

  See "PROPAGATE cache table attribute" on page 4-20 for more information about using the PROPAGATE cache table attribute.

- A FLUSH CACHE GROUP statement cannot be issued on the cache group unless one or more of its cache tables use neither the READONLY nor the PROPAGATE cache table attribute.

  See "Flushing a user managed cache group" on page 5-14 for more information about the FLUSH CACHE GROUP statement.

- After the READONLY cache table attribute has been specified on a cache table, you cannot change this attribute unless you drop the cache group and re-create it.

In Example 4–8, the oratt.cust_interests cache table uses the READONLY cache table attribute.

**AUTOREFRESH cache group attribute**

The AUTOREFRESH cache group attribute can be specified when creating a read-only cache group or a user managed cache group using a CREATE CACHE GROUP statement. AUTOREFRESH specifies that committed updates on cached Oracle tables are automatically refreshed to the TimesTen cache tables. Automatic refresh is defined by default on read-only cache groups.

The following are the default settings of the automatic refresh attributes:

- The automatic refresh mode is incremental.
- The automatic refresh interval is 5 minutes.
- The automatic refresh state is PAUSED.

TimesTen supports two automatic refresh modes:

- INCREMENTAL: Committed updates on cached Oracle tables are automatically refreshed to the TimesTen cache tables based on the cache group’s automatic refresh interval. Incremental automatic refresh mode uses Oracle objects to track committed updates on cached Oracle tables. See "Oracle objects used to manage a caching environment" on page 7-10 for information on these objects.
FULL: All cache tables are automatically refreshed, based on the cache group’s automatic refresh interval, by unloading all their rows and then reloading from the cached Oracle tables.

Incremental automatic refresh mode incurs some overhead to refresh the cache group for each committed update on the cached Oracle tables. There is no overhead when using full automatic refresh mode.

When using incremental automatic refresh mode, committed updates on cached Oracle tables are tracked in change log tables in the Oracle database. Under certain circumstances, it is possible for some of the change log records to be deleted from the change log table before they are automatically refreshed to the TimesTen cache tables. If this occurs, TimesTen initiates a full automatic refresh on the cache group. See "Monitoring the cache administration user’s tablespace" on page 7-17 for information on how to configure an action to take when the tablespace that the change log tables reside in becomes full.

The automatic refresh interval determines how often automatic refresh operations occur in minutes, seconds or milliseconds. Cache groups with the same automatic refresh interval are refreshed within the same transaction. You can use the `ttCacheAutorefresh` built-in procedure to initiate an immediate automatic refresh operation. For more information, see

The automatic refresh state can be set to ON, PAUSED or OFF. Automatic refresh operations are scheduled by TimesTen when the cache group’s automatic refresh state is ON.

When the cache group’s automatic refresh state is OFF, committed updates on the cached Oracle tables are not tracked.

When the cache group’s automatic refresh state is PAUSED, committed updates on the cached Oracle tables are tracked in the Oracle database, but are not automatically refreshed to the TimesTen cache tables until the state is changed to ON.

The following restrictions apply when using the AUTOREFRESH cache group attribute:

- A FLUSH CACHE GROUP statement cannot be issued on the cache group.

  See "Flushing a user managed cache group" on page 5-14 for more information about the FLUSH CACHE GROUP statement.

- A TRUNCATE TABLE statement issued on a cached Oracle table is not automatically refreshed to the TimesTen cache table. Before issuing a TRUNCATE TABLE statement on a cached Oracle table, use an ALTER CACHE GROUP statement to change the automatic refresh state of the cache group that contains the cache table to PAUSED.

  See "Altering a cache group" on page 4-23 for more information about the ALTER CACHE GROUP statement.

  Then after issuing the TRUNCATE TABLE statement on the cached Oracle table, use a REFRESH CACHE GROUP statement to manually refresh the cache group.

- A LOAD CACHE GROUP statement can only be issued if the cache tables are empty, unless the cache group is dynamic.

  See "Loading and refreshing a cache group" on page 5-2 for more information about the LOAD CACHE GROUP and REFRESH CACHE GROUP statements.

  See "Dynamic cache groups" on page 4-36 for more information about dynamic cache groups.
The automatic refresh state must be PAUSED before you can issue a LOAD CACHE GROUP statement on the cache group, unless the cache group is dynamic, in which case the automatic refresh state must be PAUSED or ON. The LOAD CACHE GROUP statement cannot contain a WHERE clause, unless the cache group is dynamic, in which case the WHERE clause must be followed by a COMMIT EVERY \( n \) ROWS clause.

See "Using a WHERE clause" on page 4-24 for more information about WHERE clauses in cache group definitions and operations.

The automatic refresh state must be PAUSED before you can issue a REFRESH CACHE GROUP statement on the cache group. The REFRESH CACHE GROUP statement cannot contain a WHERE clause.

All tables and columns referenced in WHERE clauses when creating, loading or unloading the cache group must be fully qualified. For example:

```
user_name.table_name and user_name.table_name.column_name
```

To use the AUTOREFRESH cache group attribute in a user managed cache group, all of the cache tables must be specified with the PROPAGATE cache table attribute or all of the cache tables must be specified the READONLY cache table attribute.

You cannot specify the AUTOREFRESH cache group attribute in a user managed cache group that contains cache tables that explicitly use the NOT PROPAGATE cache table attribute.

LRU aging cannot be specified on the cache group, unless the cache group is dynamic where LRU aging is defined by default.

See "LRU aging" on page 4-31 for more information about LRU aging.

In Example 4-7, the update_anywhere_customers cache group uses the AUTOREFRESH cache group attribute.

### Altering a cache group

After creating an automatic refresh cache group, you can use an ALTER CACHE GROUP statement to change the cache group’s automatic refresh mode, interval or state. You cannot use ALTER CACHE GROUP to instantiate automatic refresh for a cache group that was originally created without automatic refresh defined.

If you change a cache group’s automatic refresh state to OFF or drop a cache group that has an automatic refresh operation in progress:

- The automatic refresh operation stops if the setting of the LockWait DSN attribute is greater than 0. The ALTER CACHE GROUP or DROP CACHE GROUP statement preempts the automatic refresh operation.

- The automatic refresh operation continues if the LockWait attribute is set to 0. The ALTER CACHE GROUP or DROP CACHE GROUP statement is blocked until the automatic refresh operation completes or the statement fails with a lock timeout error.

**Example 4-9  Altering the automatic refresh attributes of a cache group**

The following statements change the automatic refresh mode, interval and state of the customer_orders cache group:

```
ALTER CACHE GROUP customer_orders SET AUTOREFRESH MODE FULL
ALTER CACHE GROUP customer_orders SET AUTOREFRESH INTERVAL 30 SECONDS
ALTER CACHE GROUP customer_orders SET AUTOREFRESH STATE ON
```
**Manually creating Oracle objects for automatic refresh cache groups**

If you manually created the Oracle objects used to enforce the predefined behaviors of an automatic refresh cache group as described in "Manually create Oracle objects used to manage caching of Oracle data" on page 3-10, you need to set the automatic refresh state to OFF when creating the cache group.

Then you need to run the ttiSql utility's cachesqlget command with the INCREMENTAL_AUTOREFRESH option and the INSTALL flag as the cache manager user. This command generates a SQL*Plus script used to create a log table and a trigger in the Oracle database for each Oracle table that is cached in the automatic refresh cache group. These Oracle objects are used to track updates on the cached Oracle tables so that the updates can be automatically refreshed to the cache tables.

Next use SQL*Plus to run the script generated by the ttiSql utility's cachesqlget command as the sys user. Then use an ALTER CACHE GROUP statement to change the automatic refresh state of the cache group to PAUSED.

**Example 4–10  Creating a read-only cache group when Oracle objects were manually created**

The first statement creates a read-only cache group customer_orders with the automatic refresh state set to OFF. The SQL*Plus script generated by the ttiSql utility's cachesqlget command is saved to the /tmp/obj.sql file. The last statement changes the automatic refresh state of the cache group to PAUSED.

```
CREATE READONLY CACHE GROUP customer_orders
AUTOREFRESH STATE OFF
FROM oratt.customer
(cust_num NUMBER(6) NOT NULL,
 region VARCHAR2(10),
 name VARCHAR2(50),
 address VARCHAR2(100),
 PRIMARY KEY(cust_num)),
oratt.orders
(ord_num NUMBER(10) NOT NULL,
cust_num NUMBER(6) NOT NULL,
 when_placed DATE NOT NULL,
 when_shipped DATE NOT NULL,
 PRIMARY KEY(ord_num),
 FOREIGN KEY(cust_num) REFERENCES oratt.customer(cust_num))
```

```
% ttIsql "DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> cachesqlget INCREMENTAL_AUTOREFRESH customer_orders INSTALL /tmp/obj.sql;
Command> exit

% sqlplus sys as sysdba
Enter password: password
SQL> @/tmp/obj
SQL> exit

ALTER CACHE GROUP customer_orders SET AUTOREFRESH STATE PAUSED
```

**Using a WHERE clause**

A cache table definition in a CREATE CACHE GROUP statement can contain a WHERE clause to restrict the rows to cache in the TimesTen database for particular cache group types.

You can also specify a WHERE clause in a LOAD CACHE GROUP, UNLOAD CACHE GROUP, REFRESH CACHE GROUP or FLUSH CACHE GROUP statement for
particular cache group types. Some statements, such as LOAD CACHE GROUP and REFRESH CACHE GROUP, may result in concatenated WHERE clauses in which the WHERE clause for the cache table definition is evaluated before the WHERE clause in the LOAD CACHE GROUP or REFRESH CACHE GROUP statement.

The following restrictions apply to WHERE clauses used in cache table definitions and cache group operations:

- WHERE clauses can only be specified in the cache table definitions of a CREATE CACHE GROUP statement for read-only and user managed cache groups.
- A WHERE clause can be specified in a LOAD CACHE GROUP statement except on an explicitly loaded automatic refresh cache group.
  
  See "Loading and refreshing a cache group" on page 5-2 for more information about the LOAD CACHE GROUP statement.
- A WHERE clause can be specified in a REFRESH CACHE GROUP statement except on an automatic refresh cache group.
  
  See "Loading and refreshing a cache group" on page 5-2 for more information about the REFRESH CACHE GROUP statement.
- A WHERE clause can be specified in a FLUSH CACHE GROUP statement on a user managed cache group that allows committed updates on the TimesTen cache tables to be flushed to the cached Oracle tables.
  
  See "Flushing a user managed cache group" on page 5-14 for more information about the FLUSH CACHE GROUP statement.
- WHERE clauses in a CREATE CACHE GROUP statement cannot contain a subquery. Therefore, each WHERE clause cannot reference any table other than the one in its cache table definition. However, a WHERE clause in a LOAD CACHE GROUP, UNLOAD CACHE GROUP, REFRESH CACHE GROUP or FLUSH CACHE GROUP statement may contain a subquery.
- A WHERE clause in a LOAD CACHE GROUP, REFRESH CACHE GROUP or FLUSH CACHE GROUP statement can reference only the root table of the cache group, unless the WHERE clause contains a subquery.
- WHERE clauses in the cache table definitions are only enforced when the cache group is manually loaded or refreshed, or the cache tables are dynamically loaded. If a cache table is updatable, you can insert or update a row such that the WHERE clause in the cache table definition for that row is not satisfied.
- All tables and columns referenced in WHERE clauses when creating, loading, refreshing, unloading or flushing the cache group must be fully qualified. For example:
  
  `user_name.table_name` and `user_name.table_name.column_name`.

In Example 4–8, both the oratt.active_customer and oratt.orderdetails tables contain a WHERE clause.

**Proper placement of WHERE clause in a CREATE CACHE GROUP statement**

In a multiple-table cache group, a WHERE clause in a particular table definition should not reference any table in the cache group other than the table itself. For example, the following CREATE CACHE GROUP statements are valid:

```sql
CREATE READONLY CACHE GROUP customer_orders
FROM oratt.customer
  (cust_num NUMBER(6) NOT NULL,
   region VARCHAR2(10),
```

Defining Cache Groups 4-25
Creating a cache group

```
CREATE READONLY CACHE GROUP customer_orders
FROM oratt.customer
(cust_num NUMBER(6) NOT NULL,
 region VARCHAR2(10),
 name VARCHAR2(50),
 address VARCHAR2(100),
 PRIMARY KEY(cust_num)),
oratt.orders
(ord_num NUMBER(10) NOT NULL,
cust_num NUMBER(6) NOT NULL,
when_placed DATE NOT NULL,
when_shipped DATE NOT NULL,
 PRIMARY KEY(ord_num),
 FOREIGN KEY(cust_num) REFERENCES oratt.customer(cust_num))
WHERE (oratt.customer.cust_num < 100),
```

The following statement is not valid because the WHERE clause in the child table's definition references its parent table:

```
CREATE READONLY CACHE GROUP customer_orders
FROM oratt.customer
(cust_num NUMBER(6) NOT NULL,
 region VARCHAR2(10),
 name VARCHAR2(50),
 address VARCHAR2(100),
 PRIMARY KEY(cust_num)),
oratt.orders
(ord_num NUMBER(10) NOT NULL,
cust_num NUMBER(6) NOT NULL,
when_placed DATE NOT NULL,
when_shipped DATE NOT NULL,
 PRIMARY KEY(ord_num),
 FOREIGN KEY(cust_num) REFERENCES oratt.customer(cust_num))
WHERE (oratt.customer.cust_num < 100)
```

Similarly, the following statement is not valid because the WHERE clause in the parent table's definition references its child table:

```
CREATE READONLY CACHE GROUP customer_orders
FROM oratt.customer
(cust_num NUMBER(6) NOT NULL,
 region VARCHAR2(10),
 name VARCHAR2(50),
 address VARCHAR2(100),
 PRIMARY KEY(cust_num))
WHERE (oratt.orders.cust_num < 100),
oratt.orders
(ord_num NUMBER(10) NOT NULL,
cust_num NUMBER(6) NOT NULL,
```
Referencing Oracle PL/SQL functions in a WHERE clause

A user-defined PL/SQL function in the Oracle database can be invoked indirectly in a WHERE clause within a CREATE CACHE GROUP, LOAD CACHE GROUP, or REFRESH CACHE GROUP (for dynamic cache groups only) statement. After creating the function, create a public synonym for the function. Then grant the EXECUTE privilege on the function to PUBLIC.

For example, in the Oracle database:

```sql
CREATE OR REPLACE FUNCTION get_customer_name (
c_cust oratt.customer.cust_num%TYPE) RETURN VARCHAR2 IS
    c_name oratt.customer.name%TYPE;
BEGIN
    SELECT name INTO c_name FROM oratt.customer WHERE cust_num = c_cust;
    RETURN c_name;
END get_customer_name
```

```sql
CREATE PUBLIC SYNONYM retname FOR get_customer_name
GRANT EXECUTE ON get_customer_name TO PUBLIC
```

Then in the TimesTen database, for example, you can create a cache group with a WHERE clause that references the Oracle public synonym that was created for the function:

```sql
CREATE READONLY CACHE GROUP top_customer
FROM oratt.customer
(cust_num NUMBER(6) NOT NULL,
region VARCHAR2(10),
name VARCHAR2(50),
address VARCHAR2(100),
PRIMARY KEY(cust_num))
WHERE name = retname(100)
```

For cache group types that allow a WHERE clause on a LOAD CACHE GROUP or REFRESH CACHE GROUP statement, you can invoke the function indirectly by referencing the public synonym that was created for the function. For example, you can use the following LOAD CACHE GROUP statement to load the AWT cache group:

```sql
LOAD CACHE GROUP new_customers WHERE name = retname(101) COMMIT EVERY 0 ROWS;
```

**ON DELETE CASCADE cache table attribute**

The ON DELETE CASCADE cache table attribute can be specified for cache tables in any cache group type. ON DELETE CASCADE specifies that when rows containing referenced key values are deleted from a parent table, rows in child tables with dependent foreign keys are also deleted.

**Example 4–11 Using the ON DELETE CASCADE cache table attribute**

The following statement uses the ON DELETE CASCADE cache table attribute on the child table’s foreign key definition:

```sql
CREATE READONLY CACHE GROUP customer_orders
FROM oratt.customer
```
Creating a cache group

(cust_num NUMBER(6) NOT NULL,
region VARCHAR2(10),
name VARCHAR2(50),
address VARCHAR2(100),
PRIMARY KEY(cust_num)),
oratt.orders
(ord_num NUMBER(10) NOT NULL,
cust_num NUMBER(6) NOT NULL,
when_placed DATE NOT NULL,
when_shipped DATE NOT NULL,
PRIMARY KEY(ord_num),
FOREIGN KEY(cust_num) REFERENCES oratt.customer(cust_num) ON DELETE CASCADE)

All paths from a parent table to a child table must be either "delete" paths or "do not
delete" paths. There cannot be some "delete" paths and some "do not delete" paths
from a parent table to a child table. Specify the ON DELETE CASCADE cache table
attribute for child tables on a "delete" path.

The following restrictions apply when using the ON DELETE CASCADE cache table
attribute:

- For AWT and SWT cache groups, and for TimesTen cache tables in user managed
cache groups that use the PROPAGATE cache table attribute, foreign keys in cache
tables that use the ON DELETE CASCADE cache table attribute must be a proper
subset of the foreign keys in the cached Oracle tables that use the ON DELETE
CASCADE attribute. ON DELETE CASCADE actions on the cached Oracle tables
are applied to the TimesTen cache tables as individual deletes. ON DELETE
CASCADE actions on the cache tables are applied to the cached Oracle tables as a
cascaded operation.

- Matching of foreign keys between the TimesTen cache tables and the cached
Oracle tables is enforced only when the cache group is being created. A cascade
delete operation may not work if the foreign keys on the cached Oracle tables are
altered after the cache group is created.

See "CREATE CACHE GROUP" in Oracle TimesTen In-Memory Database SQL Reference
for more information about the ON DELETE CASCADE cache table attribute.

**UNIQUE HASH ON cache table attribute**

The UNIQUE HASH ON cache table attribute can be specified for cache tables in any
cache group type. UNIQUE HASH ON specifies that a hash index rather than a range
index is created on the primary key columns of the cache table. The columns specified
in the hash index must be identical to the columns in the primary key. The UNIQUE
HASH ON cache table attribute is also used to specify the size of the hash index.

**Example 4–12 Using the UNIQUE HASH ON cache table attribute**

The following statement uses the UNIQUE HASH ON cache table attribute on the
cache table’s definition.

CREATE ASYNCHRONOUS WRITETHROUGH CACHE GROUP new_customers
FROM oratt.customer
(cust_num NUMBER(6) NOT NULL,
region VARCHAR2(10),
name VARCHAR2(50),
address VARCHAR2(100),
PRIMARY KEY(cust_num))
UNIQUE HASH ON (cust_num) PAGES = 100
Caching Oracle synonyms

You can cache a private synonym in an AWT, SWT or user managed cache group that does not use the AUTOREFRESH cache group attribute. The private synonym can reference a public or private synonym, but it must eventually reference a table because it is the table that is actually being cached.

The table that is directly or indirectly referenced by the cached synonym can be owned by a user other than the Oracle user with the same name as the owner of the cache group that caches the synonym. The table must reside in the same Oracle database as the synonym. The cached synonym itself must be owned by the Oracle user with the same name as the owner of the cache group that caches the synonym.

Caching Oracle LOB data

You can cache Oracle large object (LOB) data in TimesTen cache groups. TimesTen caches the data as follows:

- Oracle CLOB data is cached as TimesTen VARCHAR2 data.
- Oracle BLOB data is cached as TimesTen VARBINARY data.
- Oracle NCLOB data is cached as TimesTen NVARCHAR2 data.

Example 4–13  Caching Oracle LOB data

Create a table in the Oracle database that has LOB fields.

```sql
CREATE TABLE t (
  i INT NOT NULL PRIMARY KEY,
  c CLOB,
  b BLOB,
  nc NCLOB);
```

Insert values into the Oracle table. The values are implicitly converted to LOB data types.

```sql
INSERT INTO t VALUES (1,
  RPAD('abcdefg8', 2048, 'abcdefg8'),
  HEXTORAW(RPAD('123456789ABCDEF8', 4000, '123456789ABCDEF8')),
  RPAD('abcdefg8', 2048, 'abcdefg8'));
```

1 row inserted.

Create a dynamic AWT cache group and start the replication agent.

```sql
CREATE DYNAMIC ASYNCHRONOUS WRITETHROUGH CACHE GROUP cg1
FROM t
{ i INT NOT NULL PRIMARY KEY,
  c VARCHAR2(4194304 BYTE),
  b VARBINARY(4194304),
  nc NVARCHAR2(2097152) }

CALL ttrepstart;
```
Load the data dynamically into the TimesTen cache group.

```
SELECT * FROM t WHERE i = 1;
```

I:  1
C: abcdefg8abcdefg8abcdefg8...
B: 123456789ABCDEF8123456789...
NC: abcdefg8abcdefg8abcdefg8...

1 row found.

**Restrictions on caching Oracle LOB data**

These restrictions apply to caching Oracle LOB data in TimesTen cache groups:

- Passthrough queries cannot bind parameters as CLOB, BLOB or NCLOB data types.
- Column size is enforced when a cache group is created. VARBINARY, VARCHAR2 and NVARCHAR2 data types have a size limit of 4 megabytes. Values that exceed the user-defined column size are truncated at run time without notification.
- Empty values in fields with CLOB and BLOB data types are initialized but not populated with data. Empty CLOB and BLOB fields are treated as follows:
  - Empty LOB fields in the Oracle database are returned as NULL values.
  - Empty BLOB fields are loaded into the TimesTen cache as NULL values.
  - Empty VARCHAR2 and VARBINARY fields in the TimesTen cache are propagated as NULL values.
  - TimesTen has no equivalent for the Oracle EMPTY_CLOB() and EMPTY_BLOB() SQL clauses.

In addition, cache groups that are configured for automatic refresh operations have these restrictions on caching LOB data:

- When LOB data is updated in the Oracle database by OCI functions or the DBMS_LOB PL/SQL package, the data is not automatically refreshed in the TimesTen cache group. This occurs because TimesTen caching depends on Oracle triggers, and Oracle triggers are not executed when these types of updates occur. TimesTen does not notify the user that updates have occurred without being refreshed in TimesTen.
- Automatic refresh operations update a complete row in the TimesTen cache. Thus the cached LOB data may appear to be updated in TimesTen when no change has occurred in the LOB data in the Oracle database.

**Implementing aging on a cache group**

You can define an aging policy on a cache group which specifies the aging type, the aging attributes, and the aging state. TimesTen supports two aging types, least recently used (LRU) aging and time-based aging.

LRU aging deletes the least recently used or referenced data based on a specified database usage range. Time-based aging deletes data based on a specified data lifetime and frequency of the aging process. You can use both LRU and time-based aging in the same TimesTen database, but you can define only one aging policy on a particular cache group.

An aging policy is specified in the cache table definition of the root table in a CREATE CACHE GROUP statement and applies to all cache tables in the cache group because
Implementing aging on a cache group

Aging is performed at the cache instance level. When rows are aged out or deleted from the cache tables, the rows in the cached Oracle table are not deleted.

You can add an aging policy to a cache group by using an ALTER TABLE statement on the root table. You can change the aging policy of a cache group by using ALTER TABLE statements on the root table to drop the existing aging policy and then add a new aging policy.

This section describes cache group definitions that contain an aging policy. The topics include:

- LRU aging
- Time-based aging
- Manually scheduling an aging process
- Configuring a sliding window

**LRU aging**

LRU aging enables you to maintain the amount of memory used in a TimesTen database within a specified threshold by deleting the least recently used data. LRU aging can be defined for all cache group types except explicitly loaded automatic refresh cache groups. LRU aging is defined by default on dynamic cache groups.

Define an LRU aging policy for a cache group by using the AGING LRU clause in the cache table definition of the CREATE CACHE GROUP statement. Aging occurs automatically if the aging state is set to its default of ON.

**Example 4–14  Defining an LRU aging policy on a cache group**

The following statement defines an LRU aging policy on the AWT cache group `new_customers`:

```sql
CREATE ASYNCHRONOUS WRITETHROUGH CACHE GROUP new_customers
FROM oratt.customer
(cust_num NUMBER(6) NOT NULL,
 region VARCHAR2(10),
 name VARCHAR2(50),
 address VARCHAR2(100),
 PRIMARY KEY(cust_num))
AGING LRU ON
```

Use the `ttAgingLRUConfig` built-in procedure to set the LRU aging attributes as a user with the ADMIN privilege. The attribute settings apply to all tables in the TimesTen database that have an LRU aging policy defined and an aging state of ON.

The following are the LRU aging attributes:

- **LowUsageThreshold**: The TimesTen database’s space usage (the ratio of the permanent partition’s in-use size over the partition’s allocated size) at or below which LRU aging is deactivated. The default low usage threshold is .8 (80 percent).
- **HighUsageThreshold**: The TimesTen database’s space usage above which LRU aging is activated. The default high usage threshold is .9 (90 percent).
- **AgingCycle**: The frequency in which aging occurs, in minutes. The default aging cycle is 1 minute.
Example 4–15 Setting the LRU aging attributes

The following procedure call specifies that the aging process checks every 5 minutes to see if the TimesTen database’s permanent partition space usage is above 95 percent. If it is, the least recently used data is automatically aged out or deleted until the space usage is at or below 75 percent.

CALL ttAgingLRUConfig(.75, .95, 5)

If you set a new value for AgingCycle after an LRU aging policy has been defined on a cache group, the next time aging occurs is based on the current system time and the new aging cycle. For example, if the original aging cycle was 15 minutes and LRU aging occurred 10 minutes ago, aging is expected to occur again in 5 minutes. However, if you change the aging cycle to 30 minutes, aging next occurs 30 minutes from the time you call ttAgingLRUConfig with the new aging cycle setting.

If a row has been accessed or referenced since the last aging cycle, it is not eligible for LRU aging in the current aging cycle. A row is considered to be accessed or referenced if at least one of the following is true:

- The row is used to build the result set of a SELECT or an INSERT ... SELECT statement.
- The row has been marked to be updated or deleted in a pending transaction.

In a multiple-table cache group, if a row in a child table has been accessed or referenced since the last aging cycle, then neither the related row in the parent table nor the row in the child table is eligible for LRU aging in the current aging cycle.

The ALTER TABLE statement can be used to perform the following tasks associated with changing or defining an LRU aging policy on a cache group:

- Change the aging state of a cache group by specifying the root table and using the SET AGING clause.
- Add an LRU aging policy to a cache group that has no aging policy defined by specifying the root table and using the ADD AGING LRU clause.
- Drop the LRU aging policy on a cache group by specifying the root table and using the DROP AGING clause.

To change the aging policy of a cache group from LRU to time-based, use an ALTER TABLE statement on the root table with the DROP AGING clause to drop the LRU aging policy. Then use an ALTER TABLE statement on the root table with the ADD AGING USE clause to add a time-based aging policy.

You must stop the cache agent before you add, alter or drop an aging policy on an automatic refresh cache group.

Time-based aging

Time-based aging deletes data from a cache group based on the aging policy’s specified data lifetime and frequency. Time-based aging can be defined for all cache group types.

Define a time-based aging policy for a cache group by using the AGING USE clause in the cache table definition of the CREATE CACHE GROUP statement. Aging occurs automatically if the aging state is set to its default of ON.

The definitions of the Oracle tables that will be cached in the AWT cache group defined in Example 4–17 are defined in Example 4–16. The Oracle tables are owned by the schema user oratt. The oratt user must be granted the CREATE SESSION and RESOURCE privileges before it can create tables.
Example 4–16  Oracle table definitions

CREATE TABLE orders
(ord_num      NUMBER(10) NOT NULL PRIMARY KEY,
cust_num     NUMBER(6) NOT NULL,
when_placed  DATE NOT NULL,
when_shipped DATE NOT NULL)

CREATE TABLE order_item
(orditem_id NUMBER(12) NOT NULL PRIMARY KEY,
ord_num    NUMBER(10),
prod_num   VARCHAR2(6),
quantity   NUMBER(3),
FOREIGN KEY(ord_num) REFERENCES oratt.orders(ord_num))

The Oracle user with the same name as the TimesTen cache manager user must be granted the SELECT privilege on the oratt.orders and oratt.order_item tables in order for the cache manager user to create an AWT cache group that caches these tables. The Oracle cache administration user must be granted the INSERT, UPDATE and DELETE privileges on the oratt.orders and oratt.order_item tables for asynchronous writethrough operations to occur from the TimesTen cache tables to the cached Oracle tables.

Example 4–17  Defining a time-based aging policy on a cache group

The following statement defines a time-based aging policy on the AWT cache group ordered_items:

CREATE ASYNCHRONOUS WRITETHROUGH CACHE GROUP ordered_items
FROM oratt.orders
(ord_num      NUMBER(10) NOT NULL,
cust_num     NUMBER(6) NOT NULL,
when_placed  DATE NOT NULL,
when_shipped DATE NOT NULL,
PRIMARY KEY(ord_num))
AGING USE when_placed LIFETIME 45 DAYS CYCLE 60 MINUTES ON,
oratt.order_item
(orditem_id NUMBER(12) NOT NULL,
ord_num    NUMBER(10),
prod_num   VARCHAR2(6),
quantity   NUMBER(3),
PRIMARY KEY(orditem_id),
FOREIGN KEY(ord_num) REFERENCES oratt.orders(ord_num))

Cache instances that are greater than 45 days old based on the difference between the current system timestamp and the timestamp in the when_placed column of the oratt.orders table are candidates for aging. The aging process checks every 60 minutes to see if there are cache instances that can be automatically aged out or deleted from the cache tables.

The AGING USE clause requires the name of a non-nullable TIMESTAMP or DATE column used for time-based aging. We refer to this column as the timestamp column.

For each row, the value in the timestamp column stores the date and time when the row was most recently inserted or updated. The values in the timestamp column is maintained by your application. If the value of this column is unknown for particular rows and you do not want those rows to be aged out of the table, define the timestamp column with a large default value.

You can create an index on the timestamp column to optimize performance of the aging process.
You cannot add a column to an existing table and then use that column as the
timestamp column because added columns cannot be defined as non-nullable. You
cannot drop the timestamp column from a table that has a time-based aging policy
defined.

Specify the lifetime in days, hours, minutes or seconds after the LIFETIME keyword in
the AGING USE clause.

The value in the timestamp column is subtracted from the current system timestamp.
The result is then truncated to the specified lifetime unit (day, hour, minute, second)
and compared with the specified lifetime value. If the result is greater than the lifetime
value, the row is a candidate for aging.

After the CYCLE keyword, specify the frequency in which aging occurs in days, hours,
minutes or seconds. The default aging cycle is 5 minutes. If you specify an aging cycle
of 0, aging is continuous.

The ALTER TABLE statement can be used to perform the following tasks associated
with changing or defining a time-based aging policy on a cache group:

- Change the aging state of a cache group by specifying the root table and using the
  SET AGING clause.
- Change the lifetime by specifying the root table and using the SET AGING
  LIFETIME clause.
- Change the aging cycle by specifying the root table and using the SET AGING
  CYCLE clause.
- Add a time-based aging policy to a cache group that has no aging policy defined
  by specifying the root table and using the ADD AGING USE clause.
- Drop the time-based aging policy on a cache group by specifying the root table
  and using the DROP AGING clause.

To change the aging policy of a cache group from time-based to LRU, use an ALTER
TABLE statement on the root table with the DROP AGING clause to drop the
time-based aging policy. Then use an ALTER TABLE statement on the root table with
the ADD AGING LRU clause to add an LRU aging policy.

You must stop the cache agent before you add, alter or drop an aging policy on an
automatic refresh cache group.

**Manually scheduling an aging process**

Use the ttAgingScheduleNow built-in procedure to manually start a one-time aging
process on a specified table or on all tables that have an aging policy defined. The
aging process starts as soon as you call the procedure unless there is already an aging
process in progress. Otherwise the manually started aging process begins when the
aging process that is in progress has completed. After the manually started aging
process has completed, the start of the table’s next aging cycle is set to the time when
ttAgingScheduleNow was called if the table’s aging state is ON.

**Example 4–18 Starting a one-time aging process**

The following procedure call starts a one-time aging process on the oratt.orders
table based on the time ttAgingScheduleNow is called:

```sql
CALL ttAgingScheduleNow('oratt.orders')
```

Rows in the oratt.orders root table that are candidates for aging are deleted as
well as related rows in the oratt.order_item child table.
When you call `ttAgingScheduleNow`, the aging process starts regardless of whether the table's aging state is ON or OFF. If you want to start an aging process on a particular cache group, specify the name of the cache group's root table when you call the procedure. If `ttAgingScheduleNow` is called with no parameters, it starts an aging process and then resets the start of the next aging cycle on all tables in the TimesTen database that have an aging policy defined.

Calling `ttAgingScheduleNow` does not change the aging state of any table. If a table's aging state is OFF when you call the procedure, the aging process starts, but it is not scheduled to run again after the process has completed. To continue aging a table whose aging state is OFF, you must call `ttAgingScheduleNow` again or change the table's aging state to ON.

To manually control aging on a cache group, disable aging on the root table by using an `ALTER TABLE` statement with the `SET AGING OFF` clause. Then call `ttAgingScheduleNow` to start an aging process on the cache group.

### Configuring a sliding window

You can use time-based aging to implement a sliding window for a cache group. In a sliding window configuration, new rows are inserted into and old rows are deleted from the cache tables on a regular schedule so that the tables contain only the data that satisfies a specific time interval.

You can configure a sliding window for a cache group by using incremental automatic refresh mode and defining a time-based aging policy. The automatic refresh operation checks the timestamp of the rows in the cached Oracle tables to determine whether new data should be refreshed into the TimesTen cache tables. The system time and the time zone must be identical on the Oracle and TimesTen systems.

If the cache group does not use incremental automatic refresh mode, you can configure a sliding window by using a `LOAD CACHE GROUP`, `REFRESH CACHE GROUP`, or `INSERT` statement, or a dynamic load operation to bring new data into the cache tables.

#### Example 4–19 Defining a cache group with sliding window properties

The following statement configures a sliding window on the read-only cache group `recent_shipped_orders`:

```sql
CREATE READONLY CACHE GROUP recent_shipped_orders
AUTOREFRESH MODE INCREMENTAL INTERVAL 1440 MINUTES STATE ON
FROM oratt.orders
(ord_num NUMBER(10) NOT NULL,
cust_num NUMBER(6) NOT NULL,
when_placed DATE NOT NULL,
when_shipped DATE NOT NULL,
PRIMARY KEY(ord_num))
AGING USE when_shipped LIFETIME 30 DAYS CYCLE 24 HOURS ON
```

New data in the `oratt.orders` cached Oracle table are automatically refreshed into the `oratt.orders` TimesTen cache table every 1440 minutes. Cache instances that are greater than 30 days old based on the difference between the current system timestamp and the timestamp in the `when_shipped` column are candidates for aging. The aging process checks every 24 hours to see if there are cache instances that can be aged out of the cache tables. Therefore, this cache group stores orders that have been shipped within the last 30 days.
The automatic refresh interval and the lifetime used for aging determine the duration that particular rows remain in the cache tables. It is possible for data to be aged out of the cache tables before it has been in the cache tables for its lifetime. For example, for a read-only cache group if the automatic refresh interval is 3 days and the lifetime is 30 days, data that is already 3 days old when it is refreshed into the cache tables is deleted after 27 days because aging is based on the timestamp stored in the rows of the cached Oracle tables that gets loaded into the TimesTen cache tables, not when the data is refreshed into the cache tables.

Dynamic cache groups

The data in a dynamic cache group is loaded on demand. For example, a call center application may not want to preload all of its customers' information into TimesTen as it may be very large. Instead it can use a dynamic cache group so that a specific customer's information is loaded only when needed such as when the customer calls or logs onto the system.

Any system managed cache group type (read-only, AWT, SWT) can be defined as a dynamic cache group. A user managed cache group can be defined as a dynamic cache group unless it uses both the AUTOREFRESH cache group attribute and the PROPAGATE cache table attribute.

Use the CREATE DYNAMIC CACHE GROUP statement to create a dynamic cache group.

**Example 4–20 Dynamic read-only cache group**

This following statement creates a dynamic read-only cache group `online_customers` that caches the `oratt.customer` table:

```
CREATE DYNAMIC READONLY CACHE GROUP online_customers
FROM oratt.customer
(cust_num NUMBER(6) NOT NULL,
 region VARCHAR2(10),
 name VARCHAR2(50),
 address VARCHAR2(100),
 PRIMARY KEY(cust_num))
```

With an explicitly loaded cache group, data is initially loaded into the cache tables from the cached Oracle tables using a LOAD CACHE GROUP statement. With a dynamic cache group, data may also be loaded into the cache tables using a LOAD CACHE GROUP statement. However, with a dynamic cache group, data is typically loaded automatically when its cache tables are referenced by a SELECT, INSERT, or UPDATE statement and the data is not found in the tables resulting in a cache miss. See “Dynamically loading a cache group” on page 5-10 for more information.

With both explicitly loaded and dynamic cache groups, a LOAD CACHE GROUP statement loads into their cache tables qualified data that exists in the cached Oracle tables but not in the TimesTen cache tables. However, if a row exists in a cache table but a newer version exists in the cached Oracle table, a LOAD CACHE GROUP statement does not load that row into the cache table even if it satisfies the predicate of the statement.

By contrast, a REFRESH CACHE GROUP statement reloads qualifying rows that exists in the cache tables, effectively refreshing the content of the cache. For an explicitly loaded cache group, the rows that are refreshed are all the rows that satisfy the predicate of the REFRESH CACHE GROUP statement. However, for a dynamic cache group, the rows that are refreshed are the ones that satisfy the predicate and already...
exist in the cache tables. In other words, rows that end up being refreshed are the ones that have been updated or deleted in the cached Oracle table, but not the ones that have been inserted. Therefore, a refresh operation processes only the rows that are already in the cache tables. No new rows are loaded into the cache tables of a dynamic cache group as a result of a refresh.

The data in the cache instance of a dynamic read-only cache group is consistent with the data in the corresponding rows of the Oracle tables. At any instant in time, the data in a cache instance of an explicitly loaded cache group is consistent with the data in the corresponding rows of the Oracle tables, taking into consideration the state and the interval settings for autorefresh.

The data in a dynamic cache group is subject to aging as LRU aging is defined by default. You can use the `ttAgingLRUConfig` built-in procedure to override the default or current LRU aging attribute settings for the aging cycle and TimesTen database space usage thresholds. Alternatively, you can define time-based aging on a dynamic cache group to override LRU aging. Rows in a dynamic AWT cache group must be propagated to Oracle before they become candidates for aging.

**Global cache groups**

An Oracle table cannot be cached in more than one cache group within the same TimesTen database. However, the table can be cached in separate cache groups in different TimesTen databases. If the table is cached in separate AWT cache groups and the same cache instance is updated simultaneously on multiple TimesTen databases, there is no guarantee as to the order in which the updates will be propagated to the cached Oracle table. Also, the contents of the updated cache table will be inconsistent between the TimesTen databases.

A TimesTen cache grid prevents this problem by providing users with Oracle databases a means to horizontally scale out cache groups across multiple systems with read/write data consistency across the TimesTen databases. A cache grid is a set of TimesTen databases that collectively manage the application data.

Tables that are cached in separate cache groups within different TimesTen databases must be cached in global cache groups in order for the cache grid to manage consistency of the cache instances across the grid members when updates are committed on the cache tables of the cache group.

Only a dynamic AWT cache group can be defined as global cache group.

The following is the definition of the Oracle table that will be cached in the dynamic AWT global cache group that is defined in Example 4–21. The Oracle table is owned by the schema user `oratt`. The `oratt` user must be granted the CREATE SESSION and RESOURCE privileges before it can create tables.

```
CREATE TABLE subscriber
(subscriberid NUMBER(10) NOT NULL PRIMARY KEY,
 name VARCHAR2(100) NOT NULL,
 minutes_balance NUMBER(5) NOT NULL,
 last_call_duration NUMBER(4) NOT NULL)
```

The Oracle user with the same name as the TimesTen cache manager user must be granted the SELECT privilege on the `oratt.subscriber` table in order for the cache manager user to create an AWT cache group that caches this table. The Oracle cache administration user must be granted the INSERT, UPDATE and DELETE privileges on the `oratt.subscriber` table for asynchronous writethrough operations to occur from the TimesTen cache table to the cached Oracle table.
Use the CREATE DYNAMIC ASYNCHRONOUS WRITETHROUGH GLOBAL CACHE GROUP statement to create a dynamic AWT global cache group.

**Example 4–21 Dynamic AWT global cache group**

The following statement creates a dynamic AWT global cache group `subscriber_accounts` that caches the `oratt.subscriber` table:

```sql
CREATE DYNAMIC ASYNCHRONOUS WRITETHROUGH GLOBAL CACHE GROUP subscriber_accounts
FROM oratt.subscriber
(subscriberid NUMBER(10) NOT NULL PRIMARY KEY,
name VARCHAR2(100) NOT NULL,
minutes_balance NUMBER(5) NOT NULL,
last_call_duration NUMBER(4) NOT NULL)
```

When a subscriber to a prepaid telephone account makes a call, the cache instance that contains his or her account balance is loaded into the `oratt.subscriber` cache table of the `subscriber_accounts` global cache group within one of the cache grid members. The grid member that the cache instance is loaded into becomes the owner of the cache instance. Other grid members cannot access the cache instance until the owner has updated the balance of minutes and the duration of the last call, and the committed update has been propagated to the cached Oracle table.

To ensure consistency among the grid members, an Oracle table that is cached in a global cache group in a TimesTen database should not also be cached in a local cache group in another TimesTen database within the same cache grid. In addition, the Oracle table should not be cached in a global cache group in another TimesTen database within a different cache grid.

For cache tables in a dynamic AWT global cache group, a particular cache instance can only be read or updated by one grid member at a time, which we refer to as the owner of the cache instance. When the owner no longer has a pending transaction on any row of the cache instance, another grid member can take ownership by reading or updating that instance. The owner relinquishes ownership of a cache instance when that instance has been deleted from that grid member as a result of:

- Aging
- A DELETE statement issued on the cache table
- An UNLOAD CACHE GROUP statement issued on the cache group
- A request from another grid member to take ownership of that instance

The owner relinquishes ownership of all its cache instances if that grid member detaches from its cache grid.

Read data consistency between nodes of a cache grid is only guaranteed when using SERIALIZABLE isolation level on the node where cache instances are being read. When using the default READ COMMITTED isolation level, a connection on a grid node that is reading a cache instance may see a data value that has been subsequently updated to a new value by another connection in the same or a different node.

The cache tables in a dynamic AWT global cache group can be populated using any of the following operations:

- Dynamic load operation
- INSERT statement on the cache tables (but not an INSERT INTO ... SELECT FROM statement)
- LOAD CACHE GROUP ... COMMIT EVERY n ROWS statement (can only be used if all the other grid members do not own any of the cache instances to be loaded)
See "Dynamically loading a cache group" on page 5-10 for information about a dynamic load operation.

A grid member can take ownership of a cache instance that is currently owned by another grid member using any of the following operations:

- Dynamic load operation
- LOAD CACHE GROUP ... WITH ID statement

A REFRESH CACHE GROUP statement can be issued on a global cache group only if it contains a WITH ID clause.

You can set the CacheGridMsgWait DSN attribute to the maximum number of seconds that a grid member performing a dynamic load operation to acquire ownership of a cache instance waits for the owner to relinquish the instance. The owner cannot relinquish ownership of a cache instance if it has a pending transaction on any row of the instance. The default maximum wait time is 60 seconds.

An INSERT statement issued on a cache table in a dynamic AWT global cache group fails if the unique key value in the inserted row already exists in the cached Oracle table.

When using a LOAD CACHE GROUP ... COMMIT EVERY n ROWS statement, if any of the cache instances to be loaded within a transaction are owned by another grid member, an error is returned. The transaction is then rolled back and no cache instances are loaded within the failed transaction.

To prevent conflicts that can occur if you update the same row in a TimesTen cache table and the cached Oracle table concurrently, update only the cache table. The cached Oracle table should not be updated directly.

A TimesTen database that is a member of a cache grid can contain local and global cache groups. Only cache tables in global cache groups are guaranteed to be consistent among the grid members.

### Start the replication agent

After you have created the dynamic AWT global cache group, start the replication agent on the TimesTen database as the cache manager user, if it is not already running:

```
% ttIsql "DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> call ttRepStart;
Command> exit
```

### Attach a TimesTen database to a cache grid

All standalone TimesTen databases, and the active and standby master databases of an active standby pair that contain global cache groups must attach to the cache grid that they are associated with in order to update the cache tables of the global cache groups. Attaching the databases to the grid allow the databases to become members of the grid so that cache instances in the cache tables of the global cache groups can maintain consistency among the databases within the grid.

#### Example 4–22 Attaching a TimesTen database to a cache grid

Attach the first standalone database to the ttGrid cache grid that it is associated with by calling the ttGridAttach built-in procedure as the cache manager user. The node number for a standalone TimesTen database is 1.
In this example, *alone1* is a name that is used to uniquely identify the grid member, *sys1* is the host name of the TimesTen system where the first standalone database resides, and *5001* is the TCP/IP port for the first standalone database's cache agent process:

```
% ttIsql "DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> call ttGridAttach(1, 'alone1', 'sys1', 5001);
Command> exit
```

Specify a unique port for the cache agent on each TimesTen database that attaches to the grid. Calling the `ttGridAttach` built-in procedure automatically starts the cache agent on the TimesTen database if it is not already running.

If a grid member did not detach from its grid before its cache agent process was stopped, the member automatically reattaches to its grid when its cache agent is automatically or manually restarted if its cache agent start policy is manual or always.

See "Configuring a cache grid" on page 3-15 for more information about a cache grid.
This chapter describes operations that can be performed on cache groups. It includes the following topics:

- Transmitting updates between the TimesTen and Oracle databases
- Loading and refreshing a cache group
- Dynamically loading a cache group
- Flushing a user managed cache group
- Unloading a cache group
- Setting a passthrough level
- Cache performance

You can use SQL statements or SQL Developer to perform most of the operations in this chapter. For more information about SQL Developer, see Oracle SQL Developer Support for Oracle TimesTen In-Memory Database User’s Guide.

### Transmitting updates between the TimesTen and Oracle databases

You can use the following SQL statements to manually transmit committed updates between the TimesTen cache tables and the cached Oracle tables:

<table>
<thead>
<tr>
<th>SQL statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAD CACHE GROUP</td>
<td>Load cache instances that are not in the TimesTen cache tables from the cached Oracle tables</td>
</tr>
<tr>
<td>REFRESH CACHE GROUP</td>
<td>Replace cache instances in the TimesTen cache tables with current data from the cached Oracle tables</td>
</tr>
<tr>
<td>FLUSH CACHE GROUP</td>
<td>Propagate committed updates on the TimesTen cache tables to the cached Oracle tables. Only applicable for user managed cache groups.</td>
</tr>
</tbody>
</table>

For AWT, SWT, and user managed cache groups that use the PROPAGATE cache table attribute, committed updates on the TimesTen cache tables are automatically propagated to the cached Oracle tables.

See “Asynchronous writethrough (AWT) cache group” on page 4-9 for more information about AWT cache groups.

See “Synchronous writethrough (SWT) cache group” on page 4-13 for more information about SWT cache groups.
Loading and refreshing a cache group

You can manually insert or update cache instances in the TimesTen cache tables from the cached Oracle tables using either a LOAD CACHE GROUP or REFRESH CACHE GROUP statement. The differences between loading and refreshing a cache group are:

- LOAD CACHE GROUP only loads committed inserts on the cached Oracle tables into the TimesTen cache tables. New cache instances are loaded into the cache tables, but cache instances that already exist in the cache tables are not updated or deleted even if the corresponding rows in the cached Oracle tables have been updated or deleted. A load operation is primarily used to initially populate a cache group.

- REFRESH CACHE GROUP replaces cache instances in the TimesTen cache tables with the most current data from the cached Oracle tables including cache instances that are already exist in the cache tables. A refresh operation is primarily used to update the contents of a cache group with committed updates on the cached Oracle tables after the cache group has been initially populated.

For an explicitly loaded cache group, a refresh operation is equivalent to issuing an UNLOAD CACHE GROUP statement followed by a LOAD CACHE GROUP statement on the cache group. In effect, all committed inserts, updates and deletes on the cached Oracle tables are refreshed into the cache tables. New cache instances may be loaded into the cache tables. Cache instances that already exist in the cache tables are updated or deleted if the corresponding rows in the cached Oracle tables have been updated or deleted. See “Unloading a cache group” on page 5-14 for more information about the UNLOAD CACHE GROUP statement.

For a dynamic cache group, a refresh operation only refreshes committed updates and deletes on the cached Oracle tables into the cache tables because only existing cache instances in the cache tables are refreshed. New cache instances are not loaded into the cache tables so after the refresh operation completes, the cache tables will contain either the same or fewer number of cache instances. To load new cache instances into the cache tables of a dynamic cache group, use a LOAD CACHE GROUP statement or perform a dynamic load operation. See
"Dynamically loading a cache group" on page 5-10 for more information about a dynamic load operation.

For most cache group types, you can use a WHERE clause in a LOAD CACHE GROUP or REFRESH CACHE GROUP statement to restrict the rows to be loaded or refreshed into the cache tables.

If the cache table definitions use a WHERE clause, only rows that satisfy the WHERE clause are loaded or refreshed into the cache tables even if the LOAD CACHE GROUP or REFRESH CACHE GROUP statement does not use a WHERE clause.

A REFRESH CACHE GROUP statement can be issued on a global cache group only if it contains a WITH ID clause.

If the cache group has a time-based aging policy defined, only cache instances where the timestamp in the root table’s row is within the aging policy’s lifetime are loaded or refreshed into the cache tables.

To prevent a load or refresh operation from processing a large number of cache instances within a single transaction which can greatly reduce concurrency and throughput, you must use the COMMIT EVERY n ROWS clause to specify a commit frequency unless you are using the WITH ID clause. If you specify COMMIT EVERY 0 ROWS, the load or refresh operation is processed in a single transaction.

A LOAD CACHE GROUP or REFRESH CACHE GROUP statement that uses the COMMIT EVERY n ROWS clause must be performed in its own transaction without any other operations within the same transaction.

**Example 5–1 Loading a cache group**

The following statement loads new cache instances into the TimesTen cache tables in the customer_orders cache group from the cached Oracle tables:

```
LOAD CACHE GROUP customer_orders COMMIT EVERY 256 ROWS
```

**Example 5–2 Loading a cache group using a WHERE clause**

The following statement loads into the TimesTen cache tables in the new_customers cache group from the cached Oracle tables, new cache instances for customers whose customer number is greater than or equal to 5000:

```
LOAD CACHE GROUP new_customers WHERE (oratt.customer.cust_num >= 5000) COMMIT EVERY 256 ROWS
```

**Example 5–3 Refreshing a cache group**

The following statement refreshes cache instances in the TimesTen cache tables within the top_products cache group from the cached Oracle tables:

```
REFRESH CACHE GROUP top_products COMMIT EVERY 256 ROWS
```

**Example 5–4 Refreshing a cache group using a WHERE clause**

The following statement refreshes in the TimesTen cache tables within the update_anywhere_customers cache group from the cached Oracle tables, cache instances of customers located in zip code 60610:

```
REFRESH CACHE GROUP update_anywhere_customers WHERE (oratt.customer.zip = '60610') COMMIT EVERY 256 ROWS
```

For more information, see "LOAD CACHE GROUP" and "REFRESH CACHE GROUP" in *Oracle TimesTen In-Memory Database SQL Reference*. 

Cache Group Operations  5-3
The rest of this section includes these topics:

- Loading and refreshing an explicitly loaded cache group with automatic refresh
- Loading and refreshing a dynamic cache group with automatic refresh
- Loading and refreshing a cache group using a WITH ID clause
- Initiating an immediate automatic refresh
- Loading and refreshing a multiple-table cache group
- Improving the performance of loading or refreshing a large number of cache instances
- Example of manually loading and refreshing an explicitly loaded cache group
- Example of manually loading and refreshing a dynamic cache group

**Loading and refreshing an explicitly loaded cache group with automatic refresh**

If the automatic refresh state of an explicitly loaded cache group is PAUSED, the automatic refresh state is changed to ON after a LOAD CACHE GROUP or REFRESH CACHE GROUP statement issued on the cache group completes.

The following restrictions apply when manually loading or refreshing an explicitly loaded cache group with automatic refresh:

- A LOAD CACHE GROUP statement can only be issued if the cache tables are empty.
- The automatic refresh state must be PAUSED before you can issue a LOAD CACHE GROUP statement.
- The automatic refresh state must be PAUSED before you can issue a REFRESH CACHE GROUP statement.
- A LOAD CACHE GROUP statement cannot contain a WHERE clause.
- A LOAD CACHE GROUP or REFRESH CACHE GROUP statement cannot contain a WITH ID clause.
- A REFRESH CACHE GROUP statement cannot contain a WHERE clause.
- All tables and columns referenced in a WHERE clause when loading the cache group must be fully qualified. For example:

  ```
  user_name.table_name and user_name.table_name.column_name
  ```

When an automatic refresh operation occurs on an explicitly loaded cache group, all committed inserts, updates and deletes on the cached Oracle tables since the last automatic refresh cycle are refreshed into the cache tables. New cache instances may be loaded into the cache tables. Cache instances that already exist in the cache tables are updated or deleted if the corresponding rows in the cached Oracle tables have been updated or deleted.

**Loading and refreshing a dynamic cache group with automatic refresh**

If the automatic refresh state of a dynamic cache group is PAUSED, the automatic refresh state is changed to ON after any of the following events occur:

- Its cache tables are initially empty, and then a dynamic load, a LOAD CACHE GROUP or an unconditional REFRESH CACHE GROUP statement issued on the cache group completes
Its cache tables are not empty, and then an unconditional REFRESH CACHE GROUP statement issued on the cache group completes.

If the automatic refresh state of a dynamic cache group is PAUSED, the automatic refresh state remains at PAUSED after any of the following events occur:

- Its cache tables are initially empty, and then a REFRESH CACHE GROUP ... WITH ID statement issued on the cache group completes.
- Its cache tables are not empty, and then a dynamic load, a REFRESH CACHE GROUP ... WITH ID, or a LOAD CACHE GROUP statement issued on the cache group completes.

For a dynamic cache group, an automatic refresh operation only refreshes committed updates and deletes on the cached Oracle tables since the last automatic refresh cycle into the cache tables because only existing cache instances in the cache tables are refreshed. New cache instances are not loaded into the cache tables. To load new cache instances into the cache tables of a dynamic cache group, use a LOAD CACHE GROUP statement or perform a dynamic load operation. See "Dynamically loading a cache group" on page 5-10 for more information about a dynamic load operation.

The following restrictions apply when manually loading or refreshing a dynamic cache group with automatic refresh:

- The automatic refresh state must be PAUSED or ON before you can issue a LOAD CACHE GROUP statement.
- The automatic refresh state must be PAUSED before you can issue a REFRESH CACHE GROUP statement.
- A LOAD CACHE GROUP statement that contains a WHERE clause must include a COMMIT EVERY n ROWS clause after the WHERE clause.
- A REFRESH CACHE GROUP statement cannot contain a WHERE clause.
- All tables and columns referenced in a WHERE clause when loading the cache group must be fully qualified. For example: user_name.table_name and user_name.table_name.column_name

### Loading and refreshing a cache group using a WITH ID clause

The WITH ID clause of the LOAD CACHE GROUP or REFRESH CACHE GROUP statement enables you to load or refresh a cache group based on values of the primary key columns without having to use a WHERE clause. The WITH ID clause is more convenient than the equivalent WHERE clause if the primary key contains more than one column. Using the WITH ID clause allows you to load one cache instance at a time. It also enables you to roll back the transaction containing the load or refresh operation, if necessary, unlike the equivalent statement that uses a WHERE clause because using a WHERE clause also requires specifying a COMMIT EVERY n ROWS clause.

#### Example 5–5 Loading a cache group using a WITH ID clause

A cache group recent_orders contains a single cache table oratt.orderdetails with a primary key of (orderid, itemid). If a customer calls about an item within a particular order, the information can be obtained by loading the cache instance for the specified order number and item number.

Load the oratt.orderdetails cache table in the recent_orders cache group with the row whose value in the orderid column of the oratt.orderdetails cached Oracle table is 1756 and its value in the itemid column is 573:
LOAD CACHE GROUP recent_orders WITH ID (1756,573)

The following is an equivalent LOAD CACHE GROUP statement that uses a WHERE clause:
LOAD CACHE GROUP recent_orders WHERE orderid = 1756 and itemid = 573
COMMIT EVERY 256 ROWS

A LOAD CACHE GROUP or REFRESH CACHE GROUP statement issued on an automatic refresh cache group cannot contain a WITH ID clause unless the cache group is dynamic.

You cannot use the COMMIT EVERY \( n \) ROWS clause with the WITH ID clause.

**Initiating an immediate automatic refresh**

If the Oracle tables have been updated with data that needs to be applied to cache tables without waiting for the next automatic refresh operation, you can call the \texttt{ttCacheAutorefresh} built-in procedure. The \texttt{ttCacheAutorefresh} built-in procedure initiates an immediate refresh operation and resets the automatic refresh cycle to start at the moment you invoke \texttt{ttCacheAutorefresh}. The refresh operation is full or incremental depending on how the cache group is configured. The automatic refresh state must be ON when \texttt{ttCacheAutorefresh} is called.

The automatic refresh operation normally刷新es all cache groups sharing the same refresh interval in one transaction in order to preserve transactional consistency across these cache groups. Therefore, although you specify a specific cache group when you call \texttt{ttCacheAutorefresh}, the automatic refresh operation occurs in one transaction for all cache groups that share the automatic refresh interval with the specified cache group. If there is an existing transaction with table locks on objects that belong to the affected cache groups, \texttt{ttCacheAutorefresh} returns an error without taking any action.

You can choose to run \texttt{ttCacheAutorefresh} asynchronously (the default) or synchronously. In synchronous mode, \texttt{ttCacheAutorefresh} returns an error if the refresh operation fails.

After calling \texttt{ttCacheAutorefresh}, you must commit or roll back the transaction before subsequent work can be performed.

**Example 5–6 Calling \texttt{ttCacheAutorefresh}**

This example calls \texttt{ttCacheAutorefresh} for the \texttt{ttuser.western_customers} cache group, using asynchronous mode.

Command> call \texttt{ttCacheAutorefresh}('ttuser', 'western_customers');

**Loading and refreshing a multiple-table cache group**

If you are loading or refreshing a multiple-table cache group while the cached Oracle tables are concurrently being updated, set the isolation level in the TimesTen database to serializable before issuing the LOAD CACHE GROUP or REFRESH CACHE GROUP statement. This causes TimesTen to query the cached Oracle tables in a serializable fashion during the load or refresh operation so that the loaded or refreshed cache instances in the cache tables are guaranteed to be transactionally consistent with the corresponding rows in the cached Oracle tables. After you have loaded or refreshed the cache group, set the isolation level back to read committed for better concurrency when accessing elements in the TimesTen database.
Improving the performance of loading or refreshing a large number of cache instances

You can improve the performance of loading or refreshing a large number of cache instances into a cache group by using the PARALLEL clause of the LOAD CACHE GROUP or REFRESH CACHE GROUP statement. Specify the number of threads to use when processing the load or refresh operation. You can specify 1 to 10 threads. One thread fetches rows from the cached Oracle tables, while the other threads insert the rows into the TimesTen cache tables. Do not specify more threads than the number of CPUs available on your system or you may encounter decreased performance than if you had not used the PARALLEL clause.

You cannot use the WITH ID clause or the COMMIT EVERY 0 ROWS clause with the PARALLEL clause.

Example 5–7 Refreshing a cache group using a PARALLEL clause

The following statement refreshes cache instances in the TimesTen cache tables within the western_customers cache group from the cached Oracle tables using one thread to fetch rows from the cached Oracle tables and three threads to insert the rows into the cache tables:

```
REFRESH CACHE GROUP western_customers COMMIT EVERY 256 ROWS PARALLEL 4
```

Example of manually loading and refreshing an explicitly loaded cache group

The following is the definition of the Oracle table that will be cached in an explicitly loaded AWT cache group. The Oracle table is owned by the schema user oratt.

```
CREATE TABLE customer
(cust_num NUMBER(6) NOT NULL PRIMARY KEY,
 region   VARCHAR2(10),
 name     VARCHAR2(50),
 address  VARCHAR2(100))
```

The following is the data in the oratt.customer cached Oracle table.

<table>
<thead>
<tr>
<th>CUST_NUM</th>
<th>REGION</th>
<th>NAME</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>West</td>
<td>Frank Edwards</td>
<td>100 Pine St. Portland OR</td>
</tr>
<tr>
<td>2</td>
<td>East</td>
<td>Angela Wilkins</td>
<td>356 Olive St. Boston MA</td>
</tr>
<tr>
<td>3</td>
<td>Midwest</td>
<td>Stephen Johnson</td>
<td>7638 Walker Dr. Chicago IL</td>
</tr>
</tbody>
</table>

The following statement creates an explicitly loaded AWT cache group new_customers that caches the oratt.customer table:

```
CREATE ASYNCHRONOUS WRITETHROUGH CACHE GROUP new_customers
FROM oratt.customer
(cust_num NUMBER(6) NOT NULL,
 region   VARCHAR2(10),
 name     VARCHAR2(50),
 address  VARCHAR2(100),
 PRIMARY KEY(cust_num))
```

The oratt.customer TimesTen cache table is initially empty.

Command> SELECT * FROM oratt.customer;
0 rows found.

The following LOAD CACHE GROUP statement loads the three cache instances from the cached Oracle table into the TimesTen cache table:

```
Command> LOAD CACHE GROUP new_customers COMMIT EVERY 256 ROWS;
```
Loading and refreshing a cache group

3 cache instances affected.
Command> SELECT * FROM oratt.customer;
< 1, West, Frank Edwards, 100 Pine St. Portland OR >
< 2, East, Angela Wilkins, 356 Olive St. Boston MA >
< 3, Midwest, Stephen Johnson, 7638 Walker Dr. Chicago IL >

Update the cached Oracle table by inserting a new row, updating an existing row, and deleting an existing row:

SQL> INSERT INTO customer
2  VALUES (4, 'East', 'Roberta Simon', '3667 Park Ave. New York NY');
SQL> UPDATE customer SET name = 'Angela Peterson' WHERE cust_num = 2;
SQL> DELETE FROM customer WHERE cust_num = 3;
SQL> COMMIT;
SQL> SELECT * FROM customer;
CUST_NUM   REGION    NAME              ADDRESS
--------   -------   ---------------   ---------------------------
1   West      Frank Edwards     100 Pine St. Portland OR
2   East      Angela Peterson   356 Olive St. Boston MA
4   East      Roberta Simon     3667 Park Ave. New York NY

A REFRESH CACHE GROUP statement issued on an explicitly loaded cache group is processed by unloading and then reloading the cache group. As a result, the cache instances in the cache table matches the rows in the cached Oracle table.

Command> REFRESH CACHE GROUP new_customers COMMIT EVERY 256 ROWS;
3 cache instance affected.
Command> SELECT * FROM oratt.customer;
< 1, West, Frank Edwards, 100 Pine St. Portland OR >
< 2, East, Angela Peterson, 356 Olive St. Boston MA >
< 4, East, Roberta Simon, 3667 Park Ave. New York NY >

Example of manually loading and refreshing a dynamic cache group

The following is the definition of the Oracle table that will be cached in a dynamic AWT cache group. The Oracle table is owned by the schema user oratt.

CREATE TABLE customer
(cust_num NUMBER(6) NOT NULL PRIMARY KEY,
region   VARCHAR2(10),
name     VARCHAR2(50),
address  VARCHAR2(100))

The following is the data in the oratt.customer cached Oracle table.

<table>
<thead>
<tr>
<th>CUST_NUM</th>
<th>REGION</th>
<th>NAME</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>West</td>
<td>Frank Edwards</td>
<td>100 Pine St. Portland OR</td>
</tr>
<tr>
<td>2</td>
<td>East</td>
<td>Angela Wilkins</td>
<td>356 Olive St. Boston MA</td>
</tr>
<tr>
<td>3</td>
<td>Midwest</td>
<td>Stephen Johnson</td>
<td>7638 Walker Dr. Chicago IL</td>
</tr>
</tbody>
</table>

The following statement creates a dynamic AWT cache group new_customers that caches the oratt.customer table:

CREATE DYNAMIC ASYNCHRONOUS WRITETHROUGH CACHE GROUP new_customers
FROM oratt.customer
(cust_num NUMBER(6) NOT NULL,
region   VARCHAR2(10),
name     VARCHAR2(50),
address  VARCHAR2(100),
PRIMARY KEY(cust_num))
The `oratt.customer` TimesTen cache table is initially empty:

Command> SELECT * FROM oratt.customer;
0 rows found.

The following LOAD CACHE GROUP statement loads the three cache instances from the cached Oracle table into the TimesTen cache table:

Command> LOAD CACHE GROUP new_customers COMMIT EVERY 256 ROWS;
3 cache instances affected.
Command> SELECT * FROM oratt.customer;
< 1, West, Frank Edwards, 100 Pine St. Portland OR >
< 2, East, Angela Wilkins, 356 Olive St. Boston MA >
< 3, Midwest, Stephen Johnson, 7638 Walker Dr. Chicago IL >

Update the cached Oracle table by inserting a new row, updating an existing row, and deleting an existing row:

SQL> INSERT INTO customer
2 VALUES (4, 'East', 'Roberta Simon', '3667 Park Ave. New York NY');
SQL> UPDATE customer SET name = 'Angela Peterson' WHERE cust_num = 2;
SQL> DELETE FROM customer WHERE cust_num = 3;
SQL> COMMIT;
SQL> SELECT * FROM customer;
CUST_NUM   REGION    NAME              ADDRESS
--------   -------   ---------------   ---------------------------
1   West      Frank Edwards     100 Pine St. Portland OR
2   East      Angela Peterson   356 Olive St. Boston MA
4   East      Roberta Simon     3667 Park Ave. New York NY

A REFRESH CACHE GROUP statement issued on a dynamic cache group only refreshes committed updates and deletes on the cached Oracle tables into the cache tables. New cache instances are not loaded into the cache tables. Therefore, only existing cache instances are refreshed. As a result, the number of cache instances in the cache tables are either fewer than or the same as the number of rows in the cached Oracle tables.

Command> REFRESH CACHE GROUP new_customers COMMIT EVERY 256 ROWS;
2 cache instances affected.
Command> SELECT * FROM oratt.customer;
< 1, West, Frank Edwards, 100 Pine St. Portland OR >
< 2, East, Angela Peterson, 356 Olive St. Boston MA >

A subsequent LOAD CACHE GROUP statement loads one cache instance from the cached Oracle table into the TimesTen cache table because only committed inserts are loaded into the cache table. Therefore, only new cache instances are loaded. Cache instances that already exist in the cache tables are not changed as a result of a LOAD CACHE GROUP statement, even if the corresponding rows in the cached Oracle tables were updated or deleted.

Command> LOAD CACHE GROUP new_customers COMMIT EVERY 256 ROWS;
1 cache instance affected.
Command> SELECT * FROM oratt.customer;
< 1, West, Frank Edwards, 100 Pine St. Portland OR >
< 2, East, Angela Peterson, 356 Olive St. Boston MA >
< 4, East, Roberta Simon, 3667 Park Ave. New York NY >
Dynamically loading a cache group

In a dynamic cache group, data is automatically loaded into the TimesTen cache tables from the cached Oracle tables when a particular SELECT, UPDATE, DELETE or INSERT statement is issued on one of the cache tables and the data does not exist in the cache table but does exist in the cached Oracle table.

If a row in the cached Oracle table satisfies the statement's WHERE clause, the entire cache instance is loaded in order to maintain the defined relationships between primary keys and foreign keys of the parent and child tables. A dynamic load operation cannot load more than one row into the root table of the cache group.

Only cache instances whose rows satisfy the WHERE clause of the cache table definitions are loaded. If the cache group has a time-based aging policy defined, the timestamp in the root table's row must be within the aging policy's lifetime in order for the cache instance to be loaded.

Dynamic load can be used to reload a cache instance that was aged out or deleted. See "Implementing aging on a cache group" on page 4-30 for more information about defining an aging policy on a cache group.

To ensure that data in the cache tables within a dynamic cache group is consistent with the cached Oracle tables, update operations on the cache tables are processed differently in a dynamic cache group than in an explicitly loaded cache group.

When an UPDATE or DELETE statement is issued on a cache table in a dynamic AWT, SWT, or user managed cache group that does not use the READONLY cache table attribute, and no rows in the cache table satisfy the statement's WHERE clause, the matching row in the cached Oracle table, if it exists, is dynamically loaded into the cache table. The loaded row is then updated or deleted in the cache table. The update or delete operation is then propagated to the cached Oracle table if the cache table is in a dynamic AWT, SWT, or user managed cache group that uses the PROPAGATE cache table attribute.

The dynamic load is executed in a different transaction than the user transaction that triggers the dynamic load. The dynamic load transaction is committed before the SQL statement that triggers the dynamic load has finished execution. Thus if the user transaction is rolled back, the dynamically loaded data remains in the cache group.

An INSERT statement issued on a cache table in a dynamic cache group fails if the key value in the inserted row already exists in the cached Oracle table.

A SELECT statement that results in a cache instance being dynamically loaded from the cached Oracle tables into the TimesTen cache tables can be issued on a cache table in a dynamic cache group of any supported type.

You cannot dynamically load a cache instance into a cache table within a dynamic global cache group unless the accompanying TimesTen database is attached to a cache grid. See "Global cache groups" on page 4-37 for more information about global cache groups and attaching a TimesTen database to a cache grid.

Types of SQL statements for which dynamic load is available

Dynamic load is available only for the following types of statements issued on a cache table in a dynamic cache group:

- SELECT, UPDATE or DELETE with an equality expression on the primary key column. The equality expression can only contain constants or parameters. For example:

  SELECT * FROM oratt.customer WHERE cust_num=50
If the primary key is composite, the SELECT, UPDATE or DELETE statement must contain equality expressions on all of the primary key columns. For example:

```
UPDATE oratt.orderdetails SET quantity = 5 WHERE orderid=2280 AND itemid=663
```

- SELECT, UPDATE or DELETE with an equality expression on the foreign key column. The equality expression can only contain constants or parameters. For example:

```
DELETE FROM oratt.cust_interests WHERE custid=364
```

If the foreign key is composite, the SELECT, UPDATE or DELETE statement must contain equality expressions on all of the foreign key columns. For example:

```
SELECT * FROM oratt.orders WHERE ord_num=4955 AND cust_num=716
```

Dynamic loading based on a primary key search of the root table has faster performance than primary key searches on a child table or foreign key searches on a child table.

The SELECT, UPDATE or DELETE statements for which dynamic load is available must satisfy the following conditions:

- If the statement contains a subquery, the equality expression must be specified in the outermost query of the statement. The statement can only reference cache tables from one cache group but it can also reference non-cache tables.
- The statement cannot contain the UNION, INTERSECT or MINUS set operators.

### Example of dynamically loading a cache group

The following is the definition of the Oracle table that will be cached in a dynamic AWT cache group. The Oracle table is owned by the schema user `oratt`.

```sql
CREATE TABLE customer
(cust_num NUMBER(6) NOT NULL PRIMARY KEY,
 region VARCHAR2(10),
 name VARCHAR2(50),
 address VARCHAR2(100))
```

The following is the data in the `oratt.customer` cached Oracle table.

```
CUST_NUM   REGION    NAME              ADDRESS
----------   -------   ---------------   ---------------------------
1   West      Frank Edwards     100 Pine St., Portland OR
2   East      Angela Wilkins    356 Olive St., Boston MA
3   Midwest   Stephen Johnson   7638 Walker Dr., Chicago IL
```

The following statement creates a dynamic AWT cache group `new_customers` that caches the `oratt.customer` table:

```
CREATE DYNAMIC ASYNCHRONOUS WRITETHROUGH CACHE GROUP new_customers
FROM oratt.customer
(cust_num NUMBER(6) NOT NULL,
 region VARCHAR2(10),
 name VARCHAR2(50),
 address VARCHAR2(100),
 PRIMARY KEY(cust_num))
```

The `oratt.customer` TimesTen cache table is initially empty:

```
Command> SELECT * FROM oratt.customer;
```
Dynamically loading a cache group

The following SELECT statement dynamically loads one cache instance from the cached Oracle table into the TimesTen cache table:

Command> SELECT * FROM oratt.customer WHERE cust_num = 1;
< 1, West, Frank Edwards, 100 Pine St., Portland OR >

The following UPDATE statement dynamically loads one cache instance from the cached Oracle table into the TimesTen cache table, updates the instance in the cache table, and then automatically propagates the update to the cached Oracle table:

Command> UPDATE oratt.customer SET name = 'Angela Peterson' WHERE cust_num = 2;
Command> SELECT * FROM oratt.customer;
< 1, West, Frank Edwards, 100 Pine St., Portland OR >
< 2, East, Angela Peterson, 356 Olive St., Boston MA >

The following is the updated data in the oratt.customer cached Oracle table:

<table>
<thead>
<tr>
<th>CUST_NUM</th>
<th>REGION</th>
<th>NAME</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>West</td>
<td>Frank Edwards</td>
<td>100 Pine St., Portland OR</td>
</tr>
<tr>
<td>2</td>
<td>East</td>
<td>Angela Peterson</td>
<td>356 Olive St., Boston MA</td>
</tr>
<tr>
<td>3</td>
<td>Midwest</td>
<td>Stephen Johnson</td>
<td>7638 Walker Dr., Chicago IL</td>
</tr>
</tbody>
</table>

The following is the updated data in the oratt.customer cached Oracle table:

<table>
<thead>
<tr>
<th>CUST_NUM</th>
<th>REGION</th>
<th>NAME</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>West</td>
<td>Frank Edwards</td>
<td>100 Pine St., Portland OR</td>
</tr>
<tr>
<td>2</td>
<td>East</td>
<td>Angela Peterson</td>
<td>356 Olive St., Boston MA</td>
</tr>
</tbody>
</table>

Disabling dynamic loading

You can use the following mechanisms to disable dynamic loading on all cache tables in dynamic cache groups that are accessed within a particular connection:

- Set the DynamicLoadEnable DSN attribute to 0
- Call the SQLSetConnectOption() ODBC function with the TT_DYNAMIC_LOAD_ENABLE connection option and the value parameter set to 0. (There is no equivalent JDBC or OCI function to disable dynamic loading.)

```
rc = SQLSetConnectOption(hDbc, TT_DYNAMIC_LOAD_ENABLE, 0)
```

- Call the SQLSetConnectOption() ODBC function with the TT_DYNAMIC_LOAD_ENABLE connection option and the value parameter set to 1 to re-enable dynamic loading.

You can use the following mechanisms to disable dynamic loading on all cache tables in dynamic cache groups that are accessed within a particular transaction:

- Use the ttIsq1 utility's set dynamicloadenable 0 command
Dynamically loading a cache group

- Call the `ttOptSetFlag` built-in procedure with the `DynamicLoadEnable` flag and the optimizer value set to 0.

```sql
call ttOptSetFlag('DynamicLoadEnable', 0)
```

Call the `ttOptSetFlag` built-in procedure with the `DynamicLoadEnable` flag and the optimizer value set to 1 to re-enable dynamic loading.

The `DynamicLoadEnable` flag setting takes effect when a statement is prepared and it is the setting that is used when the statement is executed even if the setting has changed from the time the statement was prepared to when the statement is executed. After the transaction has been committed or rolled back, the original connection setting takes effect for all subsequently prepared statements.

Displaying dynamic load errors

You can configure TimesTen to return an error if a SELECT, UPDATE or DELETE statement does not meet the requirements stated in "Types of SQL statements for which dynamic load is available" on page 5-10. The `DynamicLoadErrorMode` DSN attribute controls what happens when an application executes a SQL operation against a dynamic cache group and the SQL operation cannot use dynamic load. With a value of 0, the SQL operation executes against whatever data is already in the TimesTen cache tables and returns a result based on that data with no error indicated. With a value of 1, any statement that cannot use dynamic load (even if it does not need dynamic load) fails with an error indicating that it does not comply with dynamic load requirements.

You can use the following mechanisms to return an error if a statement within a particular connection does not comply with dynamic load requirements:

- Set the `DynamicLoadErrorMode` DSN attribute to 1

```sql
rc = SQLSetConnectOption(hDbc, TT_DYNAMIC_LOAD_ERROR_MODE, 1)
```

- Call the `ttIsql` utility's `set dynamicloaderrormode 1` command

- Call the `ttOptSetFlag` built-in procedure with the `DynamicLoadErrorMode` flag and the optimizer value set to 1.

```sql
call ttOptSetFlag('DynamicLoadErrorMode', 1)
```

- Call the `ttOptSetFlag` built-in procedure with the `DynamicLoadErrorMode` flag and the optimizer value set to 0 to suppress error reporting when a statement does not comply with dynamic load requirements.

The `DynamicLoadErrorMode` flag setting takes effect when a statement is prepared and it is the setting that is used when the statement is executed even if the setting has changed from the time the statement was prepared to when the
Flushing a user managed cache group

The FLUSH CACHE GROUP statement manually propagates committed inserts and updates on TimesTen cache tables in a user managed cache group to the cached Oracle tables. Deletes are not flushed or manually propagated. Committed inserts and updates on cache tables that use the PROPAGATE cache table attribute cannot be flushed to the cached Oracle tables because these operations are already automatically propagated to Oracle.

With automatic propagation, committed inserts, updates and deletes are propagated to Oracle in the order they were committed in TimesTen. A flush operation can manually propagate multiple committed transactions on cache tables to the cached Oracle tables.

You cannot flush a user managed cache group that uses the AUTOREFRESH cache group attribute.

You can flush a user managed cache group if at least one of its cache tables uses neither the PROPAGATE nor the READONLY cache table attribute.

You can use a WHERE clause or WITH ID clause in a FLUSH CACHE GROUP statement to restrict the rows to be flushed to the cached Oracle tables. See “FLUSH CACHE GROUP” in Oracle TimesTen In-Memory Database SQL Reference for more information.

Example 5–8  Flushing a cache group

The following statement manually propagates committed insert and update operations on the TimesTen cache tables in the western_customers cache group to the cached Oracle tables:

```
FLUSH CACHE GROUP western_customers
```

Unloading a cache group

You can delete some or all cache instances from the cache tables in a cache group with the UNLOAD CACHE GROUP statement. Unlike the DROP CACHE GROUP statement, the cache tables themselves are not dropped when a cache group is unloaded.

Use caution when using the UNLOAD CACHE GROUP statement with automatic refresh cache groups. An unloaded row can reappear in the cache table as the result of an automatic refresh operation if the row, or its related parent or child rows, are updated in the cached Oracle table.

Example 5–9  Unloading cache groups

The following statement deletes all cache instances from all cache tables in the customer_orders cache group:

```
UNLOAD CACHE GROUP customer_orders
```

The following equivalent statements delete the cache instance for customer number 227 from the cache tables in the new_customers cache group:

```
UNLOAD CACHE GROUP new_customers WITH ID (227)
UNLOAD CACHE GROUP new_customers WHERE (oratt.customer.cust_num = 227)
```
Unloading a cache group across all grid members

You can unload a cache group in all members of a cache grid by setting an optimizer flag. Before executing the UNLOAD CACHE GROUP statement, call the ttOptSetFlag built-in procedure and set the GlobalProcessing optimizer flag to 1:

```
CALL ttOptSetFlag('GlobalProcessing', 1);
```

Consider this statement:

```
UNLOAD CACHE GROUP customer WHERE customer_id=54321;
```

A local unload operation removes the customer record only if the record exists on the node where the statement is executed. A global unload operation removes the customer record regardless of which node contains the record.

Determining the number of cache instances affected by an operation

You can use the following mechanisms to determine how many cache instances were loaded by a LOAD CACHE GROUP statement, refreshed by a REFRESH CACHE GROUP statement, flushed by a FLUSH CACHE GROUP statement, or unloaded by an UNLOAD CACHE GROUP statement:

- Call the SQLRowCount() ODBC function
- Invoke the Statement.getUpdateCount() JDBC method
- Call the OCIAttrGet() OCI function with the OCI_ATTR_ROW_COUNT option

Setting a passthrough level

An application can issue statements on a TimesTen connection and either have the statement be executed in the TimesTen database or passed through to the Oracle database for execution.

Whether the statement is executed in the TimesTen or Oracle database depends on the composition of the statement and the setting of the PassThrough DSN attribute. You can set the PassThrough attribute to define which statements are to be executed locally in TimesTen and which are to be redirected to Oracle for execution.

PassThrough=0 is the default setting and specifies that all statements are to be executed in the TimesTen database.

As shown in Figure 5–1, you can set PassThrough=1 to specify that a statement referencing a table that does not exist in the TimesTen database is passed through to the Oracle database for execution. For example, an update on table A is executed in the TimesTen database because there exists a cache table A. However, an update on table G is passed through to Oracle for execution because no cache table G exists.

If PassThrough=1, a TimesTen built-in procedure call, such as ttCacheStart is executed in the TimesTen database, while procedure calls whose name TimesTen does not recognize are passed through to the Oracle database for execution. No DDL statements are passed through to Oracle. Except for SELECT statements, if a DML statement contains invalid TimesTen syntax it is not passed through to Oracle.
Setting a passthrough level

5-16 Oracle In-Memory Database Cache User's Guide

Figure 5–1 Behavior of the PassThrough=1 setting

PassThrough = 2  
For read-only cache groups and cache tables, DML statements are passed through to the Oracle database for execution.

For all other cache groups, the behavior is the same as PassThrough=1.

A read-only cache group or a user managed cache group that uses the READONLY cache table attribute prohibits updates on its cache tables. If an update operation references a cache table in either of these cache group types, you can set PassThrough=2 to pass the statement through to the Oracle database for execution. Otherwise, the behavior of PassThrough=2 is identical to PassThrough=1. For example, consider the user managed cache group shown in Figure 5–1. If cache table A uses the READONLY cache table attribute and PassThrough=2, an update operation on table A is passed through to the Oracle database for execution.

PassThrough=3 specifies that all statements are to be passed through to the Oracle database for execution, except for INSERT, UPDATE and DELETE statements issued on cache tables in a dynamic AWT global cache group as these statements are to be executed in the TimesTen database. See "Global cache groups" on page 4-37 for more information about global cache groups.

PassThrough=4 specifies that, in addition to the PassThrough=3 behavior, all SELECT statements issued on cache tables in a dynamic AWT global cache group that cannot be processed as a dynamic load operation are passed through to the Oracle database for execution.
PassThrough=5 is similar to the PassThrough=4 behavior except that the SELECT statements will not be passed through to the Oracle database for execution until all committed updates on cache tables in dynamic AWT global cache groups by previous transactions within the connection have been propagated to Oracle.

Passing through update operations to the Oracle database for execution is not recommended when issued on cache tables in an AWT or SWT cache group. Committed updates on cache tables in an AWT cache group are automatically propagated to the cached Oracle tables in asynchronous fashion. However, passing through an update operation to the Oracle database for execution within the same transaction as the update on the cache table in the AWT cache group renders the propagate of the cache table update synchronous, which may have undesired results. Committed updates on cache tables in an SWT cache group can result in self-deadlocks if, within the same transaction, updates on the same tables are passed through to the Oracle database for execution.

A PL/SQL block cannot be passed through to the Oracle database for execution. Also, you cannot pass through to Oracle for execution a reference to a stored procedure or function that is defined in the Oracle database but not in the TimesTen database.

For more information about how the PassThrough attribute setting determines which statements are executed in the TimesTen database and which are passed through to the Oracle database for execution and under what circumstances, see "PassThrough" in Oracle TimesTen In-Memory Database Reference.

---

**Note:** The passthrough feature is an OCI client. The OCI diagnostic framework installs signal handlers that may impact any signal handling that you use in your application. You can disable OCI signal handling by setting DIAG_SIGHANDLER_ENABLED=FALSE in the sqlnet.ora file. Refer to "Fault Diagnosability in OCI" in Oracle Call Interface Programmer’s Guide for information.

---

### Changing the passthrough level for a connection or transaction

You can override the current passthrough level using the ttIsql utility's set passthrough command which applies to the current transaction.

You can also override the setting for a specific transaction by calling the ttOptSetFlag built-in procedure with the PassThrough flag. The following procedure call sets the passthrough level to 3:

```sql
call ttOptSetFlag('PassThrough', 3)
```

The PassThrough flag setting takes effect when a statement is prepared and it is the setting that is used when the statement is executed even if the setting has changed from the time the statement was prepared to when the statement is executed. After the transaction has been committed or rolled back, the original connection setting takes effect for all subsequently prepared statements.

### Determining the passthrough setting for a statement

You can call the SQLGetStmtOption() ODBC function with the TT_STMT_PASSTHROUGH_TYPE statement option to determine whether a statement is to be executed in the TimesTen database or passed through to the Oracle database for execution.

```sql
rc = SQLGetStmtOption(hStmt, TT_STMT_PASSTHROUGH_TYPE, &execDB)
```
If `TT_STMT_PASSTHROUGH_NONE` is returned to the output parameter of the `SQLGetStmtOption()` ODBC function, the statement is to be executed in TimesTen. Otherwise, if `TT_STMT_PASSTHROUGH_ORACLE` is returned to the output parameter, the statement is to be passed through to Oracle for execution.

You may not always be certain whether a SQL statement will be executed in TimesTen or passed through to Oracle for execution. For example, if `PassThrough=1`, the syntax of a SELECT statement determines whether it is executed in TimesTen or passed through to Oracle. In this case, prepare the statement before calling `SQLGetStmtOption()` with the `TT_STMT_PASSTHROUGH_TYPE` statement option.

There is no equivalent JDBC or OCI function to determine the passthrough setting for a statement.

**Cache performance**

This section contains information about cache performance.

See *Oracle TimesTen In-Memory Database Troubleshooting Procedures Guide* for extensive information about monitoring autorefresh operations and improving autorefresh performance. See "Monitoring autorefresh cache groups" and "Poor autorefresh performance".

*Oracle TimesTen In-Memory Database Troubleshooting Procedures Guide* also has information about AWT cache group performance. See "Monitoring AWT performance" and "Possible causes of poor AWT performance".

**Dynamic load performance**

Dynamic loading based on a primary key search of the root table has faster performance than primary key searches on a child table or foreign key searches on a child table.

**Improving AWT throughput for mixed transactions and network latency**

By default, an AWT cache group uses SQL array execution to apply changes within TimesTen to the Oracle database. This method works well when the same type of operation is repeated. For example, array execution is very efficient when a user does an update that affects several rows of the table. Updates are grouped together and sent to the Oracle server in one batch.

Use the `cacheAWTMethod` first connection attribute to specify the PL/SQL execution method. When you specify the PL/SQL execution method, AWT bundles all pending operations into a single PL/SQL collection that is sent to the Oracle server to be executed. This method can improve AWT throughput when there are mixed transactions and network latency between TimesTen and the Oracle server.

PL/SQL execution method transparently falls back to SQL array execution mode temporarily when it encounters one of the following:

- A statement that is over 32761 bytes in length.
- A statement that references a column of type BINARY FLOAT, BINARY DOUBLE and VARCHAR/VARBINARY of length greater than 4000 bytes.

For more information, see "cacheAWTMethod" in *Oracle TimesTen In-Memory Database Reference*.
This chapter describes the tasks for creating a second standalone TimesTen database and an active standby pair, and attaching these members to the cache grid that was created in Chapter 3, "Setting Up a Caching Infrastructure". It includes the following topics:

- Creating and configuring a subsequent standalone TimesTen database
- Replicating cache tables
- Example of data sharing among the grid members
- Performing global queries on a cache grid
- Adding other elements to a cache grid or grid member

**Creating and configuring a subsequent standalone TimesTen database**

The following is the definition of the `cachealone2` DSN for the second standalone TimesTen database that will become a member of the `ttGrid` cache grid:

```
[cachealone2]
DataStore=/users/OracleCache/alone2
PermSize=64
OracleNetServiceName=orcl
DatabaseCharacterSet=WE8ISO8859P1
```

Start the `ttIsql` utility and connect to the `cachealone2` DSN as the instance administrator to create the database. Then create the cache manager user `cacheuser` whose name, in this example, is the same as the Oracle cache administration user. Then create a cache table user `oratt` whose name is the same as the Oracle schema user who will own the Oracle tables to be cached in the TimesTen database.

```
% ttIsql cachealone2
Command> CREATE USER cacheuser IDENTIFIED BY timesten;
Command> CREATE USER oratt IDENTIFIED BY timesten;
```

As the instance administrator, use the `ttIsql` utility to grant the cache manager user `cacheuser` the privileges required to perform the operations listed in Example 3–8:

```
Command> GRANT CREATE SESSION, CACHE_MANAGER, CREATE ANY TABLE TO cacheuser;
Command> exit
```

Start the `ttIsql` utility and connect to the `cachealone2` DSN as the cache manager user. Set the cache administration user name and password by calling the `ttCacheUidPwdSet` built-in procedure.

```
% ttIsql 'DSN=cachealone2;UID=cacheuser;PWD=timesten;OraclePWD=oracle'
```
Replicating cache tables

Command> call ttCacheUidPwdSet('cacheuser','oracle');

Associate the second standalone database to the ttGrid cache grid by calling the ttGridNameSet built-in procedure as the cache manager user:

Command> call ttGridNameSet('ttGrid');

The ttGrid cache grid was created from the first standalone TimesTen database. Since the grid already exists, it does not need to be created again.

If desired, you can test the connectivity between the second standalone TimesTen database and the Oracle database using the instructions stated in “Testing the connectivity between the TimesTen and Oracle databases” on page 3-17.

Start the cache agent on the second standalone database by calling the ttCacheStart built-in procedure as the cache manager user:

Command> call ttCacheStart;

Then create cache groups in the database as the cache manager user. For example, the following statement creates a dynamic AWT global cache group subscriber_accounts that caches the oratt.subscriber table:

```
CREATE DYNAMIC ASYNCHRONOUS WRITETHROUGH GLOBAL CACHE GROUP subscriber_accounts
FROM oratt.subscriber
(subscriberid NUMBER(10) NOT NULL PRIMARY KEY,
name VARCHAR2(100) NOT NULL,
minutes_balance NUMBER(5) NOT NULL,
last_call_duration NUMBER(4) NOT NULL)
```

The definition of the oratt.subscriber cached Oracle table is shown in “Global cache groups”.

If any AWT cache groups were created, start the replication agent on the TimesTen database by calling the ttRepStart built-in procedure as the cache manager user:

Command> call ttRepStart;

If any global cache groups were created, the database must attach to the cache grid that it is associated with in order to update the cache tables of the global cache groups. Attaching the database to the grid allows the database to become a member of the grid so that cache instances in the cache tables of the global cache groups can maintain consistency among the databases within the grid.

As the cache manager user, attach the second standalone database to the ttGrid cache grid that it is associated with by calling the ttGridAttach built-in procedure. The node number for a standalone TimesTen database is 1.

In the following example, alone2 is a name that is used to uniquely identify the grid member, sys2 is the host name of the TimesTen system where the second standalone database resides, and 5002 is the TCP/IP port for the second standalone database's cache agent process:

Command> call ttGridAttach(1,'alone2','sys2',5002);
Command> exit

Replicating cache tables

To achieve high availability, configure an active standby pair replication scheme for cache tables in a read-only cache group or an AWT cache group.
An active standby pair that replicates cache tables from one of these cache group types can automatically change the role of a TimesTen database as part of failover and recovery with minimal chance of data loss. Cache groups themselves provide resilience from Oracle database outages, further strengthening system availability. See "Administering an Active Standby Pair with Cache Groups" in Oracle TimesTen In-Memory Database TimesTen to TimesTen Replication Guide for more information.

An active standby pair replication scheme provides for high availability of a TimesTen database. Multiple grid members provide for high availability of a TimesTen cache grid. Oracle Real Application Clusters (Oracle RAC) provides for high availability of an Oracle database. For more information about using Oracle In-Memory Database Cache in an Oracle RAC environment, see "Using Oracle In-Memory Database Cache in an Oracle RAC Environment" on page 9-1.

Perform the following tasks to configure an active standby pair for TimesTen databases that cache Oracle tables:

- Create and configure the active master database
- Create and configure the standby master database
- Create and configure the read-only subscriber database

**Create and configure the active master database**

The following is the definition of the `cacheactive` DSN for the active master database of the active standby pair that will become a member of the `ttGrid` cache grid:

```
[cacheactive]
DataStore=/users/OracleCache/cacheact
PermSize=64
OracleNetServiceName=orcl
DatabaseCharacterSet=WE8ISO8859P1
```

Start the `ttIsql` utility and connect to the `cacheactive` DSN as the instance administrator to create the database. Then create the cache manager user `cacheuser` whose name, in this example, is the same as the Oracle cache administration user. Then create a cache table user `oratt` whose name is the same as the Oracle schema user who will own the Oracle tables to be cached in the TimesTen database.

```
% ttIsql cacheactive
Command> CREATE USER cacheuser IDENTIFIED BY timesten;
Command> CREATE USER oratt IDENTIFIED BY timesten;
```

As the instance administrator, use the `ttIsql` utility to grant the cache manager user `cacheuser` the privileges required to perform the operations listed in Example 3-8 as well as create an active standby pair replication scheme which requires the ADMIN privilege:

```
Command> GRANT CREATE SESSION, CACHE_MANAGER, 
       > CREATE ANY TABLE, ADMIN TO cacheuser;
Command> exit
```

Start the `ttIsql` utility and connect to the `cacheactive` DSN as the cache manager user. Set the cache administration user name and password by calling the `ttCacheUidPwdSet` built-in procedure.

```
% ttIsql 'DSN=cacheactive;UID=cacheuser;PWD=timesten;OraclePWD=oracle'
Command> call ttCacheUidPwdSet('cacheuser','oracle');
```
Associate the active master database to the ttGrid cache grid by calling the 
\texttt{ttGridNameSet} built-in procedure as the cache manager user:

\begin{verbatim}
Command> call ttGridNameSet('ttGrid');
\end{verbatim}

The \texttt{ttGrid} cache grid was created from the first standalone TimesTen database. Since 
the grid already exists, it does not need to be created again.

If desired, you can test the connectivity between the active master database and the 
Oracle database using the instructions stated in "Testing the connectivity between the 
TimesTen and Oracle databases" on page 3-17.

Start the cache agent on the active master database by calling the \texttt{ttCacheStart} 
built-in procedure as the cache manager user:

\begin{verbatim}
Command> call ttCacheStart;
\end{verbatim}

Then create cache groups in the database as the cache manager user. For example, the 
following statement creates a dynamic AWT global cache group \texttt{subscriber_accounts} 
that caches the \texttt{oratt.subscriber} table:

\begin{verbatim}
CREATE DYNAMIC ASYNCHRONOUS WRITETHROUGH GLOBAL CACHE GROUP subscriber_accounts
FROM oratt.subscriber
  (subscriberid       NUMBER(10) NOT NULL PRIMARY KEY,
   name               VARCHAR2(100) NOT NULL,
   minutes_balance    NUMBER(5) NOT NULL,
   last_call_duration NUMBER(4) NOT NULL)
\end{verbatim}

The definition of the \texttt{oratt.subscriber} cached Oracle table is shown in "Global 
cache groups".

As the cache manager user, create an active standby pair replication scheme in the 
active master database using a \texttt{CREATE ACTIVE STANDBY PAIR} statement.

In the following example, \texttt{cacheact}, \texttt{cachestand} and \texttt{subscr} are the file name 
prefixes of the checkpoint and transaction log files of the active master database, 
standby master database and read-only subscriber database. \texttt{sys3}, \texttt{sys4} and \texttt{sys5} are 
the host names of the TimesTen systems where the active master database, standby 
master database and read-only subscriber database reside, respectively.

\begin{verbatim}
Command> CREATE ACTIVE STANDBY PAIR cacheact ON 'sys3', cachestand ON 'sys4'
  > SUBSCRIBER subscr ON 'sys5';
\end{verbatim}

As the cache manager user, start the replication agent on the active master database by 
calling the \texttt{ttRepStart} built-in procedure. Then declare the database as the active 
master by calling the \texttt{ttRepStateSet} built-in procedure.

\begin{verbatim}
Command> call ttRepStart;
Command> call ttRepStateSet('active');
\end{verbatim}

If any global cache groups were created, the database must attach to the cache grid 
that it is associated with in order to update the cache tables of the global cache groups. 
Attaching the database to the grid allows the database to become a member of the grid 
so that cache instances in the cache tables of the global cache groups can maintain 
consistency among the databases within the grid.

As the cache manager user, attach the active master database to the \texttt{ttGrid} cache grid 
that it is associated with by calling the \texttt{ttGridAttach} built-in procedure. The node 
number for an active master database is 1.

In the following example:
Replicating cache tables

- `cacheact` is a name that is used to uniquely identify the active master database grid node
- `cachestand` is a name that is used to uniquely identify the standby master database grid node
- `sys3` is the host name of the TimesTen system where the active master database resides
- `sys4` is the host name of the TimesTen system where the standby master database resides
- `5003` is the TCP/IP port for the active master database's cache agent process
- `5004` is the TCP/IP port for the standby master database's cache agent process

```
Command> call ttGridAttach(1,'cacheact','sys3',5003,'cachestand','sys4',5004);
Command> exit
```

Create and configure the standby master database

The following is the definition of the `cachestandby` DSN for the standby master database of the active standby pair that will become a member of the `ttGrid` cache grid:

```
[cachestandby]
DataStore=/users/OracleCache/cachestand
PermSize=64
OracleNetServiceName=orcl
DatabaseCharacterSet=WE8ISO8859P1
```

As the instance administrator, create the standby master database as a duplicate of the active master database by running a `ttRepAdmin -duplicate` utility command from the standby master database's system. The instance administrator user name of the active master database's and standby master database's instances must be identical.

Use the `-keepCG` option so that cache tables in the active master database are duplicated as cache tables in the standby master database because the standby master database will have connectivity with the Oracle database.

In the following example:

- The `-from` option specifies the file name prefix of the active master database's checkpoint and transaction log files
- The `-host` option specifies the host name of the TimesTen system where the active master database resides
- The `-uid` and `-pwd` options specify a user name and password of a TimesTen internal user defined in the active master database that has been granted the ADMIN privilege
- The `-cacheuid` and `-cachepwd` options specify the Oracle cache administration user name and password
- `cachestandby` is the DSN of the standby master database

```
% ttRepAdmin -duplicate -from cacheact -host "sys3" -uid cacheuser -pwd timesten -cacheuid cacheuser -cachepwd oracle -keepCG cachestandby
```

Start the `ttIsql` utility and connect to the `cachestandby` DSN as the cache manager user. Set the cache administration user name and password by calling the `ttCacheUidPwdSet` built-in procedure.
Replicating cache tables

% ttSql "DSN=cachestandby;UID=cacheuser;PWD=times ten;OraclePWD=oracle"
Command> call ttCacheUidPwdSet('cacheuser','oracle');

The ttGrid cache grid was created from the first standalone TimesTen database. Since the grid already exists, it does not need to be created again.

The ttRepAdmin -duplicate -keepCG utility command associated the standby master database to the ttGrid cache grid so this association does not need to be done explicitly.

If desired, you can test the connectivity between the standby master database and the Oracle database using the instructions stated in “Testing the connectivity between the TimesTen and Oracle databases” on page 3-17.

Start the cache agent on the standby master database by calling the ttCacheStart built-in procedure as the cache manager user:
Command> call ttCacheStart;

As the cache manager user, start the replication agent on the standby master database by calling the ttRepStart built-in procedure.
Command> call ttRepStart;

If any global cache groups were created, the database must attach to the cache grid that it is associated with in order to update the cache tables of the global cache groups. Attaching the database to the grid allows the database to become a member of the grid so that cache instances in the cache tables of the global cache groups can maintain consistency among the databases within the grid.

As the cache manager user, attach the standby master database to the ttGrid cache grid that it is associated with by calling the ttGridAttach built-in procedure. The node number for a standby master database is 2. Use the same TCP/IP ports specified for the cache agent of the active master and standby master databases that were specified when configuring the active master database.

In the following example:
- cacheact is a name that is used to uniquely identify the active master database grid node
- cachestand is a name that is used to uniquely identify the standby master database grid node
- sys3 is the host name of the TimesTen system where the active master database resides
- sys4 is the host name of the TimesTen system where the standby master database resides
- 5003 is the TCP/IP port for the active master database's cache agent process
- 5004 is the TCP/IP port for the standby master database's cache agent process
Command> call ttGridAttach(2,'cacheact','sys3',5003,'cachestand','sys4',5004);
Command> exit

Create and configure the read-only subscriber database

The following is the definition of the rosubscriber DSN for the read-only subscriber database of the active standby pair:

[rosubscriber]
DataStore=/users/OracleCache/subscr
PermSize=64
DatabaseCharacterSet=WE8ISO8859P1

As the instance administrator, create the read-only subscriber database as a duplicate of the standby master database by running a ttRepAdmin -duplicate utility command from the read-only subscriber database system. The instance administrator user name of the standby master database instance and read-only subscriber database instance must be identical.

Use the -noKeepCG option so that cache tables in the standby master database are duplicated as regular tables in the read-only subscriber database because the read-only subscriber database will have no connectivity with the Oracle database. As a result, the read-only subscriber database will not be associated with a cache grid.

In the following example:

- The -from option specifies the file name prefix of the standby master database's checkpoint and transaction log files.
- The -host option specifies the host name of the TimesTen system where the standby master database resides.
- The -uid and -pwd options specify a user name and password of a TimesTen internal user defined in the standby master database that has been granted the ADMIN privilege.
- rosubscriber is the DSN of the read-only subscriber database.

% ttRepAdmin -duplicate -from cachestand -host "sys4" -uid cacheuser -pwd timesten -noKeepCG rosubscriber

As the cache manager user, start the replication agent on the read-only subscriber database by calling the ttRepStart built-in procedure.

% ttIsql 'DSN=rosubscriber;UID=cacheuser;PWD=timesten'
Command> call ttRepStart;
Command> exit

Example of data sharing among the grid members

The definition of the oratt.subscriber cached Oracle table is shown in "Global cache groups" on page 4-37.

The following is the data in the oratt.subscriber cached Oracle table.

<table>
<thead>
<tr>
<th>SUBSCRIBERID</th>
<th>NAME</th>
<th>MINUTES_BALANCE</th>
<th>LAST_CALL_DURATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Jane Anderson</td>
<td>75</td>
<td>15</td>
</tr>
<tr>
<td>1004</td>
<td>Robert Phillips</td>
<td>60</td>
<td>20</td>
</tr>
<tr>
<td>1005</td>
<td>William Ackerman</td>
<td>40</td>
<td>10</td>
</tr>
<tr>
<td>1009</td>
<td>Sandy Little</td>
<td>90</td>
<td>30</td>
</tr>
</tbody>
</table>

The oratt.subscriber TimesTen cache table in the subscriber_accounts global cache group is initially empty in all five TimesTen databases (cachealone1, cachealone2, cacheactive, cachestandby, rosubscriber):

Command> SELECT * FROM oratt.subscriber;
0 rows found.

Issue the following SELECT statement on the cachealone1 TimesTen database to dynamically load one cache instance from the cached Oracle table into the TimesTen cache table:
Performing global queries on a cache grid

If you want to access data on all the nodes of a cache grid, perform a global query. For example, consider this statement:

```
SELECT MAX(salary) FROM employees;
```

When global query processing is not enabled, the statement returns the maximum salary for the rows that exist on the local node. When global query processing is enabled, it returns the maximum salary across all employee records in the cache grid without changing ownership of the cache instance where the data is found. A global query can reference a cache table or a noncache table. The specified table must have the same definition across grid members.

Enable global query processing by setting an optimizer flag. Before executing a global query, call the ttOptSetFlag built-in procedure and set the GlobalProcessing optimizer flag to 1:

```
Command> ttOptSetFlag GlobalProcessing 1
```
CALL ttOptSetFlag('GlobalProcessing', 1);

Restrictions on global queries

Global queries have these restrictions:

- The query must reference exactly one table.
- The query cannot reference a global temporary table.
- The query cannot include a self join, a derived table or subqueries.
- ROWNUM and GROUP BY clauses cannot be used in the same query.

Adding other elements to a cache grid or grid member

If a database that contains a global cache group is attached to a cache grid, a subsequent database can attach to the same grid and become a grid member only if it contains a global cache group with the same definition as the global cache group in the database that is attached to the grid. The subsequent database cannot attach to the same grid if it contains more or fewer global cache groups than the database that is attached to the grid. Each database can contain a different number of local cache groups with non-matching definitions between the databases.

Before you can create a new dynamic AWT global cache group in a TimesTen database that is attached to a cache grid, stop the replication agent on the database. Then restart the replication agent after creating the global cache group. The new global cache group cannot be manually or dynamically loaded, and its cache tables cannot be updated until the cache group has been created with the same definition in all the grid members. In the standalone databases and the active master database, create the new global cache group manually. For the standby master database and the read-only subscriber databases, use the ttDestroy utility to drop the databases and a ttRepAdmin -duplicate utility command to re-create the databases so that they contain the new global cache group.
Adding other elements to a cache grid or grid member
This chapter describes how to manage and monitor various aspects of a caching system such as cache grids, cache groups and the cache agent process. It includes the following topics:

- Checking the status of the cache and replication agents
- Monitoring cache groups and cache grids
- Oracle objects used to manage a caching environment
- Impact of failed automatic refresh operations on TimesTen databases
- Dropping Oracle objects used by automatic refresh cache groups
- Monitoring the cache administration user's tablespace

Checking the status of the cache and replication agents

You can use either the ttAdmin or ttStatus utility to check whether the TimesTen cache agent and replication agent processes are running as well as determine each agent's start policy.

**Example 7–1 Using ttAdmin to determine the cache and replication agents status**

You can use a ttAdmin -query utility command to determine whether the cache and replication agents are running, and the cache and replication agent start policies for a TimesTen database:

```
% ttAdmin -query cachealone1
RAM Residence Policy            : inUse
Replication Agent Policy        : manual
Replication Manually Started    : True
Cache Agent Policy              : always
Cache Agent Manually Started    : True
```

For more information about the ttAdmin utility, see "ttAdmin" in Oracle TimesTen In-Memory Database Reference.

**Example 7–2 Using ttStatus to determine the cache and replication agents status**

You can use the ttStatus utility to determine whether the cache and replication agents are running, and the cache and replication agent start policies for all TimesTen databases in the installed instance:

```
% ttStatus
TimesTen status report as of Thu May 7 13:42:01 2009
```
Checking the status of the cache and replication agents

Daemon pid 9818 port 4173 instance myinst
TimesTen server pid 9826 started on port 4175

Data store /users/OracleCache/alone1
There are 38 connections to the data store
Shared Memory KEY 0x02011c82 ID 895844354
PL/SQL Memory KEY 0x03011c82 ID 895877123 Address 0xl0000000

<table>
<thead>
<tr>
<th>Type</th>
<th>PID</th>
<th>Context</th>
<th>Connection Name</th>
<th>ConnID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache Agent</td>
<td>1019</td>
<td>0x0828f840</td>
<td>Handler</td>
<td>2</td>
</tr>
<tr>
<td>Cache Agent</td>
<td>1019</td>
<td>0x083a3d40</td>
<td>Timer</td>
<td>3</td>
</tr>
<tr>
<td>Cache Agent</td>
<td>1019</td>
<td>0x0842d820</td>
<td>Aging</td>
<td>4</td>
</tr>
<tr>
<td>Cache Agent</td>
<td>1019</td>
<td>0x08664f8d8</td>
<td>Garbage Collector(-1580741728)</td>
<td>5</td>
</tr>
<tr>
<td>Cache Agent</td>
<td>1019</td>
<td>0x084d6ef8</td>
<td>Marker(-1580213344)</td>
<td>6</td>
</tr>
<tr>
<td>Cache Agent</td>
<td>1019</td>
<td>0xa5bb8058</td>
<td>DeadDsMonitor(-1579684960)</td>
<td>7</td>
</tr>
<tr>
<td>Cache Agent</td>
<td>1019</td>
<td>0x088b49a0</td>
<td>CacheGridEnv</td>
<td>14</td>
</tr>
<tr>
<td>Cache Agent</td>
<td>1019</td>
<td>0x089b9d0</td>
<td>CacheGridSend</td>
<td>15</td>
</tr>
<tr>
<td>Cache Agent</td>
<td>1019</td>
<td>0x089d0b20</td>
<td>CacheGridSend</td>
<td>16</td>
</tr>
<tr>
<td>Cache Agent</td>
<td>1019</td>
<td>0x08a619f8</td>
<td>CacheGridSend</td>
<td>17</td>
</tr>
<tr>
<td>Cache Agent</td>
<td>1019</td>
<td>0x08ace538</td>
<td>CacheGridRec</td>
<td>18</td>
</tr>
<tr>
<td>Cache Agent</td>
<td>1019</td>
<td>0x08b42e88</td>
<td>CacheGridRec</td>
<td>19</td>
</tr>
<tr>
<td>Cache Agent</td>
<td>1019</td>
<td>0x08bb77d8</td>
<td>CacheGridRec</td>
<td>20</td>
</tr>
<tr>
<td>Cache Agent</td>
<td>1019</td>
<td>0x08c2c128</td>
<td>CacheGridRec</td>
<td>21</td>
</tr>
<tr>
<td>Cache Agent</td>
<td>1019</td>
<td>0x08ca0a78</td>
<td>CacheGridRec</td>
<td>22</td>
</tr>
<tr>
<td>Cache Agent</td>
<td>1019</td>
<td>0x08d153c8</td>
<td>CacheGridRec</td>
<td>23</td>
</tr>
<tr>
<td>Cache Agent</td>
<td>1019</td>
<td>0x08d9418</td>
<td>CacheGridRec</td>
<td>24</td>
</tr>
<tr>
<td>Cache Agent</td>
<td>1019</td>
<td>0x08dfe668</td>
<td>CacheGridRec</td>
<td>25</td>
</tr>
<tr>
<td>Cache Agent</td>
<td>1019</td>
<td>0x08ef2f8b</td>
<td>CacheGridRec</td>
<td>26</td>
</tr>
<tr>
<td>Cache Agent</td>
<td>1019</td>
<td>0x08ee8020</td>
<td>CacheGridRec</td>
<td>27</td>
</tr>
<tr>
<td>Cache Agent</td>
<td>1019</td>
<td>0x08f5d088</td>
<td>CacheGridRec</td>
<td>28</td>
</tr>
<tr>
<td>Cache Agent</td>
<td>1019</td>
<td>0x08fd23f8</td>
<td>CacheGridRec</td>
<td>29</td>
</tr>
<tr>
<td>Cache Agent</td>
<td>1019</td>
<td>0x09047768</td>
<td>CacheGridRec</td>
<td>30</td>
</tr>
<tr>
<td>Replication</td>
<td>18051</td>
<td>0x08c3d900</td>
<td>RECEIVER</td>
<td>8</td>
</tr>
<tr>
<td>Replication</td>
<td>18051</td>
<td>0x08b53298</td>
<td>REPHOLD</td>
<td>9</td>
</tr>
<tr>
<td>Replication</td>
<td>18051</td>
<td>0x08af8138</td>
<td>REPLISTENER</td>
<td>10</td>
</tr>
<tr>
<td>Replication</td>
<td>18051</td>
<td>0x08a82f20</td>
<td>LOGFORCE</td>
<td>11</td>
</tr>
<tr>
<td>Replication</td>
<td>18051</td>
<td>0x08bce660</td>
<td>TRANSMITTER</td>
<td>12</td>
</tr>
<tr>
<td>Subdaemon</td>
<td>9822</td>
<td>0x080a2180</td>
<td>Manager</td>
<td>2032</td>
</tr>
<tr>
<td>Subdaemon</td>
<td>9822</td>
<td>0x080ff260</td>
<td>Rollback</td>
<td>2033</td>
</tr>
<tr>
<td>Subdaemon</td>
<td>9822</td>
<td>0x0854c38</td>
<td>Flusher</td>
<td>2034</td>
</tr>
<tr>
<td>Subdaemon</td>
<td>9822</td>
<td>0x085e3b00</td>
<td>Monitor</td>
<td>2035</td>
</tr>
<tr>
<td>Subdaemon</td>
<td>9822</td>
<td>0x0828fc10</td>
<td>Deadlock Detector</td>
<td>2036</td>
</tr>
<tr>
<td>Subdaemon</td>
<td>9822</td>
<td>0x082ed470</td>
<td>Checkpoint</td>
<td>2037</td>
</tr>
<tr>
<td>Subdaemon</td>
<td>9822</td>
<td>0x08345ed0</td>
<td>Aging</td>
<td>2038</td>
</tr>
<tr>
<td>Subdaemon</td>
<td>9822</td>
<td>0x083a1030</td>
<td>Log Marker</td>
<td>2039</td>
</tr>
<tr>
<td>Subdaemon</td>
<td>9822</td>
<td>0x083c190</td>
<td>AsyncMV</td>
<td>2040</td>
</tr>
<tr>
<td>Subdaemon</td>
<td>9822</td>
<td>0x084572f0</td>
<td>HistGC</td>
<td>2041</td>
</tr>
</tbody>
</table>

Replication policy : Manual
Replication agent is running.
Cache Agent policy : Always
TimesTen’s Cache agent is running for this data store
PL/SQL enabled.

The information displayed by the ttStatus utility include the following that pertains to Oracle In-Memory Database Cache for each TimesTen database in the installed instance:

- The names of the cache agent process threads that are connected to the TimesTen database
- The names of the replication agent process threads that are connected to the TimesTen database
Monitoring cache groups and cache grids

The following sections describe how to obtain information about cache grids and cache groups, and how to monitor the status of cache group operations:

- Using the ttIsq1 utility's cachegroups command
- Monitoring automatic refresh operations on cache groups
- Monitoring AWT cache group operations
- Obtaining information about cache grids
- Tracking DDL statements issued on cached Oracle tables

Using the ttIsq1 utility's cachegroups command

You can obtain information about cache groups in a TimesTen database using the ttIsq1 utility's cachegroups command.

Example 7–3  ttIsq1 utility's cachegroups command

```%
% ttIsq1 'DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle'
Command> cachegroups;
```

Cache Group CACHEUSER.RECENT_SHIPPED_ORDERS:

- Cache Group Type: Read Only
- Autorefresh: Yes
- Autorefresh Mode: Incremental
- Autorefresh State: On
- Autorefresh Interval: 1440 Minutes
- Autorefresh Status: ok
- Aging: Timestamp based uses column WHEN_SHIPPED lifetime 30 days cycle 24 hours
- Root Table: ORATT.ORDERS
- Table Type: Read Only

Cache Group CACHEUSER.SUBSCRIBER_ACCOUNTS:

- Cache Group Type: Asynchronous Writethrough global (Dynamic)
- Autorefresh: No
- Aging: LRU on
- Root Table: ORATT.SUBSCRIBER
- Table Type: Propagate

Cache Group CACHEUSER.WESTERN_CUSTOMERS:
Monitoring cache groups and cache grids

Cache Group Type: User Managed
Autorefresh: No
Aging: No aging defined

Root Table: ORATT.ACTIVE_CUSTOMER
Where Clause: (oratt.active_customer.region = 'West')
Table Type: Propagate

Child Table: ORATT.ORDERTAB
Table Type: Propagate

Child Table: ORATT.ORDERDETAILS
Where Clause: (oratt.orderdetails.quantity >= 5)
Table Type: Not Propagate

Child Table: ORATT.CUST_INTERESTS
Table Type: Read Only

3 cache groups found.

The information displayed by the ttIsql utility's cachegroups command include:
- Cache group type including whether the cache group is dynamic or global
- Automatic refresh attributes (mode, state, interval) and status, if applicable
- Aging policy, if applicable
- Name of root table and, if applicable, name of child tables
- Cache table WHERE clause, if applicable
- Cache table attributes (read-only, propagate, not propagate)

For more information about the ttIsql utility's cachegroups command, see "ttIsql" in Oracle TimesTen In-Memory Database Reference.

Monitoring automatic refresh operations on cache groups

You can use the following mechanisms to obtain information and statistics about automatic refresh operations on cache groups, and monitor when cache groups are automatically refreshed:

- Call the ttCacheAutorefreshStatsGet built-in procedure to obtain information about the last ten automatic refresh operations on a specified cache group:
  % ttIsql "DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
  Command> call ttCacheAutorefreshStatsGet('cacheuser','recent_shipped_orders');

  The information returned includes the start time and duration of each automatic refresh operation, and the number of rows that were refreshed into the cache tables during each operation.

  For more information about the ttCacheAutorefreshStatsGet built-in procedure, see Oracle TimesTen In-Memory Database Reference and "Using the ttCacheAutorefreshStatsGet procedure" in Oracle TimesTen In-Memory Database Troubleshooting Procedures Guide.

- Run the TimesTen_install_dir/oraclescripts/cacheInfo.sql SQL*Plus script as the cache administration user to display change log table information for all Oracle tables cached in an automatic refresh cache group:
% cd TimesTen_install_dir/oraclescripts
% sqlplus cacheuser/oracle
SQL> @cacheInfo

*************Autorefresh Objects Information  ***************
Host name: sys1
Timesten datastore name: /users/OracleCache/alone1
Cache table name: ORATT.ORDERS
Change log table name: tt_05_69245_L
Number of rows in change log table: 1
Maximum logseq on the change log table: 0
Timesten has autorefreshed updates upto logseq: 0
Number of updates waiting to be autorefreshed: 0
Number of updates that has not been marked with a valid logseq: 0

The information returned for each change log table includes the name of the change log table, the name of its corresponding TimesTen cache table, the number of rows in the change log table, and the number of updates in the change log table that have not been automatically refreshed into the cache table.

For more details about the information returned from the cacheInfo.sql script, see "Displaying information from the change log tables" in Oracle TimesTen In-Memory Database Troubleshooting Procedures Guide.

- Use the ttDaemonLog utility to view informational, warning and error messages in the support log that pertain to automatic refresh operations:

  % ttDaemonLog -show ora

  For details about the ttDaemonLog utility, see Oracle TimesTen In-Memory Database Reference.

  For details about the automatic refresh operations messages in the support log, see "Understanding messages about autorefresh in the support log" and "Diagnosing autorefresh failure" in Oracle TimesTen In-Memory Database Troubleshooting Procedures Guide.

- Use the ttTraceMon utility to generate trace records for the AUTOREFRESH component:

  % ttTraceMon cachealone1
  Trace monitor; empty line to exit
  Trace> level autorefresh 2
  Trace> flush
  Trace> outfile autorefresh.out
  Trace> <Press Enter to exit ttTraceMon>

  The information shown in the trace records include the name of the cache group that each automatic refresh operation was performed on, the duration of each operation, and the number of rows that were refreshed into the cache tables during each operation.

  For more information about the ttTraceMon utility, see Oracle TimesTen In-Memory Database Reference.

  For details about the automatic refresh trace records, see "AUTOREFRESH tracing" in Oracle TimesTen In-Memory Database Troubleshooting Procedures Guide.

- Enable SNMP traps to be thrown by setting the -enabled flag to 1 in the TimesTen_install_dir/info/snmp.ini and monitor for the various traps from the TimesTen_install_dir/mibs/TimesTen-MIB.txt file that apply to automatic refresh error conditions.
For more information about SNMP traps and which traps pertain to automatic refresh operations, see "Diagnostics Through SNMP Traps" in Oracle TimesTen In-Memory Database Error Messages and SNMP Traps and "Using SNMP traps for alerts about autorefresh problems" in Oracle TimesTen In-Memory Database Troubleshooting Procedures Guide.

Monitoring AWT cache group operations

You can use the following instructions to monitor the performance of AWT cache groups and determine how much time is spent performing particular tasks in each component of the AWT cache group workflow:

1. Call the ttCacheAWTMonitorConfig built-in procedure as the cache manager user to enable AWT monitoring and specify an AWT workflow sampling factor:

   % ttIsql "DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
   Command> call ttCacheAWTMonitorConfig('ON',16);
   Command> exit

   The replication agent must be running in order to enable monitoring of AWT cache groups.

   For more information about the ttCacheAWTMonitorConfig built-in procedure, see "ttCacheAWTMonitorConfig" in Oracle TimesTen In-Memory Database Reference.

2. Run a ttRepAdmin -showstatus -awtmoninfo utility command to display the monitoring results.

   % ttRepAdmin -showstatus -awtmoninfo cachealone1

   For information about the output of the ttRepAdmin -showstatus -awtmoninfo utility command, see "Monitoring AWT performance" in Oracle TimesTen In-Memory Database Troubleshooting Procedures Guide.

To determine whether asynchronous writethrough operations are keeping up with the rate of updates on cache tables in AWT cache groups, query the LAST_LOG_FILE, REPHOLD_LOG_FILE and REPHOLD_LOG_OFF columns of the SYS.MONITOR system table. If the difference between the value in the LAST_LOG_FILE column and the value in the REPHOLD_LOG_FILE column is increasing over time, and the value in the REPHOLD_LOG_OFF column is increasing slowly or not changing, then the tables are being updated at a faster rate than the updates are being replicated.

Then run a ttRepAdmin -receiver -list utility command and find the row where _ORACLE is in the 'Peer name' field. View the values in the Last Msg Sent, Last Msg Recv, Latency and TPS fields within the same row to determine if the replication activity that is falling behind is asynchronous writethrough operations.

For information on possible causes of poor asynchronous writethrough performance and potential resolutions to the problem, see "Troubleshooting AWT Cache Groups" in Oracle TimesTen In-Memory Database Troubleshooting Procedures Guide.

Configuring a transaction log file threshold for AWT cache groups

The replication agent uses the transaction log to determine which updates on cache tables in AWT cache groups have been propagated to the cached Oracle tables and which updates have not. If updates are not being automatically propagated to Oracle because of a failure, transaction log files accumulate on disk. Examples of a failure that prevents propagation are that the replication agent is not running or the Oracle server is unavailable.
You can call the `ttCacheAWTThresholdSet` built-in procedure as the cache administration user to set a threshold for the number of transaction log files that can accumulate before TimesTen stops tracking updates on cache tables in AWT cache groups. The default threshold is 0. This built-in procedure can only be called if the TimesTen database contains AWT cache groups.

Once the threshold has been exceeded, you need to manually synchronize the cache tables with the cached Oracle tables using an UNLOAD CACHE GROUP statement followed by a LOAD CACHE GROUP statement. TimesTen may purge transaction log files even if they contain updates that have not been propagated to the cached Oracle tables.

Example 7–4 Setting a transaction log file threshold for AWT cache groups

In this example, if the number of transaction log files that contain updates on cache tables in AWT cache groups exceeds 5, TimesTen stops tracking updates and can then purge transaction log files that may contain unpropagated updates:

```
% ttIsql 'DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle'
Command> call ttCacheAWTThresholdSet(5);
```

You can call the `ttCacheAWTThresholdGet` built-in procedure to determine the current transaction log file threshold setting:

```
Command> call ttCacheAWTThresholdGet;
< 5 >
Command> exit
```

Obtaining information about cache grids

You can use the following mechanisms to display information about cache grids and their grid members:

- Call the `ttGridInfo` built-in procedure as the cache manager user to return the grid name, cache administration user name, operating system platform, and TimesTen major release number for a specified or all existing cache grids:

```
% ttIsql 'DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle'
Command> call ttGridInfo('ttGrid');
< TTGRID, CACHEUSER, Linux Intel x86, 32-bit, 11, 2, 1 >
```

For more information about the `ttGridInfo` built-in procedure, see "ttGridInfo" in *Oracle TimesTen In-Memory Database Reference*.

- Call the `ttGridNodeStatus` built-in procedure as the cache manager user to return the grid name, member ID, node number, host name, member name, IP address, and cache agent TCP/IP port number for all members of a specified or all existing cache grids:

```
% ttIsql 'DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle'
Command> call ttGridNodeStatus;
< TTGRID, 1, 1, T, sys1, TTGRID_alone1_1, 140.87.0.201, 5001, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <N...
Suspending global AWT cache group operations

You can use the ttGridGlobalCGSuspend built-in procedure to temporarily block these operations for global AWT cache groups:

- Dynamic loading
- Deleting cache instances

Use the ttGridGlobalCGResume built-in procedure to re-enable these operations.

Tracking DDL statements issued on cached Oracle tables

When a DDL statement is issued on a cached Oracle table, this statement can be tracked in the Oracle TT_version_DLL_L table when the Oracle TT_version_schema-ID_DDL_T trigger is fired to insert a row into the table, where version is an internal TimesTen version number and schema-ID is the ID of user that owns the cached Oracle table. A trigger is created for each Oracle user that owns cached Oracle tables. One DDL tracking table is created to store DDL statements issued on any cached Oracle table. The cache administration user owns the TT_version_DLL_L table and the TT_version_schema-ID_DDL_T trigger.

To enable tracking of DDL statements issued on cached Oracle tables, call the ttCacheDDLTrackingConfig built-in procedure as the cache manager user. By default, DDL statements are not tracked.

For more information about the ttCacheDDLTrackingConfig built-in procedure, see "ttCacheDDLTrackingConfig" in Oracle TimesTen In-Memory Database Reference.

Example 7–5  Enabling tracking of DDL statements issued on cached Oracle tables

```sql
% ttIsql "DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> call ttCacheDDLTrackingConfig('enable');
```

The TT_version_DLL_L table and TT_version_schema-ID_DDL_T trigger are automatically created if the cache administration user has been granted the set of required privileges including RESOURCE and CREATE ANY TRIGGER. These Oracle objects are created when you create a cache group after tracking of DDL statements has been enabled.

If you manually created the Oracle objects used to manage the caching of Oracle data, you need to run the ttIsql utility's cachesqlget command with the ORACLE_DDL_TRACKING option and the INSTALL flag as the cache manager user. This command should be run for each Oracle user that owns cached Oracle tables that you want to track DDL statements on. Running this command generates a SQL*Plus script used to create the TT_version_DLL_L table and TT_version_schema-ID_DDL_T trigger in the Oracle database.

After generating the script, use SQL*Plus to run the script as the sys user.

Example 7–6  Creating DDL tracking table and trigger when Oracle objects were manually created

In this example, the SQL*Plus script generated by the ttIsql utility's cachesqlget command is saved to the /tmp/trackddl.sql file. The owner of the cached Oracle table oratt is passed as an argument to the command.

```sql
% ttIsql "DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> cachesqlget ORACLE_DDL_TRACKING oratt INSTALL /tmp/trackddl.sql;
Command> exit
```
When you need to issue DDL statements such as CREATE, DROP or ALTER on cached Oracle tables in order to make changes to the Oracle schema, drop the affected cache groups before you modify the Oracle schema. Otherwise operations such as automatic refresh may fail. You do not need to drop cache groups if you are altering the Oracle table to add a column. To issue other DDL statements for Oracle tables, first perform the following tasks:

1. Use DROP CACHE GROUP statements to drop all cache groups that cache the affected Oracle tables. If you are dropping an AWT cache group, use the ttRepSubscriberWait built-in procedure to make sure that all committed updates on the cache tables have been propagated to the cached Oracle tables before the cache group is dropped.

   % ttIsql 'DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle'
   Command> call ttRepSubscriberWait('_AWTREPSCHMEM', 'TTREP', '_ORACLE', 'sys1', -1);

2. Stop the cache agent.

3. Make the desired changes to the Oracle schema.

4. Use CREATE CACHE GROUP statements to re-create the cache groups, if feasible.

If you want to truncate an Oracle table that is cached in an automatic refresh cache group, perform the following tasks:

1. Use an ALTER CACHE GROUP statement to set the cache group's automatic refresh state to PAUSED.

2. Truncate the Oracle table.

3. Manually refresh the cache group using a REFRESH CACHE GROUP statement without a WHERE or WITH ID clause.

Automatic refresh operations resume after you refresh the cache group.

You can run the TimesTen_install_dir/oraclescripts/cacheInfo.sql SQL*Plus script as the cache administration user to display information about the Oracle objects used to track DDL statements issued on cached Oracle tables:

   % cd TimesTen_install_dir/oraclescripts
   % sqlplus cacheuser/oracle
   SQL> @cacheInfo
   ****************************
   The information returned for each Oracle user that owns cached Oracle tables includes the name of the DDL tracking table, the name of its corresponding DDL trigger, the name of the user that the DDL trigger is associated with, and the number of cache groups that cache a table owned by the user associated with the DDL trigger.

   If a particular table is cached in more than one grid member, each grid member contributes to the cache group count. An active standby pair counts as one grid member. If a cache group contains more than one cache table, each cache table owned by the user associated with the DDL trigger contributes to the cache group count.
Oracle objects used to manage a caching environment

For an automatic refresh cache group, TimesTen creates a change log table and trigger in the Oracle database for each cache table in the cache group. The trigger is fired for each committed insert, update or delete operation on the cached Oracle table. The trigger records the primary key of the updated rows in the change log table. The cache agent periodically scans the change log table for updated keys and then joins this table with the cached Oracle table to get a snapshot of the latest updates.

The Oracle objects used to process automatic refresh operations can be automatically created by TimesTen as described in "Automatically create Oracle objects used to manage caching of Oracle data" on page 3-9 when you create a cache group with the AUTOREFRESH MODE INCREMENTAL cache group attribute. Alternatively, you can manually create these objects as described in "Manually create Oracle objects used to manage caching of Oracle data" on page 3-10 before performing any cache grid or cache group operation if, for security purposes, you do not want to grant the RESOURCE and CREATE ANY TRIGGER privileges to the cache administration user required to automatically create these objects.

Before the Oracle objects can be automatically or manually created, you must:

- Create a cache administration user in the Oracle database as described in "Create the Oracle users" on page 3-2.
- Set the cache administration user name and password in the TimesTen database as described in "Set the cache administration user name and password" on page 3-14.
- Start the cache agent as described in "Managing the cache agent" on page 3-17.

For each cache administration user, TimesTen creates the following Oracle tables, where version is an internal TimesTen version number and object-ID is the ID of the cached Oracle table:

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TT_version_AGENT_STATUS</td>
<td>Created when the first cache group is created. Stores information about each Oracle table cached in an automatic refresh cache group.</td>
</tr>
<tr>
<td>TT_version_AR_PARAMS</td>
<td>Created when the cache administration user name and password is set. Stores the action to take when the cache administration user's tablespace is full.</td>
</tr>
<tr>
<td>TT_version_CACHE_STATS</td>
<td>Created when the cache administration user name and password is set.</td>
</tr>
<tr>
<td>TT_version_DATABASES</td>
<td>Created when the cache administration user name and password is set. Stores the automatic refresh status for all TimesTen databases that cache data from the Oracle database.</td>
</tr>
<tr>
<td>TT_version_DB_PARAMS</td>
<td>Created when the cache administration user name and password is set. Stores the cache agent timeout, recovery method for dead cache groups, and the cache administration user's tablespace usage threshold.</td>
</tr>
<tr>
<td>TT_version_DDL_L</td>
<td>Created when the cache administration user name and password is set. Tracks DDL statements issued on cached Oracle tables.</td>
</tr>
</tbody>
</table>
For each cache administration user, TimesTen creates the following Oracle triggers, where version is an internal TimesTen version number, object-ID is the ID of the cached Oracle table, and schema-ID is the ID of user who owns the cached Oracle table:

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TT_version_DDL_TRACKING</td>
<td>Created when the cache administration user name and password is set. Stores a flag indicating whether tracking of DDL statements on cached Oracle tables is enabled or disabled.</td>
</tr>
<tr>
<td>TT_version_REPACTIVESTANDBY</td>
<td>Created when the first AWT cache group is created. Tracks the state and roles of TimesTen databases containing cache tables in an AWT cache group that are replicated in an active standby pair replication scheme.</td>
</tr>
<tr>
<td>TT_version_REPPEERS</td>
<td>Created when the first AWT cache group is created. Tracks the time and commit sequence number of the last update on the cache tables that was asynchronously propagated to the cached Oracle tables.</td>
</tr>
<tr>
<td>TT_version_SYNC_OBJ</td>
<td>Created when the first cache group is created.</td>
</tr>
<tr>
<td>TT_version_USER_COUNT</td>
<td>Created when the first cache group is created. Stores information about each cached Oracle table.</td>
</tr>
<tr>
<td>TT_version_object-ID_L</td>
<td>One change log table is created per Oracle table cached in an automatic refresh cache group when the cache group is created. Tracks updates on the cached Oracle table.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trigger Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TT_version_REPACTIVESTANDBY_T</td>
<td>Created when the first AWT cache group is created. When fired, inserts rows into the TT_version_REPACTIVESTANDBY table.</td>
</tr>
<tr>
<td>TT_version_object-ID_T</td>
<td>One trigger is created per Oracle table cached in an automatic refresh cache group when the cache group is created. Fired for each insert, delete or update operation issued on the cached Oracle table to track operations in the TT_version_object-ID_L change log table.</td>
</tr>
<tr>
<td>TT_version_schema-ID_DDL_T</td>
<td>One trigger for each user who owns cached Oracle tables. Created when a cache group is created after tracking of DDL statements has been enabled. Fired for each DDL statement issued on a cached Oracle table to track operations in the TT_version_DDL_L table.</td>
</tr>
</tbody>
</table>

The Oracle objects used to process asynchronous writethrough operations can be automatically created by TimesTen as described in "Automatically create Oracle objects used to manage caching of Oracle data" on page 3-9 when you create an AWT cache group. Alternatively, you can manually create these objects as described in "Manually create Oracle objects used to manage caching of Oracle data" on page 3-10 before performing any cache grid or cache group operation if, for security purposes, you do not want to grant the RESOURCE privilege to the cache administration user required to automatically create these objects.
For the timesten user, TimesTen creates the following Oracle tables:

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TT_GRIDID</td>
<td>Created by running the SQL*Plus script initCacheGlobalSchema.sql. Stores the ID number assigned to the most recently created cache grid.</td>
</tr>
<tr>
<td>TT_GRIDINFO</td>
<td>Created by running the SQL*Plus script initCacheGlobalSchema.sql. Stores the grid name, grid ID, and name of the cache administration user for all existing cache grids.</td>
</tr>
</tbody>
</table>

For each cache administration user, TimesTen creates the following Oracle tables, where version is an internal TimesTen version number and grid-ID is the ID number of the cache grid:

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TT_version_grid-name_grid-IDCGNODEID</td>
<td>One table is created per cache grid when a grid is created. Stores the operating system name and version, and TimesTen release number.</td>
</tr>
<tr>
<td>TT_version_grid-name_grid-IDCGNODEINFO</td>
<td>One table is created per cache grid when a grid is created. Stores the host name, member name, IP address, and cache agent TCP/IP port of all attached grid members.</td>
</tr>
<tr>
<td>TT_version_grid-name_grid-IDCGROUPDEFS</td>
<td>One table is created per cache grid when a grid is created. Stores the cache group name, owner, reference count and SQL text of all global cache groups in standalone TimesTen databases or active standby pairs that are associated with the cache grid.</td>
</tr>
</tbody>
</table>

**Impact of failed automatic refresh operations on TimesTen databases**

A change log table is created in the cache administration user's tablespace for each Oracle table that is cached in an automatic refresh cache group. For each update operation issued on these cached Oracle tables, a row is inserted into their change log table to keep track of updates that need to be applied to the TimesTen cache tables upon the next incremental automatic refresh cycle. TimesTen periodically deletes rows in the change log tables that have been applied to the cache tables.

An Oracle table cannot be cached in more than one cache group within a TimesTen database. However, an Oracle table can be cached in more than one TimesTen database. This results in an Oracle table corresponding to multiple TimesTen cache tables. If updates on cached Oracle tables are not being automatically refreshed into all of their corresponding cache tables because the cache agent is not running on one or more of the TimesTen databases that the Oracle tables are cached in, rows in their change log tables are not deleted by default. The cache agent may not be running on a particular TimesTen database because the agent was explicitly stopped or never started, the database was destroyed, or the installed instance that the database resides in is down. As a result, rows accumulate in the change log tables and degrade the performance of automatic refresh operations on cache tables in TimesTen databases where the cache agent is running. This can also cause the cache administration user's tablespace to fill up.
You can set a cache agent timeout to prevent rows from accumulating in the change log tables and not getting deleted. The following criteria must be met in order for TimesTen to delete rows in the change log tables when the cache agent is not running on a TimesTen database and a cache agent timeout is set:

- Oracle tables are cached in automatic refresh cache groups within more than one TimesTen database
- The cache agent is running on at least one of the TimesTen databases but is not running on at least another database
- Rows in the change log tables have been applied to the cache tables on all TimesTen databases where the cache agent is running
- For those databases where the cache agent is not running, the agent process has been down for a period of time that exceeds the cache agent timeout

Call the `ttCacheConfig` built-in procedure as the cache manager user from any of the TimesTen databases that cache data from the Oracle database. Pass the `AgentTimeout` string to the `Param` parameter and the timeout setting as a numeric string to the `Value` parameter. Do not pass in any values to the `tblOwner` and `tblName` parameters as they are not applicable to setting a cache agent timeout.

**Example 7–7  Setting a cache agent timeout**

In the following example, the cache agent timeout is set to 900 seconds (15 minutes):

```
% ttIsql 'DSN=cachalone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle'
Command> call ttCacheConfig('AgentTimeout',,,,'900');
```

To determine the current cache agent timeout setting, call `ttCacheConfig` passing only the `AgentTimeout` string to the `Param` parameter:

```
Command> call ttCacheConfig('AgentTimeout');
< AgentTimeout, <NULL>, <NULL>, 900 >
```

The default cache agent timeout is 0 seconds which means rows in the change log tables are not deleted until they have been applied to all its cache tables. If you set the cache agent timeout to a value between 1 and 600 seconds, the timeout is set to 600 seconds. The cache agent timeout applies to all TimesTen databases that cache data from the same Oracle database and have the same cache administration user name setting.

When determining a proper cache agent timeout setting, consider the time it takes to load the TimesTen database into memory, the time to start the cache agent process, potential duration of network outages, and anticipated duration of planned maintenance activities.

Each TimesTen database, and all of its automatic refresh cache groups have an automatic refresh status to determine whether any deleted rows from the change log tables were not applied to the cache tables in the cache groups. If rows were deleted from the change log tables and not applied to some cache tables because the cache agent on the database was down for a period of time that exceeded the cache agent timeout, those cache tables are no longer synchronized with the cached Oracle tables. Subsequent updates on the cached Oracle tables are not automatically refreshed into the cache tables until the accompanying cache group is recovered.

The following are the possible automatic refresh statuses for an automatic refresh cache group:
Impact of failed automatic refresh operations on TimesTen databases

- **ok**: All of the deleted rows from the change log tables were applied to its cache tables. Incremental automatic refresh operations continue to occur on the cache group.

- **dead**: Some of the deleted rows from the change log tables were not applied to its cache tables so the cache tables are not synchronized with the cached Oracle tables. Automatic refresh operations have ceased on the cache group and will not resume until the cache group has been recovered.

- **recovering**: The cache group is being recovered. Once recovery completes, the cache tables are synchronized with the cached Oracle tables, the cache group’s automatic refresh status is set to ok, and incremental automatic refresh operations resume on the cache group.

The following are the possible automatic refresh statuses for a TimesTen database:

- **alive**: All of its automatic refresh cache groups have an automatic refresh status of ok
- **dead**: All of its automatic refresh cache groups have an automatic refresh status of dead
- **recovering**: At least one of its automatic refresh cache groups have an automatic refresh status of recovering

If the cache agent on a TimesTen database is down for a period of time that exceeds the cache agent timeout, the automatic refresh status of the database is set to dead. Also, the automatic refresh status of all automatic refresh cache groups within that database are set to dead.

If you have enabled SNMP traps, a trap is thrown when the automatic refresh status of a database is set to dead.

Call the `ttCacheDbCgStatus` built-in procedure as the cache manager user to determine the automatic refresh status of a cache group and its accompanying TimesTen database. Pass the owner of the cache group to the `cgOwner` parameter and the name of the cache group to the `cgName` parameter.

**Example 7–8  Determining the automatic refresh status of a cache group and TimesTen database**

In the following example, the automatic refresh status of the database is alive and the automatic refresh status of the `cacheuser.customer_orders` read-only cache group is ok:

```sql
% ttIsql "DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> call ttCacheDbCgStatus('cacheuser','customer_orders');
< alive, ok >
```

To view only the automatic refresh status of the database and not of a particular cache group, call `ttCacheDbCgStatus` without any parameters:

```sql
Command> call ttCacheDbCgStatus;
< dead, <NULL> >
```

If the automatic refresh status of a cache group is ok, its cache tables are being automatically refreshed based on its automatic refresh interval. If the automatic refresh status of a database is alive, the automatic refresh status of all its automatic refresh cache groups are ok.

If the automatic refresh status of a cache group is dead, its cache tables are no longer being automatically refreshed when updates are committed on the cached Oracle
tables. The cache group must be recovered in order to resynchronize the cache tables with the cached Oracle tables.

You can configure a recovery method for cache groups whose automatic refresh status is dead.

Call the `ttCacheConfig` built-in procedure as the cache manager user from any of the TimesTen databases that cache data from the Oracle database. Pass the `DeadDbRecovery` string to the `Param` parameter and the recovery method as a string to the `Value` parameter. Do not pass in any values to the `tblOwner` and `tblName` parameters as they are not applicable to setting a recovery method for dead cache groups.

The following are the valid recovery methods:

- **Normal**: When the cache agent starts, a full automatic refresh operation is performed on cache groups whose automatic refresh status is dead in order to recover those cache groups. This is the default recovery method.
- **Manual**: For each explicitly loaded cache group whose automatic refresh status is dead, a `REFRESH CACHE GROUP` statement must be issued in order to recover these cache groups after the cache agent starts.
  
  For each dynamic cache group whose automatic refresh status is dead, a `REFRESH CACHE GROUP` or `UNLOAD CACHE GROUP` statement must be issued in order to recover these cache groups after the cache agent starts.
- **None**: Cache groups whose automatic refresh status is dead must be dropped and then re-created after the cache agent starts in order to recover them.

**Example 7–9 Configuring the recovery method for dead cache groups**

In the following example, the recovery method is set to `Manual` for cache groups whose automatic refresh status is dead:

```sql
% ttIsql 'DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle'
Command> call ttCacheConfig('DeadDbRecovery',,,,'Manual');
```

To determine the current recovery method for dead cache groups, call `ttCacheConfig` passing only the `DeadDbRecovery` string to the `Param` parameter:

```sql
Command> call ttCacheConfig('DeadDbRecovery');
< DeadDbRecovery, <NULL>, <NULL>, manual >
```

The recovery method applies to all automatic refresh cache groups in all TimesTen databases that cache data from the same Oracle database and have the same cache administration user name setting.

If you have enabled SNMP traps, a trap is thrown when the cache agent starts and the recovery method is set to `Manual` or `None` to alert you to manually issue a statement such as `REFRESH CACHE GROUP` or `DROP CACHE GROUP` in order to recover cache groups in the database whose automatic refresh status is dead.

When a cache group begins the recovery process, its automatic refresh status is changed from dead to recovering, and the status of the accompanying TimesTen database is changed to recovering, if it is currently dead.

After the cache group has been recovered, its automatic refresh status is changed from `recovering` to `ok`. Once all cache groups have been recovered and their automatic refresh status are `ok`, the status of the accompanying TimesTen database is changed from `recovering` to `alive`. 
A full automatic refresh operation requires more system resources to process than an incremental automatic refresh operation when there is a small volume of updates to refresh and a large number of rows in the cache tables. If you need to bring a TimesTen database down for maintenance activities and the volume of updates anticipated during the downtime on the Oracle tables that are cached in automatic refresh cache groups is small, you can consider temporarily setting the cache agent timeout to 0. When the database is brought back up and the cache agent restarted, incremental automatic refresh operations resumes on cache tables in automatic refresh cache groups. Full automatic refresh operations are avoided because the automatic refresh status on the accompanying cache groups were not changed from ok to dead so those cache groups do not need to go through the recovery process. Make sure to set the cache agent timeout back to its original value once the database is back up and the cache agent has been started.

**Dropping Oracle objects used by automatic refresh cache groups**

If a TimesTen database that contains automatic refresh cache groups becomes unavailable, Oracle objects such as change log tables and triggers used to implement automatic refresh operations continue to exist in the Oracle database. A TimesTen database is unavailable, for example, when the TimesTen system is taken offline or the database has been destroyed without dropping its automatic refresh cache groups.

Oracle objects used to implement automatic refresh operations also continue to exist in the Oracle database when a TimesTen database is no longer being used but still contains automatic refresh cache groups. Rows continue to accumulate in the change log tables. This impacts automatic refresh performance on other TimesTen databases. Therefore, it is desirable to drop these Oracle objects associated with the unavailable or abandoned TimesTen database.

Run the `TimesTen_install_dir/oraclescripts/cacheCleanUp.sql` SQL*Plus script as the cache administration user to drop the Oracle objects used to implement automatic refresh operations. The host name of the TimesTen system and the TimesTen database path name are passed as arguments to the `cacheCleanUp.sql` script. You can run the `cacheInfo.sql` script as the cache administration user to determine the host name of the TimesTen system and the database path name. The `cacheInfo.sql` script can also be used to determine whether any objects used to implement automatic refresh operations exist in the Oracle database.

**Example 7–10 Dropping Oracle objects for automatic refresh cache groups**

In the following example, the TimesTen database still contained one read-only cache group `customer_orders` with cache tables `oratt.customer` and `oratt.orders` when the database was dropped. The `cacheCleanUp.sql` script drops the change log tables and triggers associated with the two cache tables.

```bash
% cd TimesTen_install_dir/oraclescripts
% sqlplus cacheuser/oracle
SQL> @cacheCleanUp 'sys1' '/users/OracleCache/alone1'
```

```
*****************************OUTPUT**************************************
Performing cleanup for object_id: 69959 which belongs to table : CUSTOMER
Executing: delete from tt_05_agent_status where host = sys1 and datastore = /users/OracleCache/alone1 and object_id = 69959
Executing: drop table tt_05_69959_L
Executing: drop trigger tt_05_69959_T
Executing: delete from tt_05_user_count where object_id = object_id1
Performing cleanup for object_id: 69966 which belongs to table : ORDERS
```
Monitoring the cache administration user's tablespace

By default, when the cache administration user's tablespace is full, an error is returned to the application from Oracle when an update operation such as an UPDATE, INSERT or DELETE statement is issued on a particular cached Oracle table.

Rather than return an error to the application when the cache administration user's tablespace is full, you can configure TimesTen to delete existing rows from the change log tables to make space for new rows when an update operation is issued on a particular cached Oracle table. If some of the deleted change log table rows have not been applied to the TimesTen cache tables, a full automatic refresh operation is performed on those cache tables in each TimesTen database that contains the tables upon the next automatic refresh cycle.

Call the ttCacheConfig built-in procedure as the cache manager user from any of the TimesTen databases that cache tables from the Oracle database. Pass the TblSpaceFullRecovery string to the Param parameter, the owner and name of the cached Oracle table to the tblOwner and tblName parameters, respectively, on which you want to configure an action to take if the cache administration user's tablespace becomes full, and the action itself as a string to the Value parameter.

The following are the valid actions:

- **None**: Return an Oracle error to the application when an update operation is issued on the cached Oracle table. This is the default action.
- **Reload**: Delete rows from the change log table and perform a full automatic refresh operation on the cache table upon the next automatic refresh cycle when an update operation is issued on the cached Oracle table.

**Example 7–11 Configuring an action when the cache administration user's tablespace becomes full**

In the following example, rows are deleted from the change log table and a full automatic refresh operation is performed on the cache table upon the next automatic refresh cycle when an update operation is issued on the oratt.customer cached Oracle table while the cache administration user's tablespace is full:

```sql
% ttIsql 'DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle'
Command> call ttCacheConfig('TblSpaceFullRecovery','oratt','customer','Reload');
```

To determine the current action to take when an update operation is issued on a particular cached Oracle table if the cache administration user's tablespace is full, call ttCacheConfig passing only the TblSpaceFullRecovery string to the Param parameter, and the owner and name of the cached Oracle table to the tblOwner and tblName parameters, respectively:

```sql
Command> call ttCacheConfig('TblSpaceFullRecovery','oratt','customer');
< TblSpaceFullRecovery, ORATT, CUSTOMER, reload >
```

The action to take when update operations are issued on a cached Oracle table while the cache administration user's tablespace is full applies to all TimesTen databases that
Monitoring the cache administration user's tablespace

cache tables from the same Oracle database and have the same cache administration user name setting.

If you have enabled SNMP traps, a trap is thrown when an update operation is issued on a cached Oracle table and the cache administration user's tablespace is full.

You can configure TimesTen to return a warning to the application when an update operation such as an UPDATE, INSERT or DELETE statement is issued on cached Oracle tables and causes the usage of the cache administration user's tablespace to exceed a specified threshold.

Call the ttCacheConfig built-in procedure as the cache manager user from any of the TimesTen databases that cache tables from the Oracle database. Pass the TblSpaceThreshold string to the Param parameter and the threshold as a numeric string to the Value parameter. The threshold value represents the percentage of space used in the cache administration user's tablespace upon which a warning is returned to the application when an update operation is issued on a cached Oracle table. Do not pass in any values to the tblOwner and tblName parameters as they are not applicable to setting a warning threshold for the usage of the cache administration user's tablespace.

The cache administration user must be granted the SELECT privilege on the Oracle SYS.DBA_DATA_FILES table in order for the cache manager user to set a warning threshold on the cache administration user's tablespace usage, and for the cache administration user to monitor its tablespace to determine if the configured threshold has been exceeded.

Example 7–12 Setting a cache administration user's tablespace usage warning threshold

The following example configures a warning to be returned to the application that issues an update operation on a cached Oracle table if it results in the usage of the cache administration user's tablespace to exceed 80 percent:

% ttIsql "DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> call ttCacheConfig('TblSpaceThreshold',,,'80');

To determine the current cache administration user's tablespace usage warning threshold, call ttCacheConfig passing only the TblSpaceThreshold string to the Param parameter:

Command> call ttCacheConfig('TblSpaceThreshold');
< TblspaceThreshold, <NULL>, <NULL>, 80 >

The default cache administration user's tablespace usage warning threshold is 0 percent which means that no warning is returned to the application regardless of the tablespace usage. The cache administration user's tablespace usage warning threshold applies to all TimesTen databases that cache tables from the same Oracle database and have the same cache administration user name setting.

If you have enabled SNMP traps, a trap is thrown when the cache administration user's tablespace usage has exceeded the configured threshold.
This chapter describes the various tasks that need to be performed in the TimesTen and Oracle databases to destroy a cache grid and drop cache groups. It includes the following topics:

- Detaching a TimesTen database from a cache grid
- Stopping the replication agent
- Dropping a cache group
- Destroying a cache grid
- Stopping the cache agent
- Destroying the TimesTen databases
- Dropping the Oracle users and objects

**Detaching a TimesTen database from a cache grid**

Call the `ttGridDetach` built-in procedure to detach a grid member from the cache grid that it is attached to. If the grid member is an active standby pair, both the active master and standby master databases must be detached separately. When a grid member has been detached, you can no longer perform operations on its global cache groups or on their cache tables. The grid member also relinquishes ownership of all cache instances that it had owned. The cache agent and replication agent processes cannot be stopped until the database detaches from its cache grid.

From the `cachealone1`, `cachealone2`, `cacheactive` and `cachestandby` databases, call the `ttGridDetach` built-in procedure as the cache manager user to detach the member from the `ttGrid` cache grid. The following example shows the built-in procedure call from the `cachealone1` database:

```
% ttIsql 'DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle'
Command> call ttGridDetach;
```

Use the `ttRepSubscriberWait` built-in procedure to make sure that all committed updates on cache tables in its global cache groups have been propagated to the cached Oracle tables before the TimesTen database is detached from its cache grid.

```
% ttIsql 'DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle'
Command> call ttRepSubscriberWait('_AWTREPSCHME','TTREP','_ORACLE','sys1',-1);
```

Then after the database has been detached from its grid, the replication agent running on the database can be stopped.
You can force detach a grid member that becomes unavailable but is still attached to the grid. A grid member's underlying TimesTen database is unavailable, for example, when the TimesTen system is taken offline or the database has been destroyed. Call the `ttGridDetach` built-in procedure as the cache manager user passing the value 1 to the `force` parameter from any one of the TimesTen databases that are available except from the read-only subscriber databases.

Command> call ttGridDetach('TTGRID_alone2_2',1);

To determine the names of all attached grid members, call the `ttGridNodeStatus` built-in procedure.

You can force detach a set of grid members that become unavailable but are still attached to the grid by calling the `ttGridDetachList` built-in procedure as the cache manager user from any one of the TimesTen databases that are available except from the read-only subscriber databases. Pass the value 1 to the `force` parameter.

Command> call ttGridDetachList('TTGRID_cacheact_3A TTGRID_cachestand_3B',1);

You can detach all of the grid members by calling the `ttGridDetachAll` built-in procedure:

Command> call ttGridDetachAll();

**Stopping the replication agent**

Call the `ttRepStop` built-in procedure to stop the replication agent. This must be done on each TimesTen database of the active standby pair including any read-only subscriber databases, and any standalone TimesTen databases that contain AWT cache groups.

From the `cachealone1`, `cachealone2`, `cacheactive`, `cachestandby` and `rosubscriber` databases, call the `ttRepStop` built-in procedure as the cache manager user to stop the replication agent on the database:

Command> call ttRepStop;

**Dropping a cache group**

Use the DROP CACHE GROUP statement to drop a cache group and its cache tables. Oracle objects used to manage the caching of Oracle data are automatically dropped when you use the DROP CACHE GROUP statement to drop a cache group, or an ALTER CACHE GROUP statement to set the automatic refresh state to OFF for automatic refresh cache groups.

If you issue a DROP CACHE GROUP statement on a cache group that has an automatic refresh operation in progress:

- The automatic refresh operation stops if the `LockWait` DSN attribute setting is greater than 0. The DROP CACHE GROUP statement preempts the automatic refresh operation.
- The automatic refresh operation continues if the `LockWait` attribute setting is 0. The DROP CACHE GROUP statement is blocked until the automatic refresh operation completes or the statement fails with a lock timeout error.

If cache tables are being replicated in an active standby pair and the cache tables are the only elements that are being replicated, the active standby pair must be dropped using a DROP ACTIVE STANDBY PAIR statement before dropping the cache groups.
An active standby pair replication scheme cannot be dropped from a TimesTen database if that database is still attached to a cache grid.

Execute the following statement as the cache manager user on the cacheactive, cachestandby and rosubscriber databases to drop the active standby pair replication scheme:

Command> DROP ACTIVE STANDBY PAIR;
Command> exit

Before you can drop a cache group, you must grant the DROP ANY TABLE privilege to the cache manager user.

Execute the following statement as the instance administrator on the cachealone1, cachealone2, cacheactive and cachestandby databases to grant the DROP ANY TABLE privilege to the cache manager user. The following example shows the SQL statement issued from the cachealone1 database:

% ttIsql cachealone1
Command> GRANT DROP ANY TABLE TO cacheuser;
Command> exit

Use a DROP CACHE GROUP statement to drop the cache groups from the standalone TimesTen databases, and the active master and standby master databases.

Execute the following statement as the cache manager user on the cachealone1, cachealone2, cacheactive and cachestandby databases to drop the subscriber_accounts global cache group. The following example shows the SQL statement issued from the cachealone1 database:

% ttIsql "DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> DROP CACHE GROUP subscriber_accounts;

If you are dropping an AWT cache group, use the ttRepSubscriberWait built-in procedure to make sure that all committed updates on its cache tables have been propagated to the cached Oracle tables before dropping the cache group.

% ttIsql "DSN=cachealone1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> call ttRepSubscriberWait('_AWTREPSCHEM','TTREP','_ORACLE','sys1','-1');

The replication scheme that was created for the AWT cache group to enable committed updates on its cache tables to be asynchronously propagated to the cached Oracle tables is automatically dropped when you drop the cache group.

If a grid member has a global cache group that contains cache instances, the cache group cannot be dropped until all of its cache tables are empty or the member has detached from its cache grid.

Note: If the cache agent is stopped immediately after dropping a cache group, or altering the cache group's automatic refresh state to OFF, the Oracle objects used to manage the caching of Oracle data may not have been dropped. When the cache agent is restarted, it drops the Oracle objects that were created for the dropped or altered cache group.
Destroying a cache grid

Call the `ttGridDestroy` built-in procedure to destroy a cache grid. By default, a cache grid cannot be destroyed if there are existing global cache groups or attached grid members.

From any one of the TimesTen databases, except from the read-only subscriber databases, call the `ttGridDestroy` built-in procedure as the cache manager user to destroy the `ttGrid` cache grid:

Command> call ttGridDestroy('ttGrid');

You can force destroy a cache grid even if a grid member whose TimesTen database becomes unavailable while it contains global cache groups or is attached to the grid. A TimesTen database is unavailable, for example, when the TimesTen system is taken offline or the database has been destroyed. Call the `ttGridDestroy` built-in procedure as the cache manager user passing the value 1 to the `force` parameter from any one of the TimesTen databases except from the read-only subscriber databases.

Command> call ttGridDestroy('ttGrid',1);

A cache grid should be destroyed only if it is no longer needed and there is no intent to attach to it again.

Stopping the cache agent

Call the `ttCacheStop` built-in procedure to stop the cache agent. This must be done on the active master and standby master databases of the active standby pair, and all standalone TimesTen databases.

From the `cachealone1`, `cachealone2`, `cacheactive` and `cachestandby` databases, issue the following built-in procedure call to stop the cache agent on the database:

Command> call ttCacheStop;
Command> exit

The cache agent cannot be stopped if the TimesTen database is still attached to a cache grid.

Destroying the TimesTen databases

If the TimesTen databases are no longer needed, you can use the `ttDestroy` utility to destroy the databases.

The following example shows the `ttDestroy` utility connecting to and then destroying the `cachealone1` database:

% ttDestroy cachealone1

Dropping the Oracle users and objects

Use SQL*Plus as the `sys` user to drop the `timesten` user, the schema user `oratt`, and the cache administration user `cacheuser`, and all objects such as tables and triggers owned by these users. Then drop the TT_CACHE_ADMIN_ROLE role, and the default tablespace `cachetblsp` used by the `timesten` user and the cache administration user including the contents of the tablespace and its data file.

% sqlplus sys as sysdba
Enter password: password
SQL> DROP USER timesten CASCADE;
SQL> DROP USER oratt CASCADE;
SQL> DROP USER cacheuser CASCADE;
SQL> DROP ROLE tt_cache_admin_role;
SQL> DROP TABLESPACE cachetblsp INCLUDING CONTENTS AND DATAFILES;
SQL> exit
Dropping the Oracle users and objects
This chapter describes how to use Oracle In-Memory Database Cache in an Oracle Real Application Clusters (Oracle RAC) environment. It includes the following topics:

- How Oracle In-Memory Database Cache works in an Oracle RAC environment
- Restrictions on using IMDB Cache in an Oracle RAC environment
- Setting up IMDB Cache in an Oracle RAC environment

How Oracle In-Memory Database Cache works in an Oracle RAC environment

Oracle RAC enables multiple Oracle instances to access one Oracle database with shared resources, including all data files, control files, PFILEs and redo log files that reside on cluster-aware shared disks. Oracle RAC handles read/write consistency and load balancing while providing high availability.

Fast Application Notification (FAN) is an Oracle RAC feature that was integrated with Oracle Call Interface (OCI) in Oracle Database 10g Release 2. FAN publishes information about changes in the cluster to applications that subscribe to FAN events. FAN prevents unnecessary operations such as the following:

- Attempts to connect when services are down
- Attempts to finish processing a transaction when the server is down
- Waiting for TCP/IP timeouts

See Oracle Real Application Clusters Administration and Deployment Guide for more information about Oracle RAC and FAN.

IMDB Cache uses OCI integrated with FAN to receive notification of Oracle events. With FAN, IMDB Cache detects connection failures within a minute. Without FAN, it can take several minutes for IMDB Cache to receive notification of an Oracle failure. Without FAN, IMDB Cache detects a connection failure the next time the connection is used or when a TCP/IP timeout occurs. IMDB Cache can recover quickly from Oracle failures without user intervention.

IMDB Cache also uses Transparent Application Failover (TAF), which is a feature of Oracle Net Services that enables you to specify how you want applications to reconnect after a failure. See Oracle Database Net Services Administrator’s Guide for more information about TAF. TAF attempts to reconnect to the Oracle database for four minutes. If this is not successful, the cache agent restarts and attempts to reconnect with the Oracle database every minute.
OCI applications can use one of the following types of Oracle Net failover functionality:

- **None**: No failover functionality is used. This can also be explicitly specified to prevent failover from happening. This is the default failover functionality.
- **Session**: If an application’s connection is lost, a new connection is automatically created for the application. This type of failover does not attempt to recover selects.
- **Select**: This type of failover enables applications that began fetching rows from a cursor before failover to continue fetching rows after failover.

The behavior of IMDB Cache depends on the actions of TAF and how TAF is configured. By default, TAF and FAN callbacks are installed if you are using IMDB Cache in an Oracle RAC environment. If you do not want TAF and FAN capabilities, set the `RACCallback DSN` attribute to 0.

Table 9–1 shows the behaviors of IMDB Cache operations in an Oracle RAC environment with different TAF failover types.

**Table 9–1 Behavior of IMDB Cache operations in an Oracle RAC environment**

<table>
<thead>
<tr>
<th>Operation</th>
<th>TAF Failover Type</th>
<th>Behavior After a Failed Connection on the Oracle Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic refresh</td>
<td>None</td>
<td>The cache agent automatically stops, restarts and waits until a connection can be established on the Oracle database. This behavior is the same as in a non-Oracle RAC environment. No user intervention is needed.</td>
</tr>
</tbody>
</table>
| Automatic refresh | Session            | One of the following occurs:  
  - All failed connections are recovered.  
  - Automatic refresh operations that were in progress are rolled back and retried.  
  - If TAF times out or cannot recover the connection, the cache agent automatically stops, restarts and waits until a connection can be established on the Oracle database.  
  - In all cases, no user intervention is needed. |
| Automatic refresh | Select             | One of the following occurs:  
  - Automatic refresh operations resume from the point of connection failure.  
  - Automatic refresh operations that were in progress are rolled back and retried.  
  - If TAF times out or cannot recover the connection, the cache agent automatically stops, restarts and waits until a connection can be established on the Oracle database.  
  - In all cases, no user intervention is needed. |
| AWT       | None               | The receiver thread of the replication agent for the AWT cache group exits. A new thread is spawned and tries to connect to the Oracle database.  
  - No user intervention is needed. |
Table 9–1 (Cont.) Behavior of IMDB Cache operations in an Oracle RAC environment

<table>
<thead>
<tr>
<th>Operation</th>
<th>TAF Failover Type</th>
<th>Behavior After a Failed Connection on the Oracle Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWT</td>
<td>Session, Select</td>
<td>One of the following occurs:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ If the connection is recovered and there are uncommitted DML operations in the transaction, the transaction is rolled back and then reissued.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ If the connection is recovered and there are no uncommitted DML operations, new operations can be issued without rolling back.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In all cases, no user intervention is needed.</td>
</tr>
<tr>
<td>SWT, propagate, flush, and</td>
<td>None</td>
<td>The application is notified of the connection loss. The cache agent disconnects from the Oracle database and the current transaction is rolled back. All modified session attributes are lost.</td>
</tr>
<tr>
<td>passthrough</td>
<td></td>
<td>During the next passthrough operation, the cache agent tries to reconnect to the Oracle database. This behavior is the same as in a non-Oracle RAC environment.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No user intervention is needed.</td>
</tr>
<tr>
<td>SWT, propagate and flush</td>
<td>Session</td>
<td>One of the following occurs:</td>
</tr>
<tr>
<td>passthrough</td>
<td></td>
<td>■ The connection to the Oracle database is recovered. If there were open cursors, DML or lock operations on the lost connection, an error is returned and the user must rollback the transaction before continuing. Otherwise, the user can continue without rolling back.</td>
</tr>
<tr>
<td></td>
<td>Select</td>
<td>■ If TAF times out or cannot recover the connection, the application is notified of the connection loss. The cache agent disconnects from the Oracle database and the current transaction is rolled back. All modified session attributes are lost.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>During the next passthrough operation, the cache agent tries to reconnect to the Oracle database.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In this case, no user intervention is needed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The connection to the Oracle database is recovered. If there were DML or lock operations on the lost connection, an error is returned and the user must rollback the transaction before continuing. Otherwise, the user can continue without rolling back.</td>
</tr>
<tr>
<td>Passthrough</td>
<td>Select</td>
<td>The application receives a loss of connection error.</td>
</tr>
<tr>
<td></td>
<td>None</td>
<td>One of the following occurs:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ The load or refresh operation succeeds.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ An error is returned stating that a fetch operation on Oracle cannot be executed.</td>
</tr>
</tbody>
</table>
Restrictions on using IMDB Cache in an Oracle RAC environment

IMDB Cache support of Oracle RAC has the following restrictions:

- IMDB Cache behavior is limited to Oracle RAC, FAN and TAF capabilities. For example, if all nodes for a service fail, the service is not restarted. IMDB Cache waits for the user to restart the service.
- TAF does not recover ALTER SESSION operations. The user is responsible for restoring changed session attributes after a failover.
- IMDB Cache uses OCI integrated with FAN. This interface automatically spawns a thread to wait for an Oracle event. This is the only IMDB Cache feature that spawns a thread in a TimesTen direct link application. Adapt your application to account for this thread creation. If you do not want the extra thread, set the RACCallback DSN attribute to 0 so that TAF and FAN are not used.

Setting up IMDB Cache in an Oracle RAC environment

Install Oracle RAC and IMDB Cache. Ensure that the cache administration user has the SELECT ANY DICTIONARY privilege.

There are two TimesTen environment variables that control TAF timeouts:

- TT_ORA_FAILOVER_TIMEOUT: TAF timeout, in minutes, for the application. Applies to SWT cache groups, user managed cache groups that use the PROPAGATE cache table attribute, and the use of the passsthrough feature. The default is 5 minutes.
- TT_AGENT_ORA_FAILOVER_TIMEOUT: TAF timeout, in minutes, for the replication agent. Applies to AWT cache groups. The default is 5 minutes.

Make sure that the TimesTen daemon and cache agent are started. The following Oracle components should also be started:

- Oracle instances
- Oracle listeners
- Oracle service that will be used for Oracle In-Memory Database Cache

Then perform the following tasks:

1. Verify that the RACCallback DSN attribute is set to 1 (default).

Table 9–1 (Cont.) Behavior of IMDB Cache operations in an Oracle RAC environment

<table>
<thead>
<tr>
<th>Operation</th>
<th>TAF Failover Type</th>
<th>Behavior After a Failed Connection on the Oracle Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load and refresh</td>
<td>Select</td>
<td>One of the following occurs:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- If the Oracle cursor is open and the cursor is recovered, or if the Oracle cursor is not open, then the load or refresh operation succeeds.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- An error is returned if TAF was unable to recover either the session or open Oracle cursors.</td>
</tr>
</tbody>
</table>

Note: An error is less likely to be returned than if the TAF failover type is Session.
2. Use the `DBMS_SERVICE.MODIFY_SERVICE` function or Oracle Enterprise Manager to enable publishing of FAN events. This changes the value in the `AQ_HA_NOTIFICATIONS` column of the Oracle ALL_SERVICES view to YES.

See Oracle Database PL/SQL Packages and Types Reference for more information about the `DBMS_SERVICE` Oracle PL/SQL package.

3. Enable TAF on the Oracle service used for IMDB Cache with one of the following methods:
   - Create a service for TimesTen in the Oracle `tnsnames.ora` file with the following settings:
     - `LOAD_BALANCE=ON` (optional)
     - `FAILOVER_MODE=(TYPE=SELECT) or FAILOVER_MODE=(TYPE=SESSION)`
   - Use the `DBMS_SERVICE.MODIFY_SERVICE` function to set the TAF failover type.

See Oracle Database Net Services Administrator's Guide for more information about enabling TAF.

4. If you have a TimesTen direct link application, link it with a thread library so that it will receive FAN notifications. FAN spawns a thread to monitor for failures.
Using Oracle In-Memory Database Cache with Data Guard

This chapter describes how to configure Oracle In-Memory Database Cache to work with either Synchronous Local Data Guard or Data Guard used during planned maintenance. It includes the following topics:

- Components of MAA for Oracle In-Memory Database Cache
- How Oracle In-Memory Database Cache works with Data Guard

Components of MAA for Oracle In-Memory Database Cache

Oracle Maximum Availability Architecture (MAA) is Oracle’s best practices blueprint based on proven Oracle high availability (HA) technologies and recommendations. The goal of MAA is to achieve the optimal high availability architecture at the lowest cost and complexity.

To be compliant with MAA, Oracle In-Memory Database Cache must support Oracle Real Application Clusters (Oracle RAC) and Oracle Data Guard, as well as have its own HA capability. Oracle In-Memory Database Cache provides its own HA capability through active standby pair replication of cache tables in read-only and AWT cache groups. See "Using Oracle In-Memory Database Cache in an Oracle RAC Environment" on page 9-1 for more information on Oracle In-Memory Database Cache and Oracle RAC.

Oracle Data Guard provides the management, monitoring, and automation software infrastructure to create and maintain one or more synchronized standby databases to protect data from failures, disasters, errors and corruptions. If the primary database becomes unavailable because of a planned or an unplanned outage, Data Guard can switch any standby database to the primary role, thus minimizing downtime and preventing any data loss. For more information about Data Guard, see Oracle Data Guard Concepts and Administration.

The MAA framework for Oracle In-Memory Database Cache supports cache tables in explicitly loaded read-only and AWT cache groups. For cache tables in dynamic cache groups of any cache group type, SWT cache groups, and user managed cache groups that use the AUTOREFRESH cache group attribute, Oracle In-Memory Database Cache cannot access the Oracle database during a failover and switchover because cache applications will have to wait until the failover and switchover completes.

In general, however, all cache groups types are supported with Synchronous Local Data Guard or Data Guard during planned maintenance.
How Oracle In-Memory Database Cache works with Data Guard

Oracle In-Memory Database Cache works with Synchronous Physical Standby failover and switchover and Logical Standby switchover as long as the object IDs for cached Oracle tables remain the same on the primary and standby databases. Object IDs can change if the table is dropped and re-created, altered, or a truncated flashback operation or online segment shrink is executed.

During a transient upgrade, a physical standby database is transformed into a logical standby database. For the time that the standby database is logical, the user must ensure that the object IDs of the cached Oracle tables do not change. Specifically, tables that are cached should not drop and re-created, truncated, altered, flashed back or online segment shrunk.

Configuring the Oracle databases

In order for Oracle In-Memory Database Cache to failover and switchover properly, configure the Oracle primary and standby databases using the following steps:

1. The Data Guard configuration must be managed by the Data Guard Broker so that Oracle In-Memory Database Cache daemon processes and application clients respond faster to failover and switchover events.

2. If you are configuring an Oracle RAC database, use the Oracle Enterprise Manager Cluster Managed Database Services Page to create database services that Oracle In-Memory Database Cache and its client applications use to connect to the Oracle primary database. See "Introduction to Automatic Workload Management" in Oracle Real Application Clusters Administration and Deployment Guide for information on creating database services.

3. If you created the database service in step 2, use the MODIFY_SERVICE function of the DBMS_SERVICE PL/SQL package to modify the service to enable high availability notification to be sent through Advanced Queuing (AQ) by setting the aq_ha_notifications attribute to TRUE. To configure server side TAF settings, set the FAILOVER attributes, as shown in the following example:

   BEGIN
   DBMS_SERVICE.MODIFY_SERVICE
   (service_name => 'DBSERV',
    goal => DBMS_SERVICE.GOAL_NONE,
    dtp => false,
    aq_ha_notifications => true,
    failover_method => 'BASIC',
    failover_type => 'SELECT',
    failover_retries => 180,
    failover_delay => 1);
   END;

4. If you did not create the database service in step 2, use the CREATE_SERVICE function of the DBMS_SERVICE PL/SQL package to create the database service, enable high availability notification, and configure server side TAF settings:

   BEGIN
   DBMS_SERVICE.CREATE_SERVICE
   (service_name => 'DBSERV',
    network_name => 'DBSERV',
    goal => DBMS_SERVICE.GOAL_NONE,
    dtp => false,
    aq_ha_notifications => true,
    failover_method => 'BASIC',
    failover_type => 'SELECT',
    failover_retries => 180,
    failover_delay => 1);
5. Create two triggers to relocate the database service to a Data Guard standby database (Oracle RAC or non-Oracle RAC) after it has switched to the primary role. The first trigger fires on the system start event and starts up the DBSERV service:

```
cREATE OR REPLACE TRIGGER manage_service
AFTER STARTUP ON DATABASE
DECLARE
  role VARCHAR(30);
BEGIN
  SELECT database_role INTO role FROM v$database;
  IF role = 'PRIMARY' THEN
    dbms_service.start_service('DBSERV');
  END IF;
END;
```

The second trigger fires when the standby database remains open during a failover and switchover upon a database role change. It relocates the DBSERV service from the old primary to the new primary database and disconnects any connections to that service on the old primary database so that Oracle In-Memory Database Cache and its client applications can reconnect to the new primary database:

```
cREATE OR REPLACE TRIGGER relocate_service
AFTER DB_ROLE_CHANGE ON DATABASE
DECLARE
  role VARCHAR(30);
BEGIN
  SELECT database_role INTO role FROM v$database;
  IF role = 'PRIMARY' THEN
    dbms_service.start_service('DBSERV');
  ELSE
    dbms_service.stop_service('DBSERV');
    dbms_lock.sleep(2);
    FOR x IN (SELECT s.sid, s.serial#
                FROM v$session s, v$process p
                WHERE s.service_name='DBSERV' AND s.paddr=p.addr)
    LOOP
      BEGIN
        EXECUTE IMMEDIATE 'ALTER SYSTEM DISCONNECT SESSION '' || x.sid || ',
          || x.serial# || ' IMMEDIATE';
        EXCEPTION WHEN OTHERS THEN
        BEGIN
          DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);
        END;
      END LOOP;
  END IF;
END;
```

6. As an option, to reduce the performance impact to Oracle In-Memory Database Cache applications and minimize the downtime during a physical or logical standby database switchover, run the following procedure right before initiating the Data Guard switchover to a physical or logical standby database:

```
failover_type => 'SELECT',
failover_retries => 180,
failover_delay => 1);
```
DECLARE
    role varchar(30);
BEGIN
    SELECT database_role INTO role FROM v$database;
    IF role = 'PRIMARY' THEN
        dbms_service.stop_service('DBSERV');
        dbms_lock.sleep(2);
        FOR x IN (SELECT s.sid, s.serial#
                    FROM v$session s, v$process p
                    WHERE s.service_name='DBSERV' AND s.paddr=p.addr)
            LOOP
                BEGIN
                    EXECUTE IMMEDIATE 'ALTER SYSTEM DISCONNECT SESSION ''' || x.sid || ',
                    || x.serial# || ''' IMMEDIATE';
                    EXCEPTION WHEN OTHERS THEN
                        BEGIN
                            DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);
                        END;
                END;
            END LOOP;
    ELSE
        dbms_service.start_service('DBSERV');
    END IF;
END;

This procedure should be executed first on the physical or logical standby
database, and then on the primary database, right before the switchover process.
Before executing the procedure for a physical standby database switchover, Active
Data Guard must be enabled on the physical standby database.

Before performing a switchover to a logical standby database, stop the Oracle service
for TimesTen on the primary database and disconnect all sessions connected to that
service. Then start the service on the standby database.

At this point, the cache applications try to reconnect to the standby database. If a
switchover occurs, there is no wait required to migrate the connections from the
primary database to the standby database. This eliminates the performance impact on
Oracle In-Memory Database Cache and its applications.

See the Maximum Availability Architecture, Oracle Best Practices for High Availability
white paper for more information.

Configuring the TimesTen database

Configure TimesTen to receive notification of FAN HA events and to avoid
reconnecting to a failed Oracle instance. Use the Oracle Client shipped with Oracle
In-Memory Database Cache.

1. Create an Oracle Net service name that includes all primary and standby hosts in
ADDRESS_LIST. For example:

   DBSERV =
   (DESCRIPTION =
    (ADDRESS_LIST =
     (ADDRESS = (PROTOCOL = TCP)(HOST = PRIMARYDB)(PORT = 1521))
     (ADDRESS = (PROTOCOL = TCP)(HOST = STANDBYDB)(PORT = 1521))
     (LOAD_BALANCE = yes)
    )
    (CONNECT_DATA= (SERVICE_NAME=DBSERV))
)
2. In the client's sqlnet.ora file, set the SQLNET.OUTBOUND_CONNECT_TIMEOUT parameter to enable clients to quickly traverse an address list in the event of a failure. For example, if a client attempts to connect to a host that is unavailable, the connection attempt is bounded to the time specified by the SQLNET.OUTBOUND_CONNECT_TIMEOUT parameter, after which the client attempts to connect to the next host in the address list. Connection attempts continue for each host in the address list until a connection is made.

Setting the SQLNET.OUTBOUND_CONNECT_TIMEOUT parameter to a value of 3 seconds suffices in most environments. For example, add the following entry to the sqlnet.ora file:

```
SQLNET.OUTBOUND_CONNECT_TIMEOUT=3
```
Compatibility Between TimesTen and Oracle

This chapter lists compatibility issues between TimesTen and Oracle. The list is not complete, but it indicates areas that require special attention. It includes the following topics:

- Summary of compatibility issues
- Transaction semantics
- API compatibility
- SQL compatibility
- Mappings between Oracle and TimesTen data types

Summary of compatibility issues

Consider the following differences between TimesTen and Oracle:

- TimesTen and Oracle database metadata are stored differently. See "API compatibility" on page 11-2 for more information.
- TimesTen and Oracle have different transaction isolation models. See "Transaction semantics" on page 11-1 for more information.
- TimesTen and Oracle have different connection and statement properties. For example, TimesTen does not support catalog names, scrollable cursors or updateable cursors.
- Sequences are not cached and synchronized between the TimesTen database and the corresponding Oracle database. See "SQL expressions" on page 11-5 for more information.
- Side effects of Oracle triggers and stored procedures are not reflected in the TimesTen database until after an automatic or manual refresh operation.

Transaction semantics

TimesTen and Oracle transaction semantics differ as follows:

- Oracle serializable transactions can fail at commit time because the transaction cannot be serialized. TimesTen uses locking to enforce serializability.
- Oracle users can lock tables explicitly through SQL. This locking feature is not supported in TimesTen.
- Oracle supports savepoints while TimesTen does not.
In Oracle, a transaction can be set to be read-only or read/write. This is not supported in TimesTen.

For more information about TimesTen isolation levels and transaction semantics, see "Transaction Management and Recovery" in Oracle TimesTen In-Memory Database Operations Guide.

API compatibility

For a complete list of the JDBC API classes and interfaces that TimesTen supports with notes on which methods have a compatibility issue, see "Key JDBC classes and interfaces" in Oracle TimesTen In-Memory Database Java Developer’s Guide.

For a complete list of the ODBC API functions that TimesTen supports with notes on which functions have a compatibility issue, see "TimesTen ODBC Functions and Options" in Oracle TimesTen In-Memory Database C Developer’s Guide.

For a complete list of the OCI functions for Oracle database, release 11.1.0.7, that TimesTen supports with notes on which functions have a compatibility issue, see "TimesTen Support for Oracle Call Interface" in Oracle TimesTen In-Memory Database C Developer’s Guide.

SQL compatibility

This section compares TimesTen’s SQL implementation with Oracle’s SQL. The purpose is to provide users with a list of Oracle SQL features not supported in TimesTen or supported with different semantics.

Schema objects

TimesTen does not recognize some of the schema objects that are supported in Oracle. TimesTen returns a syntax error when a statement manipulates or uses these objects and passes the statement to Oracle. The unsupported objects are:

Access Control objects:
- Roles
- Profiles
- Contexts

Storage Management features:
- Clusters
- Tablespaces
- Rollback segments
- CREATE/DROP DATABASE statements
- Database links
- Directories
- Partitions
- Extended Features
- External procedure libraries
- Object tables
- Object types
- Object views

TimesTen supports views and materialized views, but it cannot cache an Oracle view.
Differences between Oracle and TimesTen tables

The Oracle table features that TimesTen does not support are:

■ ON DELETE SET NULL
■ Check constraints

Data type support

The following Oracle data types are not supported by TimesTen:

TIMESTAMP WITH TIME ZONE
TIMESTAMP WITH LOCAL TIME ZONE
INTERVAL YEAR TO MONTH
INTERVAL DAY TO SECOND
UROWID
CLOB
NCLOB
BLOB
BFILE
REF
"Any" types
XML types
Spatial types
Media types

---

Note: TimesTen can cache CLOB, NCLOB and BLOB data. See “Caching Oracle LOB data” on page 4-29.

Passthrough queries that reference Oracle CLOB, NCLOB and BLOB fields are supported.

---

The following TimesTen data types are not supported by Oracle:

TT_CHAR
TT_VARCHAR
TT_NCHAR
TT_NVARCHAR
TT_BINARY
TT_VARBINARY
TINYINT and TT_TINYINT
TT_SMALLINT
TT_INTEGER
TT_BIGINT
TT_DECIMAL
TT_DATE
TIME and TT_TIME
TT_TIMESTAMP
SQL compatibility

---

**Note:** TimesTen NCHAR/NVARCHAR2 data types are encoded as UTF-16. Oracle NCHAR/NVARCHAR2 data types are encoded as either UTF-16 or UTF-8.

To cache an Oracle NCHAR or NVARCHAR2 column, the Oracle NLS_NCHAR_CHARACTERSET encoding must be AL16UTF16, not AL32UTF8.

---

**SQL operators**

TimesTen supports these operators and predicates that are supported by Oracle:

- **Unary**
  - `-`
  - `+`, `-`, `*`, `/`
  - `=`, `<`, `>`, `<=`, `>=`, `<>`
  - `|`
  - `IS NULL`, `IS NOT NULL`
  - `LIKE` (Oracle LIKE operator ignores trailing spaces, but TimesTen does not)
  - `BETWEEN`
  - `IN`
  - `NOT IN` (list)
  - `AND`
  - `OR`
  - `+` (outer join)
  - `ANY`, `SOME`
  - `ALL` (list)
  - `EXISTS`
  - `UNION`
  - `MINUS`
  - `INTERSECT`

To perform a bitwise AND operation of two bit vector expressions, TimesTen uses the ampersand character (`&`) between the expressions while Oracle uses the `BITAND` function with the expressions as arguments.

---

**SQL functions**

TimesTen supports these functions that are supported by Oracle:

- `ABS`
- `ADD_MONTHS`
- `AVG`
- `CEIL`
- `COALESCE`
- `CONCAT`
- `COUNT`
- `DECODE`
- `EXTRACT`
- `FLOOR`
- `GREATEST`
- `INSTR`
- `LEAST`
- `LENGTH`
- `LPAD`
- `LTRIM`
- `MAX`
MIN
MOD
NUMTOYMINTERVAL
NUMTODSINTERVAL
NVL
POWER
ROUND
RPAD
RTRIM
SIGN
SQRT
SUBSTR
SUM
SYSDATE
TO_DATE
TO_CHAR
TO_NUMBER
TRIM
TRUNC

TimesTen and the Oracle Database interpret the literal \N' \UNNNN' differently. In TimesTen, \N' \unnnn' (where nnnn is a number) is interpreted as the national character set character with the code nnnn. In the Oracle Database, \N' \unnnn' is interpreted as 6 literal characters. The \u is not treated as an escape. This difference causes unexpected behavior. For example, loading a cache group with a WHERE clause that contains a literal can fail. This can also affects dynamic loading and cache grid operation. Applications should use the UNISTR SQL function instead of literals.

SQL expressions

TimesTen supports these expressions that are supported by Oracle:

Column Reference
Sequence
NULL

() Binding parameters
CASE expression
CAST

These TimesTen expressions are not supported by Oracle:

CURRENT_USER
GETDATE
ORA_SYSDATE
SESSION_USER
SYSTEM_USER
TT_HASH
TT_SYSDATE

SQL subqueries

TimesTen supports these subqueries that are supported by Oracle:

IN (subquery)
>,<,= ANY (subquery)
>,=,<= SOME (subquery)
SQL compatibility

- EXISTS (subquery)
- >,=,< (scalar subquery)
- In WHERE clause of DELETE/UPDATE
- In FROM clause

Note: A nonverifiable scalar subquery is a scalar subquery whose 'single-row-result-set' property cannot be determined until execution time. TimesTen allows at most one nonverifiable scalar subquery in the entire query and the subquery cannot be specified in an OR expression.

SQL queries

TimesTen supports these queries that are supported by Oracle:

- FOR UPDATE
- ORDER BY
- GROUP BY
- Table alias
- Column alias

Oracle supports flashback queries, which are queries against a database that is in some previous state (for example, a query on a table as of yesterday). TimesTen does not support flashback queries.

INSERT/DELETE/UPDATE statements

TimesTen supports these DML statements that are supported by Oracle:

- INSERT INTO ... VALUES
- INSERT INTO ... SELECT
- UPDATE WHERE expression (expression may contain a subquery)
- DELETE WHERE expression (expression may contain a subquery)
- MERGE

TimesTen-only SQL and built-in procedures

This section lists TimesTen SQL statements and built-in procedures that are not supported by Oracle. With PassThrough=3, these statements are passed to Oracle for execution and an error is generated.

- All TimesTen cache group DDL and DML statements, including CREATE CACHE GROUP, DROP CACHE GROUP, ALTER CACHE GROUP, LOAD CACHE GROUP, UNLOAD CACHE GROUP, REFRESH CACHE GROUP and FLUSH CACHE GROUP.
- All TimesTen replication management DDL statements, including CREATE REPLICATION, DROP REPLICATION, ALTER REPLICATION, CREATE ACTIVE STANDBY PAIR, ALTER ACTIVE STANDBY PAIR and DROP ACTIVE STANDBY PAIR.
- FIRST n clause
All TimesTen built-in procedures. See "Built-In Procedures" in Oracle TimesTen
In-Memory Database Reference.

PL/SQL constructs

TimesTen supports a subset of stored procedure constructs, functions, data types, packages and package bodies that are supported by Oracle. See Oracle TimesTen
In-Memory Database PL/SQL Developer’s Guide for details.

Mappings between Oracle and TimesTen data types

When you choose data types for columns in the TimesTen cache tables, consider the data types of the columns in the Oracle tables and choose an equivalent or compatible data type for the columns in the cache tables.

Primary and foreign key columns are distinguished from non-key columns. The data type mappings allowed for key columns in a cache table are shown in Table 11–1.

Table 11–1  Data type mappings allowed for key columns

<table>
<thead>
<tr>
<th>Oracle data type</th>
<th>TimesTen data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER((p, s))</td>
<td>NUMBER((p, s))</td>
</tr>
<tr>
<td></td>
<td>Note: DECIMAL((p, s)) or NUMERIC((p, s)) can also be used. They are aliases for NUMBER((p, s)).</td>
</tr>
<tr>
<td>NUMBER((p, 0))</td>
<td>TT_TINYINT</td>
</tr>
<tr>
<td>INTEGER</td>
<td>TT_SMALLINT</td>
</tr>
<tr>
<td></td>
<td>TT_INTEGER</td>
</tr>
<tr>
<td></td>
<td>TT_BIGINT</td>
</tr>
<tr>
<td></td>
<td>NUMBER((p, 0))</td>
</tr>
<tr>
<td>NUMBER</td>
<td>TT_TINYINT</td>
</tr>
<tr>
<td></td>
<td>TT_SMALLINT</td>
</tr>
<tr>
<td></td>
<td>TT_INTEGER</td>
</tr>
<tr>
<td></td>
<td>TT_BIGINT</td>
</tr>
<tr>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>CHAR((m))</td>
<td>CHAR((m))</td>
</tr>
<tr>
<td>VARCHAR2((m))</td>
<td>VARCHAR2((m))</td>
</tr>
<tr>
<td>RAW((m))</td>
<td>VARBINARY((m))</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
</tr>
<tr>
<td>TIMESTAMP((m))</td>
<td>TIMESTAMP((m))</td>
</tr>
<tr>
<td>NCHAR((m))</td>
<td>NCHAR((m))</td>
</tr>
<tr>
<td>NVARCHAR2((m))</td>
<td>NVARCHAR2((m))</td>
</tr>
</tbody>
</table>

Table 11–2 shows the data type mappings allowed for non-key columns in a cache table.
<table>
<thead>
<tr>
<th>Oracle data type</th>
<th>TimesTen data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER(p,s)</td>
<td>NUMBER(p,s)</td>
</tr>
<tr>
<td></td>
<td>REAL</td>
</tr>
<tr>
<td></td>
<td>FLOAT</td>
</tr>
<tr>
<td></td>
<td>BINARY_FLOAT</td>
</tr>
<tr>
<td></td>
<td>DOUBLE</td>
</tr>
<tr>
<td></td>
<td>BINARY_DOUBLE</td>
</tr>
<tr>
<td>NUMBER(p,0)</td>
<td>TT_TINYINT</td>
</tr>
<tr>
<td>INTEGER</td>
<td>TT_SMALLINT</td>
</tr>
<tr>
<td></td>
<td>TT_INTEGER</td>
</tr>
<tr>
<td></td>
<td>TT_BIGINT</td>
</tr>
<tr>
<td></td>
<td>NUMBER(p,0)</td>
</tr>
<tr>
<td></td>
<td>FLOAT</td>
</tr>
<tr>
<td></td>
<td>BINARY_FLOAT</td>
</tr>
<tr>
<td></td>
<td>DOUBLE</td>
</tr>
<tr>
<td></td>
<td>BINARY_DOUBLE</td>
</tr>
<tr>
<td>NUMBER</td>
<td>TT_TINYINT</td>
</tr>
<tr>
<td></td>
<td>TT_SMALLINT</td>
</tr>
<tr>
<td></td>
<td>TT_INTEGER</td>
</tr>
<tr>
<td></td>
<td>TT_BIGINT</td>
</tr>
<tr>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td></td>
<td>REAL</td>
</tr>
<tr>
<td></td>
<td>FLOAT</td>
</tr>
<tr>
<td></td>
<td>BINARY_FLOAT</td>
</tr>
<tr>
<td></td>
<td>DOUBLE</td>
</tr>
<tr>
<td></td>
<td>BINARY_DOUBLE</td>
</tr>
<tr>
<td>CHAR(m)</td>
<td>CHAR(m)</td>
</tr>
<tr>
<td>VARCHAR2(m)</td>
<td>VARCHAR2(m)</td>
</tr>
<tr>
<td>RAW(m)</td>
<td>VARBINARY(m)</td>
</tr>
<tr>
<td>LONG</td>
<td>VARCHAR2(m)</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> m can be any valid value within the range defined for the VARCHAR2 data type.</td>
</tr>
<tr>
<td>LONG RAW</td>
<td>VARBINARY(m)</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> m can be any valid value within the range defined for the VARBINARY data type.</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
</tr>
<tr>
<td>TIMESTAMP(m)</td>
<td>TIMESTAMP(m)</td>
</tr>
</tbody>
</table>
Table 11–2  (Cont.) Data type mappings allowed for non-key columns

<table>
<thead>
<tr>
<th>Oracle data type</th>
<th>TimesTen data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLOAT(n)</td>
<td>FLOAT(n)</td>
</tr>
<tr>
<td><strong>Note:</strong> Includes DOUBLE and FLOAT, which are equivalent to FLOAT(126). Also includes REAL, which is equivalent to FLOAT(63).</td>
<td></td>
</tr>
<tr>
<td>BINARY_FLOAT</td>
<td>BINARY_FLOAT</td>
</tr>
<tr>
<td>BINARY_DOUBLE</td>
<td>BINARY_DOUBLE</td>
</tr>
<tr>
<td>NCHAR(m)</td>
<td>NCHAR(m)</td>
</tr>
<tr>
<td>NVARCHAR2(m)</td>
<td>NVARCHAR2(m)</td>
</tr>
<tr>
<td>CLOB</td>
<td>VARCHAR2(m)</td>
</tr>
<tr>
<td><strong>Note:</strong> 1&lt;=m&lt;=4 megabytes</td>
<td></td>
</tr>
<tr>
<td>BLOB</td>
<td>VARBINARY(m)</td>
</tr>
<tr>
<td><strong>Note:</strong> 1&lt;=m&lt;=4 megabytes</td>
<td></td>
</tr>
<tr>
<td>NCLOB</td>
<td>NVARCHAR2(m)</td>
</tr>
<tr>
<td><strong>Note:</strong> 1&lt;=m&lt;=2,097,152 characters</td>
<td></td>
</tr>
</tbody>
</table>
This chapter lists the SQL*Plus scripts that are installed with Oracle In-Memory Database Cache used to perform various configuration, administrative and monitoring tasks, and provides links to more information including examples. All scripts are installed in the TimesTen_install_dir/oraclescripts directory.

### Installed SQL*Plus scripts

- **cacheCleanUp.sql**: Drops Oracle objects such as change log tables and triggers used to implement automatic refresh operations. Script is used when a TimesTen database containing automatic refresh cache groups is unavailable because the TimesTen system is offline, or the database was destroyed without dropping its automatic refresh cache groups. Run this script as the cache administration user. See "Dropping Oracle objects used by automatic refresh cache groups" on page 7-16 for more information.

- **cacheInfo.sql**: Returns change log table information for all Oracle tables cached in an automatic refresh cache group, and information about Oracle objects used to track DDL statements issued on cached Oracle tables. Script is used to monitor automatic refresh operations on cache groups and DDL statements issued on cached Oracle tables. Run this script as the cache administration user. See "Monitoring automatic refresh operations on cache groups" on page 7-4 and "Tracking DDL statements issued on cached Oracle tables" on page 7-8 for more information.

- **grantCacheAdminPrivileges.sql**: Grants privileges to the cache administration user that are required to automatically create Oracle objects used to manage the caching of Oracle data when particular cache grid and cache group operations are performed. Run this script as the sys user. See "Automatically create Oracle objects used to manage caching of Oracle data" on page 3-9 for more information.

- **initCacheAdminSchema.sql**: Grants a minimal set of privileges to the cache administration user and manually creates Oracle objects used to manage the caching of Oracle data. Run this script as the sys user. See "Manually create Oracle objects used to manage caching of Oracle data" on page 3-10 for more information.

- **initCacheGlobalSchema.sql**: Creates the Oracle timessten user, the Oracle tables owned by the timessten user to store information about cache grids, and the TT_CACHE_ADMIN_ROLE role that defines privileges on these Oracle tables. Script must be run regardless of whether you are automatically or manually
creating Oracle objects used to manage caching of Oracle data. Run this script as
the sys user. See "Create the Oracle users" on page 3-2 for more information.

- `initCacheGridSchema.sql`: Manually creates Oracle tables used to store
information about TimesTen databases that are associated with a particular cache
grid. Run this script as the sys user. See "Manually create Oracle objects used to
manage caching of Oracle data" on page 3-10 for more information.

- README: Contains descriptions of the SQL*Plus scripts that are installed with
Oracle In-Memory Database Cache.
aging
Delete cache instances from the cache tables of a cache group after a specified period of time (time-based) or when a specified level of database usage is reached (LRU).

asynchronous writethrough (AWT) cache group
A cache group in which committed updates on TimesTen cache tables are automatically and asynchronously propagated to the cached Oracle tables. The commit on the TimesTen database occurs asynchronously from the commit on the Oracle database.

automatic refresh
Committed updates on cached Oracle tables are automatically refreshed to the TimesTen cache tables.

automatic refresh cache group
A read-only cache group or a user managed cache group that uses the AUTOREFRESH MODE INCREMENTAL cache group attribute.

bidirectional transmit
Propagate committed updates on TimesTen cache tables to the cached Oracle tables, and refresh committed updates on cached Oracle tables to the TimesTen cache tables.

cache administration user
Oracle user that creates and maintains Oracle objects that store information used to manage cache grids and enforce predefined behaviors of particular cache group types.

cache agent
A TimesTen process that processes cache group operations, such as automatic refresh, loading a cache group, and passing through statements to the Oracle database for execution.

cache group
Defines the data from the Oracle tables to cache in a TimesTen database. A cache group can cache all or a subset of a single Oracle table or a set of related Oracle tables. If multiple Oracle tables are cached, each cache table (except for the root table) must reference another cache table in the cache group through foreign key constraints.
**cache grid**
A set of distributed grid members consisting of TimesTen in-memory databases that work together to cache data from a single Oracle database and guarantee cache coherence among the TimesTen databases.

**cache group primary key**
The primary key of the cache group’s root table.

**cache instance**
A specific row of data identified by the primary key in the cache group’s root table. If there are multiple tables in the cache group, the cache instance consists of the set of rows in the child tables associated by foreign key relationships with the row in the root table.

**cache manager user**
TimesTen user that performs cache grid and cache group operations such as creating and configuring a cache grid, and creating cache groups.

**cache table**
The root table or a child table in a cache group.

**cache table users**
TimesTen users who own cache tables.

**child table**
A cache table that has a foreign key reference to either the primary key of the root table or another child table that either directly or indirectly references the root table. The table hierarchy in your cache group can designate child tables to be parents of other child tables. No cache table can be a child to more than one parent in the cache group.

**dynamic cache group**
A cache group category where data in its cache tables can be loaded on demand, manually loaded or automatically loaded.

**explicitly loaded cache group**
A cache group category where data in its cache tables are manually or automatically loaded.

**flush**
To manually propagate committed inserts or updates on TimesTen cache tables in a user managed cache group to the cached Oracle tables.

**grid member**
A component of a cache grid consisting of either a standalone TimesTen database or an active standby pair.

**grid node**
A TimesTen database of a grid member that is either a standalone database, or the active master database or standby master database of an active standby pair.

**global cache group**
A cache group classification where data in its cache tables are shared across multiple TimesTen databases within a cache grid.
**load**
Copy new cache instances from the cached Oracle tables to the TimesTen cache tables. Cache instances that are already exist in the cache tables are not updated or deleted.

**local cache group**
A cache group classification where data in its cache tables are not shared across multiple TimesTen databases even if the databases are members of the same cache grid.

**Oracle schema users**
Oracle users who own Oracle tables to be cached in a TimesTen database.

**Oracle timesten user**
Oracle user who owns Oracle tables that store information about cache grids.

**propagate**
Transmit committed updates on TimesTen cache tables to the cached Oracle tables.

**read-only cache group**
A cache group in which committed updates on cached Oracle tables are automatically refreshed to the TimesTen cache tables. You cannot update cache tables directly in a read-only cache group.

**refresh**
For an explicitly loaded cache group, unload and then load the cache group.
For a dynamic cache group, replace existing cache instances in the cache tables with the most current data from the cached Oracle tables.

**replication**
The process of maintaining duplicate copies of data in multiple databases.

**replication agent**
Replication at each master and subscriber TimesTen database is controlled by a replication agent process. The replication agent on the master database reads the transaction log records and transmits any committed updates on replicated elements to the replication agent on the subscriber database. The replication agent on the subscriber database then applies the updates to its database.
For an AWT cache group, the replication agent transmits committed updates on its cache tables to the cached Oracle tables.

**root table**
The parent table in the cache group that does not reference any other table in the cache group through a foreign key constraint. The primary key of the root table is the primary key of the cache group.

**synchronous writethrough (SWT) cache group**
A cache group in which committed updates on TimesTen cache tables are automatically and synchronously propagated to the cached Oracle tables. When an application commits a transaction, it is committed on Oracle before it is committed on TimesTen.
system managed cache group
System managed cache groups enforce predefined behaviors. The types of system managed cache groups are read-only, synchronous writethrough and asynchronous writethrough.

TT_CACHE_ADMIN_ROLE role
Role granted to the cache administration user that defines privileges on the Oracle tables owned by the timesten user which store information about cache grids.

user managed cache group
A cache group that implements customized behavior such as bidirectional transmit.

unload
Delete cache instances from a cache table. The rows in the cached Oracle tables are not affected.
A

active standby pair
  active master database, 6-3
  create, 6-4
  definition, 6-3
  read-only subscriber database, 6-6
  standby master database, 6-5

aging policy, 4-30
  LRU aging, 4-31
  time-based aging, 4-32

asynchronous writethrough (AWT) cache group
  create, 2-8, 4-10
  definition, 1-5, 4-9
  monitor, 7-6
  restrictions, 4-12

automatic refresh
  definition, 1-5
  example, 2-11

automatic refresh cache group
  change log table, 7-12
  definition, 3-3
  drop Oracle objects to manage cache, 7-16
  load and refresh, 5-4
  manually create Oracle objects, 4-24
  monitor, 7-4
  recovery method, 7-15
  status, 7-13

automatic refresh interval, 4-22
automatic refresh mode, 4-21
automatic refresh state, 4-22

automatically create Oracle objects to manage
  cache, 3-9

AUTOREFRESH cache group attribute, 4-21
autorefresh now, 4-22, 5-6

B

BLOB data
  caching in TimesTen, 4-29

built-in procedures
  ttAgingLRUCfg, 4-31
  ttAgingScheduleNow, 4-34
  ttCacheAutorefreshStatsGet, 7-4
  ttCacheAWTMonitorCfg, 7-6
  ttCacheAWTThresholdSet, 7-7
  ttCacheCfg built-in procedure with
    DeadDbRecovery parameter, 7-15
  ttCacheCfg built-in procedure with
    TblSpaceFullRecovery parameter, 7-17
  ttCacheCfg built-in procedure with
    TblSpaceThreshold parameter, 7-18
  ttCacheCfg with AgentTimeout parameter, 7-13
  ttCacheDbCgStatus, 7-14
  ttCacheDDLTrackingCfg, 7-8
  ttCachePolicySet, 3-19
  ttCacheStart, 2-7, 3-17
  ttCacheStop, 2-15, 3-18, 8-4
  ttCacheUidGet, 3-17
  ttCacheUidPwdSet, 2-4, 3-14
  ttGridAttach, 2-10, 4-39
  ttGridCreate, 2-5, 3-16
  ttGridDestroy, 2-14, 8-4
  ttGridDetach, 2-14, 8-1
  ttGridDetachList, 8-2
  ttGridInfo, 7-7
  ttGridNameSet, 2-5, 3-17
  ttGridNodeStatus, 7-7
  ttRepPolicySet, 4-11
  ttRepStart, 2-9, 4-10
  ttRepStateSet, 6-4
  ttRepStop, 2-14, 4-10, 8-2
  ttRepSubscriberWait, 7-9, 8-1, 8-3

C

cache administration user
  create, 2-2, 3-3
  definition, 2-2, 3-3
  determine, 3-17
  set in TimesTen database, 2-4, 3-14

cache administration user default tablespace
  create, 2-2, 3-3
  drop, 2-15, 8-5
  monitor, 7-17
  recover when full, 7-17
  usage warning threshold, 7-18

cache agent
  reconnecting with Oracle Database, 9-1
  start, 2-7, 3-17
  status, 7-1
stop, 2-15, 3-18, 8-4
timeout, 7-13
cache agent start policy
definition, 3-18
set, 3-19
cache grid
associate TimesTen database, 2-5, 3-17
attach a TimesTen database, 2-10, 4-39
create, 2-5, 3-16
definition, 1-1, 3-15
destroy, 2-14, 8-4
detach a set of TimesTen databases, 8-2
detach a TimesTen database, 2-14, 8-1
figure, 1-2
Oracle system parameters, 3-15
ten or more nodes, 3-15
cache group
create, 2-5
definition, 1-3, 4-1	table hierarchy, 4-3
cache instance
definition, 1-4
cache manager user
create, 2-4, 3-13
defined, 3-13
definition, 2-3, 3-13
minimum privileges, 2-4, 3-14
cache table attributes
ON DELETE CASCADE, 4-27
PROPAGATE, 4-20
READONLY, 4-21
UNIQUE HASH ON, 4-28
cache table user
defined, 3-13
cache table users
create, 2-4, 3-13
definition, 2-4, 3-13
cacheCleanUp.sql SQL*Plus script, 7-16, 12-1
CacheGridEnable DSN attribute, 3-16
CacheGridMsgWait DSN attribute, 4-39
cachelnfo.sql SQL*Plus script, 7-4, 12-1
child table
definition, 1-3, 4-2
CLOB data
caching in TimesTen, 4-29
connection failures
Oracle Database, 9-1
connectivity between TimesTen and Oracle databases
test, 3-17
CREATE ACTIVE STANDBY PAIR statement, 6-4
DROP ACTIVE STANDBY PAIR statement, 8-2
DROP CACHE GROUP statement
eample, 2-14, 8-3
DSN attributes
CacheGridEnable, 3-16
CacheGridMsgWait, 4-39
DatabaseCharacterSet, 3-12
DynamicLoadEnable, 5-12
DynamicLoadErrorMode, 5-13
LockLevel, 3-12
OracleNetServiceName, 3-12
OraclePWD, 3-12
PassThrough, 3-12, 5-15
PermSize, 3-12
PWD, 3-12
RACCallback, 9-4
TypeMode, 3-12
UID, 3-12
DSN for TimesTen database
eample, 2-3, 3-12
dynamic cache group
create, 4-36
definition, 1-6, 4-36
dynamic load
definition, 5-10
disable, 5-12
display errors, 5-13
eample, 2-12, 5-10
DynamicLoadEnable DSN attribute, 5-12
DynamicLoadErrorMode DSN attribute, 5-13
evironment variables
Microsoft Windows, 3-2
UNIX, 3-1
eitically loaded cache group
definition, 1-6
Fast Application Notification (FAN), 9-1
flush a cache group
definition, 1-5
FLUSH CACHE GROUP statement
definition, 5-14
eample, 5-14
global cache group
create, 4-38
definition, 1-7, 4-37
sharing data among grid members, 6-7
global cache groups
ownership changes, 7-8
global query, 6-8
global unload operation, 5-15
GlobalProcessing optimizer hint
query, 6-8
Data Guard, 10-1
data type mapping
key columns, 11-7
non-key columns, 11-8
DatabaseCharacterSet DSN attribute, 3-12
DDL statements on cached Oracle tables
track, 7-8
GlobalProcessing optimizer hint
query, 6-8
Index-2
unload operation, 5-15
grantCacheAdminPrivileges.sql SQL*Plus script, 2-2, 3-10, 12-1
grid member
definition, 1-1, 3-15
grid members
detaching all, 8-2
grid node
definition, 1-1, 3-15

H
high availability
active standby pair, 1-8
cache grid, 1-8
Data Guard, 1-8
Oracle Real Application Clusters (Oracle RAC), 1-8

I
initCacheAdminSchema.sql SQL*Plus script, 3-11, 12-1
initCacheGlobalSchema.sql SQL*Plus script, 2-2, 3-3, 12-1
initCacheGridSchema.sql SQL*Plus script, 3-11, 12-2
instance administrator, 2-4

L
load a cache group
definition, 1-5
LOAD CACHE GROUP statement
definition, 5-2
eexample, 2-10, 5-3
PARALLEL clause, 5-7
WITH ID clause, 5-5
LOB data
cache administration user privileges, 3-8
caching in TimesTen, 4-29
restrictions on caching, 4-30
local cache group
definition, 1-7
LockLevel DSN attribute, 3-12
LRU aging policy, 4-31

M
manually create Oracle objects to manage cache, 3-10
Maximum Availability Architecture (MAA), 10-1
multiple-table cache group, 4-2

N
NCLOB data
caching in TimesTen, 4-29

O
OCIAttrGet() OCI function

OCI_ATTR_ROW_COUNT option, 5-15
ON DELETE CASCADE cache table attribute, 4-27
Oracle Database connection failures, 9-1
Oracle database character set
determine, 2-3, 3-12
Oracle objects to manage cache automatically create, 3-9
determine, 3-11
manually create, 3-10
tables and triggers, 7-10
Oracle Real Application Clusters (Oracle RAC), 9-1
Oracle schema users create, 2-2, 3-3
definition, 2-1, 3-3
Oracle Server releases supported, 3-1
Oracle SQL*Plus scripts
cacheCleanUp.sql, 7-16, 12-1
cacheInfo.sql, 7-4, 12-1
grantCacheAdminPrivileges.sql, 2-2, 3-10, 12-1
initCacheAdminSchema.sql, 3-11, 12-1
initCacheGlobalSchema.sql, 2-2, 3-3, 12-1
initCacheGridSchema.sql, 3-11, 12-2
Oracle synonyms
cache, 4-29
Oracle timesten user
definition, 2-1
Oracle users
cache administration user, 2-2
drop, 2-15, 8-5
privileges, 3-3
schema users, 2-1
timesten user, 2-1, 3-2
OracleNetServiceName DSN attribute, 3-12
OraclePWD DSN attribute, 3-12

P
PassThrough DSN attribute, 3-12, 5-15
PermSize DSN attribute, 3-12
privileges
Oracle users, 3-3
propagate
element, 2-13
propagate cache instances
definition, 1-5
PROPAGATE cache table attribute, 4-20
PWD DSN attribute, 3-12

Q
query
cache grid, 6-8
global, 6-8

R
RACCallback DSN attribute, 9-4
read-only cache group
create, 2-8, 4-7
ttRepPolicySet built-in procedure, 4-11
ttRepStart built-in procedure, 2-9, 4-10
ttRepStateSet built-in procedure, 6-4
ttRepStop built-in procedure, 2-14, 4-10, 8-2
ttRepSubscriberWait built-in procedure, 7-9, 8-1, 8-3
ttStatus utility, 7-1
ttTraceMon utility
  AUTOREFRESH component, 7-5
TypeMode DSN attribute, 3-12

U
UID DSN attribute, 3-12
UNIQUE HASH ON cache table attribute, 4-28
unload cache group
  global, 5-15
UNLOAD CACHE GROUP statement
  definition, 5-14
  example, 5-14
user managed cache group
  bidirectional transmit, 4-15
  create, 4-16, 4-18
  definition, 1-5, 4-15

W
WHERE clause, 4-24
  referencing Oracle PL/SQL functions, 4-27