TimesTen
Cache Connect to Oracle Guide

Release 7.0

B31685-03

ORACLE
TIMESTEN

Copyright ©1996, 2007, Oracle. All rights reserved.

ALL SOFTWARE AND DOCUMENTATION (WHETHER IN HARD COPY OR ELECTRONIC FORM) ENCLOSED AND ON THE COMPACT DISC(S) ARE SUBJECT TO THE LICENSE AGREEMENT.

The documentation stored on the compact disc(s) may be printed by licensee for licensee's internal use only. Except for the foregoing, no part of this documentation (whether in hard copy or electronic form) may be reproduced or transmitted in any form by any means, electronic or mechanical, including photocopying, recording, or any information storage and retrieval system, without the prior written permission of TimesTen Inc.

Oracle, JD Edwards, PeopleSoft, Retek, TimesTen, the TimesTen icon, MicroLogging and Direct Data Access are trademarks or registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

September 2007

Printed in the United States of America

Contents

About this Guide

TimesTen documentation 1
Background reading 2
Conventions used in this guide 3
Technical Support 5

1 Cache Connect to Oracle Concepts

About cache groups. 8
Exchanging data between the TimesTen cache and Oracle 10
Cache group types 11

2 Quick Start

Setting up TimesTen and Oracle. 14
Install the Oracle Client on the TimesTen host. 15
Create required Oracle accounts 15
Create an account on TimesTen 16
Create a TimesTen DSN 16
Creating a READONLY cache group. 18
Step 1: Create an Oracle table 19
Step 2: Create the cache group 20
Step 3: Load the cache group 21
Step 4: Update the Oracle table. 22
Step 5: Drop the cache group 22
Step 6: Stop the cache agent 22
Enabling the SQL passthrough feature 23
Step 1: Create a new TimesTen DSN 24
Step 2: Create a READONLY cache group 25
Step 3: Load the cache group 25
Step 4: Update the cache group table 26
Step 5: Drop the cache group 27
Step 6: Stop the cache agent 27
Checklist for creating a cache group 28

3 Defining a Cache Group

Creating a cache group definition 31
Naming a cache group 32
Selecting a cache group type. 32
READONLY cache groups. 33

Restrictions on using READONLY cache groups 34
Example: Creating a READONLY cache group 34
SYNCHRONOUS WRITETHROUGH (SWT) cache groups 35
Restrictions on using SWT cache groups 36
Example: Creating an SWT cache group 36
ASYNCHRONOUS WRITETHROUGH (AWT) cache groups 38
What an AWT cache group guarantees 39
What an AWT cache does not guarantee 39
Restrictions on using AWT cache groups 40
Example: Creating an AWT cache group 41
Creating Oracle objects for AWT 41
Setting up an AWT cache group 41
USERMANAGED cache groups 43
Examples: Creating USERMANAGED cache groups 43
Defining cache group and table attributes 47
AUTOREFRESH cache group attribute 48
Specifying the AUTOREFRESH cache group attribute 48
Restrictions on using AUTOREFRESH 49
Cache table attributes 50
PROPAGATE 50
READONLY 52
ON DELETE CASCADE 52
UNIQUE HASH ON 53
Defining cache group tables 53
Defining a single cache group table 53
Defining multiple cache group tables 55
Caching Oracle partitioned tables 59
About Oracle synonyms 60
Using WHERE clauses 60
Locale neutrality in WHERE clauses 62
Implementing aging in a cache group 64
Usage-based aging 64
Time-based aging 66
Aging and foreign keys 67
Scheduling when aging starts 67
Configuring a sliding window 68
Working with data types and type modes 69
Floating point data types 69
Data type mappings for Cache Connect to Oracle 70

4 Managing Cache Groups

Configuring your system for cache groups 73
Configuring Cache Connect to Oracle on a UNIX platform 75
Configuring Cache Connect to Oracle on a Windows platform 75
Set-up tasks on the Oracle database 76
Create Oracle users and set privileges 76
Create a separate tablespace for the cache administration user 80
When the cache administration user tablespace gets full 80
Defining DSNs for cached tables 81
Starting and stopping the cache agent 82
Controlling the cache agent from the command line 83
Set the cache administration user ID and password from the command line 83
Start the cache agent from the command line 84
Stop the cache agent from the command line 85
Set the cache agent start policy from the command line 85
Controlling the cache agent from a program 85
Set the cache administration user ID and password from a program . . 86
Start the cache agent from a program 86
Stop the cache agent from a program 87
Set the cache agent start policy from a program 87
Checking the status of the cache agent 88
Starting the replication agent for an AWT cache group 90
Applying a cache group definition to a data store 91
Setting a passthrough level 91
Passthrough and parameter binding 93
Changing the passthrough level 93
Monitoring cache groups 94
Using the ttIsql cachegroup command 94
Monitoring autorefresh cache groups 95
Managing Oracle objects for READONLY, AUTOREFRESH, and AWT cache groups 95
Automatically installing Oracle objects 98
Manually installing Oracle objects 99
Manually installing Oracle objects for AWT cache groups 100
Confirming installation of Oracle objects 102
Removing Oracle objects 102
Manually removing Oracle objects for AWT cache groups 102

5 Using Cache Groups

Altering a cache group 106
Dropping a cache group 107

Copying data between the Oracle database and cache groups 108
Loading and refreshing a cache group 109
Loading and refreshing AUTOREFRESH and READONLY cache groups 110
Loading and refreshing a cache group WITH ID 110
Loading and refreshing cache groups with multiple tables 111
Improving performance of loading and refreshing large tables 112
Using transparent loading 113
Types of SELECT statements 113
Configuring transparent loading 114
Overriding transparent loading for a transaction 114
Flushing a USERMANAGED cache group 115
Unloading a cache group. 116
Replicating cache group tables 117
Changing the Oracle schema 118

6 Cache Administrator

About the Cache Administrator 119
Security 119
Starting the Cache Administrator 120
Connecting to a data store 121
Using the Cache Administrator 122

7 Implementing Cache Connect in a RAC Environment

How Cache Connect works in a RAC environment 125
Restrictions on using Cache Connect in a RAC environment 129
Setting up Cache Connect in a RAC environment 129

8 Compatibility Between TimesTen and Oracle

Summary of compatibility issues 131
Cache Connect support for Oracle client/server releases 132
Transaction semantics 132
JDBC compatibility 132
java.sql.Connection 133
java.sql.Statement 133
java.sql.ResultSet 134
java.sql.PreparedStatement 134
java.sql.CallableStatement 135
java.sql.ResultSetMetadata 135
Stream support 136
ODBC compatibility 136

SQL compatibility 139
Schema objects 139
Differences between Oracle and TimesTen tables 140
Data type support 140
SQL operators 141
SQL functions 141
SQL expressions 142
SQL subqueries 142
SQL queries 143
INSERT/DELETE/UPDATE statements 143
TimesTen-only SQL and built-in procedures 143
Other SQL issues 144

Glossary

Index

About this Guide

Oracle TimesTen In-Memory Database is a high-performance, in-memory database that supports the ODBC and JDBC interfaces.

This guide is for application developers who use and administer TimesTen and for system administrators who configure and manage Cache Connect to Oracle. A Cache Connect to Oracle data store is used to cache Oracle data in TimesTen.

To work with this guide, you should understand how database systems work. You should also have knowledge of SQL (Structured Query Language) and either ODBC (Open Database Connectivity) or JDBC (Java Database Connectivity). See "Background reading" on page 2 if you are not familiar with these interfaces.

TimesTen documentation

TimesTen documentation is available on the product distribution media and on the Oracle Technology Network: http://www.oracle.com/technology/documentation/timesten_doc.html.

Including this guide, the TimesTen documentation set consists of these documents:

Book Titles	Description
Oracle TimesTen In-Memory Database Installation Guide	Contains information needed to install and configure TimesTen on all supported platforms.
Oracle TimesTen In-Memory Database Introduction	Describes all the available features in the Oracle TimesTen In-Memory Database.
Oracle TimesTen In-Memory Database Operations Guide	Provides information on configuring TimesTen and using the ttIsql utility to manage a data store. This guide also provides a basic tutorial for TimesTen.
Oracle TimesTen In-Memory Database C Developer's and Reference Guide and the *Oracle TimesTen In-Memory Database Java Developer's and Reference Guide*	Provide information on how to use the full set of available features in TimesTen to develop and implement applications that use TimesTen.
Oracle TimesTen In-Memory Database API Reference Guide	Describes all TimesTen utilities, procedures, APIs and provides a reference to other features of TimesTen.

Oracle TimesTen In-Memory Database SQL Reference Guide	Contains a complete reference to all TimesTen SQL statements, expressions and functions, including TimesTen SQL extensions.
Oracle TimesTen In-Memory Database Error Messages and SNMP Traps	Contains a complete reference to the TimesTen error messages and information on using SNMP Traps with TimesTen.
Oracle TimesTen In-Memory Database TTClasses Guide	Describes how to use the TTClasses C++ API to use the features available in TimesTen to develop and implement applications.
TimesTen to TimesTen Replication Guide	Provides information to help you understand how TimesTen Replication works and step-by-step instructions and examples that show how to perform the most commonly needed tasks. This guide is for application developers who use and administer TimesTen and for system administrators who configure and manage TimesTen Replication.
TimesTen Cache Connect to Oracle Guide	Describes how to use Cache Connect to cache Oracle data in TimesTen data stores. This guide is for developers who use and administer TimesTen for caching Oracle data.
Oracle TimesTen In-Memory Database Troubleshooting Procedures Guide	Provides information and solutions for handling problems that may arise while developing applications that work with TimesTen, or while configuring or managing TimesTen.

Background reading

For a Java reference, see:

- Horstmann, Cay and Gary Cornell. *Core Java(TM) 2, Volume I--Fundamentals (7th Edition) (Core Java 2)*. Prentice Hall PTR; 7 edition (August 17, 2004).

A list of books about ODBC and SQL is in the Microsoft ODBC manual included in your developer's kit. Your developer's kit includes the appropriate ODBC manual for your platform:

WINDOWS

- *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide* provides all relevant information on ODBC for Windows developers.

UNIX

- *Microsoft ODBC 2.0 Programmer's Reference and SDK Guide*, included online in PDF format, provides information on ODBC for UNIX developers.

For a conceptual overview and programming how-to of ODBC, see:

- Kyle Geiger. *Inside ODBC*. Redmond, WA: Microsoft Press. 1995.

For a review of SQL, see:

- Melton, Jim and Simon, Alan R. *Understanding the New SQL: A Complete Guide*. San Francisco, CA: Morgan Kaufmann Publishers. 1993.
- Groff, James R. / Weinberg, Paul N. *SQL: The Complete Reference, Second Edition.* McGraw-Hill Osborne Media. 2002.

For information about Unicode, see:

- The Unicode Consortium, *The Unicode Standard, Version 5.0*, Addison-Wesley Professional, 2006.
- The Unicode Consortium Home Page at http://www.unicode.org

Conventions used in this guide

TimesTen supports multiple platforms. Unless otherwise indicated, the information in this guide applies to all supported platforms. The term Windows refers to Windows 2000, Windows XP and Windows Server 2003. The term UNIX refers to Solaris, Linux, HP-UX, Tru64 and AIX.

TimesTen documentation uses these typographical conventions:

If you see...	It means...
`code font`	Code examples, filenames, and pathnames. For example, the `.odbc.ini.` or `ttconnect.ini` file.
`italic code font`	A variable in a code example that you must replace. For example: `Driver=`*`install_dir`*`/lib/libtten.sl` Replace *`install_dir`* with the path of your TimesTen installation directory.

TimesTen documentation uses these conventions in command line examples and descriptions:

If you see...	It means...
`fixed width italics`	Variable; must be replaced with an appropriate value. In some cases, such as for parameter values in built-in procedures, you may need to single quote (' ') the value.
`[ ]`	Square brackets indicate that an item in a command line is optional.

`{ }`	Curly braces indicated that you must choose one of the items separated by a vertical bar (|) in a command line.
`|`	A vertical bar (or pipe) separates arguments that you may use more than one argument on a single command line.
`...`	An ellipsis (. . .) after an argument indicates that you may use more than one argument on a single command line.
`%`	The percent sign indicates the UNIX shell prompt.
`#`	The number (or pound) sign indicates the UNIX root prompt.

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

If you see...	**It means...**
`install_dir`	The path that represents the directory where the current release of TimesTen is installed.
`TTinstance`	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. The instance name “giraffe” is used in examples in this guide.
`bits` or *`bb`*	Two digits, either 32 or 64, that represent either the 32-bit or 64-bit operating system.
`release` or *`rr`*	Two digits that represent the first two digits of the current TimesTen release number, with or without a dot. For example, 51 or 7.0 represents TimesTen Release 7.0.
`jdk_version`	Two digits that represent the version number of the major JDK release. Specifically, 14 represent JDK 1.4; 5 represents JDK 5.
`timesten`	A sample name for the TimesTen instance administrator. You can use any legal user name as the TimesTen administrator. On Windows, the TimesTen instance administrator must be a member of the Administrators group. Each TimesTen instance can have a unique instance administrator name.
`DSN`	The data source name.

Technical Support

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

http://www.oracle.com/support/contact.html

1

Cache Connect to Oracle Concepts

TimesTen data exchange technology provides the connection and bidirectional transfer of data between the Oracle database and the TimesTen data manager. Data exchange technology also facilitates the capture and processing of high-volume event flows into the TimesTen data manager and subsequent transfer of data into the Oracle database.

TimesTen allows you to cache Oracle data by creating a *cache group* in TimesTen that maps to a single Oracle table or a group of related Oracle tables. The combined features of TimesTen that allow for caching Oracle data are referred to as **Cache Connect to Oracle**.

This chapter includes the following topics:

- About cache groups
- Exchanging data between the TimesTen cache and Oracle
- Cache group types

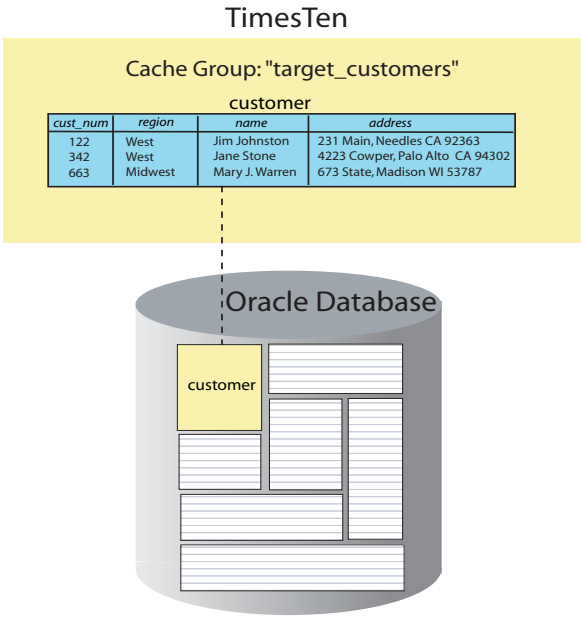
About cache groups

A *cache group* describes the Oracle data to be cached in a TimesTen data store. A cache group can be created to cache all or part of a single Oracle table or a set of related Oracle tables.

A cache group can be created by using the CREATE CACHE GROUP SQL statement described in "Creating a cache group definition" on page 31 or by using the browser-based Cache Administrator described in Chapter 6, "Cache Administrator."

Figure 1.1 shows a cache group named *target_customers* that caches the Oracle table *customer*. The data in the cache group is a subset of the much larger data set for all the customers stored in the Oracle database.

Figure 1.1 Cache Group with One Table



You can cache multiple Oracle tables in the same cache group if you define a *root table* and one or more *child tables*. There can only be one root table in a cache group.

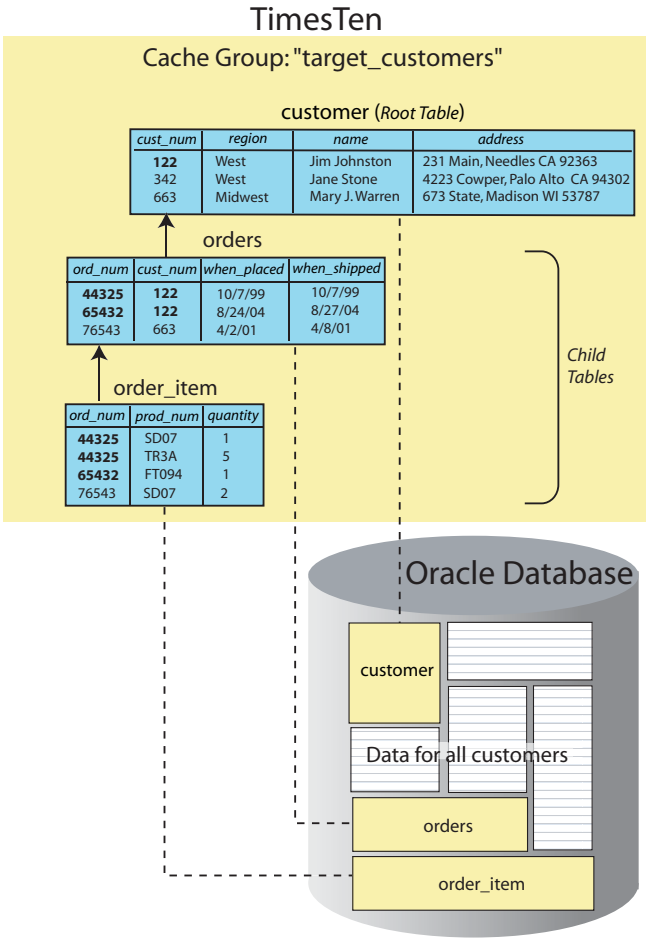
In a cache group with multiple tables, each child table in the cache group must be joined to the root table or another child table in the cache group by a foreign key constraint. Though the tables in a cache group must be joined by foreign key constraints in TimesTen, the tables do not necessarily need to be joined in the Oracle instance. The root table in the cache group does not refer to any other table in the cache group through a foreign key constraint. All other tables in the

cache group are *child tables*. For more information about cache groups with multiple tables, see "Defining multiple cache group tables" on page 55.

The basic unit loaded from Oracle into a TimesTen cache group is a *cache instance*, which describes the set of rows associated by foreign key relationships with a particular row in the cache group root table. Highlighted in each table in the cache group is the cache instance identified by primary key *122* in the root table. This *cache instance key* identifies the row in the root table and all of the related rows in the child tables that reference this row.

Figure 1.2 shows the tables in the *target_customers* cache group. The root table is *customer*. *orders* and *order_item* are child tables.

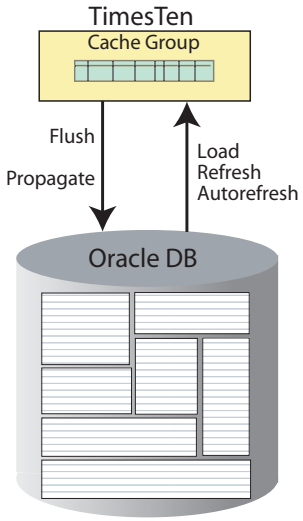
Figure 1.2 Cache Group with Multiple Tables



Exchanging data between the TimesTen cache and Oracle

As shown in Figure 1.3, to *flush* or *propagate* means to copy table data from the TimesTen cache to Oracle. To *load* or *refresh* means to copy data from Oracle to the TimesTen cache.

Figure 1.3 Exchanging data between TimesTen and Oracle



How your cache group propagates data to Oracle and refreshes data from Oracle depends on the type of cache group you select.

Cache group types

The basic types of cache groups are:

- READONLY cache groups.

  A READONLY cache group enforces a caching behavior in which updates on Oracle tables are applied to TimesTen through the AUTOREFRESH mechanism. See "AUTOREFRESH cache group attribute" on page 48.

- SYNCHRONOUS WRITETHROUGH (SWT) cache groups.

  A SYNCHRONOUS WRITETHROUGH (SWT) cache group enforces a caching behavior in which cached data is updated in TimesTen and propagated to Oracle. Updates to a SWT cache group are committed synchronously.

- ASYNCHRONOUS WRITETHROUGH (AWT) cache groups.

  An ASYNCHROUNOUS WRITETHROUGH (AWT) cache group enforces caching behavior in which cached data is updated in TimesTen and propagated to Oracle. Updates to an AWT cache group are committed asynchronously.

- USERMANAGED cache group.

  A USERMANAGED cache group can be customized. For example, the tables in a USERMANAGED cache group can have the READONLY or PROPAGATE attribute.

See Chapter 3, "Defining a Cache Group" for a description of each cache group type.

2

Quick Start

This chapter describes how to create and manage a simple cache group using the SQL statements described in *Oracle TimesTen In-Memory Database SQL Reference Guide*. You can also create a cache group using the browser-based Cache Administrator described in Chapter 6, "Cache Administrator."

This chapter includes the following topics:

- Setting up TimesTen and Oracle
- Creating a READONLY cache group
- Enabling the SQL passthrough feature
- Checklist for creating a cache group

Note: If TimesTen was installed with Access Control enabled, you must have either ADMIN privileges to the TimesTen data store or be the owner of the data store to complete the procedures in this section. See "Access Control" in *Oracle TimesTen In-Memory Database Installation Guide* for details.

Setting up TimesTen and Oracle

Before you can create a cache group, you must set up the TimesTen and Oracle environments. Complete the following tasks:

1. Install the Oracle Client on the TimesTen host.
2. Create required Oracle accounts.
3. Create an account on TimesTen.
4. Create a TimesTen DSN.

If you intend to use Cache Connect to Oracle in an Oracle Real Application Clusters (RAC) environment, see Chapter 7, “Implementing Cache Connect in a RAC Environment.”

Install the Oracle Client on the TimesTen host

The Cache Connect to Oracle feature uses Oracle shared libraries to communicate with the Oracle database. You can install these libraries by installing Oracle Client on the same machine you install TimesTen.

Install the Oracle 9*i* Client or Oracle Database 10*g* client. The Oracle client release does not have to be the same as the Oracle server release. The following Oracle client and server releases are supported by Cache Connect:

- Oracle 10g Release 2 (Oracle 10.2.0.1.0 or above)
- Oracle 10g Release 1 (Oracle 10.1.0.5.0 or above)
- Oracle 9i Release 2 (Oracle 9.2.0.8.0 or above)

See "Cache Connect support for Oracle client/server releases" on page 132.

On HP-UX, Solaris, and Linux, you need to configure the ORACLE_HOME environment variable as described in "ORACLE_HOME environment variable" in the *Oracle TimesTen In-Memory Database Installation Guide*.

When installing an Oracle Client, choose the *Application User* Installation Type. It is not necessary to configure a directory service to use Cache Connect to Oracle, so you can skip this part of the installation procedure.

After you have installed the Oracle Client, install TimesTen as described in the *Oracle TimesTen In-Memory Database Installation Guide*.

Note: If you install TimesTen *before* installing the Oracle Client, you must either reboot your system (Windows) or restart TimesTen (UNIX).

Create required Oracle accounts

Before you can use Cache Connect to Oracle, you need to obtain an Oracle account from your database administrator. If you are acting as your own database administrator, open a command prompt window and start SQL*Plus on the Oracle server, logging in as the system manager:

```
sqlplus system/password@Oracle_Service_Name
```

For example, to create a new Oracle account for `testuser` on the Oracle database identified by the connection string, `system1`, enter the following:

```
sqlplus system/manager@system1
SQL> CREATE USER testuser IDENTIFIED BY mypsswrd;
SQL> GRANT connect, resource, create any trigger TO testuser;
SQL> COMMIT;
SQL> EXIT
```

The remaining sections in this chapter describe some example cache group operations for our Oracle user, `testuser`, who has a password of `mypsswrd` and an Oracle account on `system1`.

Create an account on TimesTen

As the instance administrator, use the **ttIsql** utility to connect to the instance data store, `TT_instance`. The instance data store is defined by TimesTen at installation time to give the instance administrator a data store connection on which administrative tasks can be performed. See "Instance data store" in *Oracle TimesTen In-Memory Database Installation Guide*.

Then use the CREATE USER and GRANT statements to create a user, named `testuser`, with ADMIN and DDL privileges:

```
ttIsql TT_instance
Command> CREATE USER testuser IDENTIFIED BY 'mypsswrd';
Command> GRANT ADMIN, DDL TO testuser;
```

See Chapter 1, "Access Control" in the *Oracle TimesTen In-Memory Database Installation Guide* for more information on TimesTen Access Control.

Create a TimesTen DSN

WINDOWS

On Windows, create a simple TimesTen system data source (System DSN), named `cgDSN`, as described in Chapter 1, "Creating TimesTen Data Stores" of *Oracle TimesTen In-Memory Database Operations Guide*.

For *cgDSN*, set:

- Data Store Path and Name: **c:\temp\cgds**
- Permanent Data Sz (MB): **16**
- Temporary Data Sz (MB): **16**
- User ID: **testuser** (this ID is also used as the Oracle User ID)
- Oracle ID: **system1**
- Oracle Password: **mypsswrd**
- DatabaseCharacterSet: The database character set must be the same as the Oracle database character set. To retrieve the Oracle database character set, enter the following query on the Oracle database:

  ```
  SELECT value FROM nls_database_parameters
       WHERE parameter='NLS_CHARACTERSET';
  ```

Use defaults for all other settings.

The `odbc.ini` file on UNIX platforms should have the following settings:

UNIX

```
[cgDSN]
Datastore=/temp/cgds
PermSize=16
TempSize=16
UID=testuser
OracleId=system1
OraclePwd=mypsswrd
```

```
DatabaseCharacterSet= Oracle database character set
     (see Windows explanation)
```

See "Defining DSNs for cached tables" on page 81 for more information on defining DSNs for cache groups.

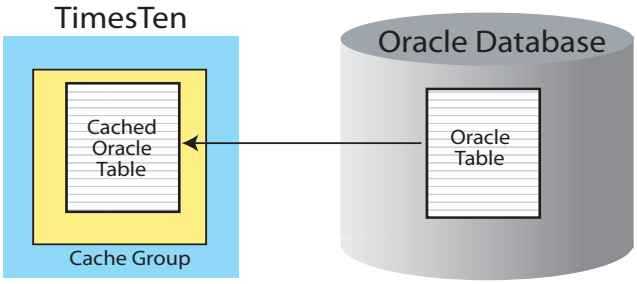
Creating a READONLY cache group

After you have TimesTen and Oracle set up and configured as described in "Setting up TimesTen and Oracle" on page 14, you are ready to create a cache group.

This section describes how to create a simple READONLY cache group that caches the contents of a single table in an Oracle database. Although a cache group can consist of multiple tables, we cache only one Oracle table to keep the example simple.

Figure 2.1 shows a READONLY cache group that caches a single Oracle table.

Figure 2.1 A Simple READONLY Cache Group

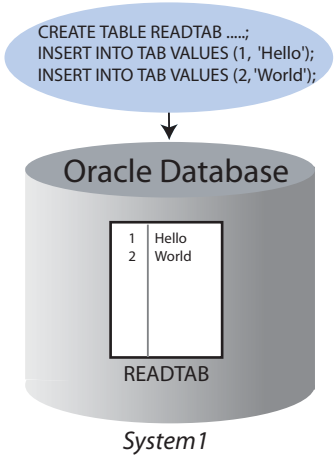


To create a simple READONLY cache group, perform the following steps:

- Step 1: Create an Oracle table
- Step 2: Create the cache group
- Step 3: Load the cache group
- Step 4: Update the Oracle table
- Step 5: Drop the cache group
- Step 6: Stop the cache agent

Step 1: Create an Oracle table

Figure 2.2 Creating an Oracle table



Connect to your new Oracle account and create a table:

```
sqlplus testuser/mypsswrd@system1
```

```
SQL> CREATE TABLE readtab (a NUMBER NOT NULL PRIMARY KEY,
                           b VARCHAR2(31));
```

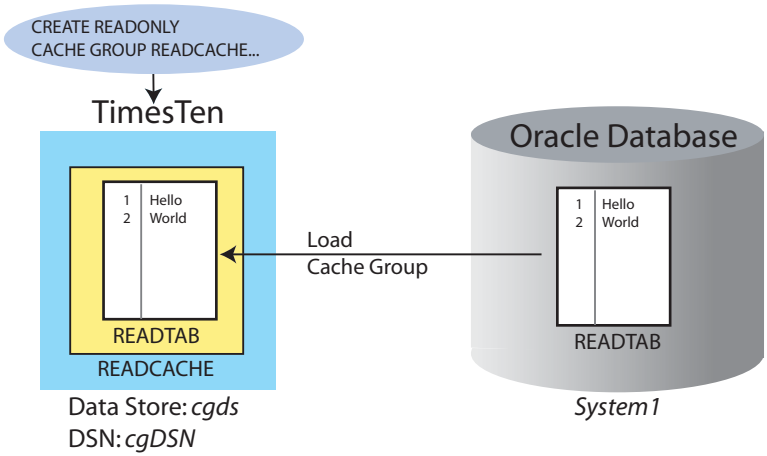
Then insert some rows and commit the changes:

```
SQL> INSERT INTO readtab VALUES (1, 'hello');
```

```
1 row created.
SQL> INSERT INTO readtab VALUES (2, 'world');
1 row created.
SQL> COMMIT;
Commit completed.
```

Step 2: Create the cache group

Figure 2.3 Creating a READONLY cache group



Use the **ttIsql** utility to connect to the `cgDSN` data store. At the command prompt, use the **ttCacheUidPwdSet** procedure to pass the cache administration user ID and password as parameters.Then call the **ttCacheStart** procedure to start the cache agent for the data store. In this example, the cache administration user ID is `testuser` and the password is `mypsswrd`:

```
> ttIsql cgDSN
```

```
Command> call ttCacheUidPwdSet('testuser','mypsswrd');
Command> call ttCacheStart;
```

Note: In this example, the cache administration user ID and password are the same as the Oracle user name and password. However, your Oracle user account may not have the privileges needed to autorefresh from Oracle and you may need to specify a separate cache administration user account. The cache administration user ID and password need to be set only once for a data store. See "Create Oracle users and set privileges" on page 76 for details.

Next, use the CREATE CACHE GROUP statement to create a READONLY cache group, named `readcache`, to cache the contents of the `readtab` Oracle table on TimesTen:

```
Command> CREATE READONLY CACHE GROUP readcache
> AUTOREFRESH INTERVAL 5 SECONDS
> FROM readtab
>(a NUMBER NOT NULL PRIMARY KEY, b VARCHAR2(31));
```

Step 3: Load the cache group

Load the contents of the Oracle table into the cache group table.

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

Check the contents of `readtab` table:

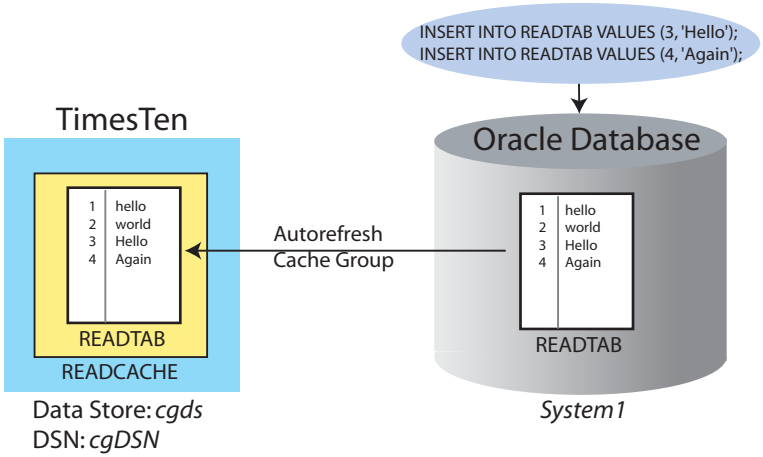
```
Command> SELECT * FROM readtab;
< 1, hello                          >
< 2, world                          >
2 rows found
```

Use the **ttIsql** `cachegroups` command to check the definition of the READCACHE cache group:

```
Command> cachegroups;

Cache Group TESTUSER.READCACHE:

  Cache Group Type: Read Only
  Autorefresh: Yes
  Autorefresh Mode: Incremental
  Autorefresh State: On
  Autorefresh Interval: 5 Seconds

  Root Table: TESTUSER.READTAB
  Table Type: Read Only

1 cache group found.
```

Step 4: Update the Oracle table

Figure 2.4 Autorefreshing TimesTen with Oracle updates



Using SQL*Plus, insert more rows into READTAB and commit the changes:

```
SQL> INSERT INTO readtab VALUES (3, 'Hello');
1 row created.
SQL> INSERT INTO readtab VALUES (4, 'Again');
1 row created.
SQL> COMMIT;
Commit completed.
```

After 5 seconds, TimesTen will automatically refresh its cached data from Oracle. Check the contents of the READTAB table in **ttIsql**:

```
Command> SELECT * FROM readtab;
< 1, hello                          >
< 2, world                          >
< 3, Hello                          >
< 4, Again                          >
4 rows found
```

Step 5: Drop the cache group

In the TimesTen window, use the DROP CACHE GROUP statement to remove the cache group from the TimesTen data store:

```
Command> DROP CACHE GROUP readcache;
```

Step 6: Stop the cache agent

Call the **ttCacheStop** procedure to stop the cache agent for the data store:

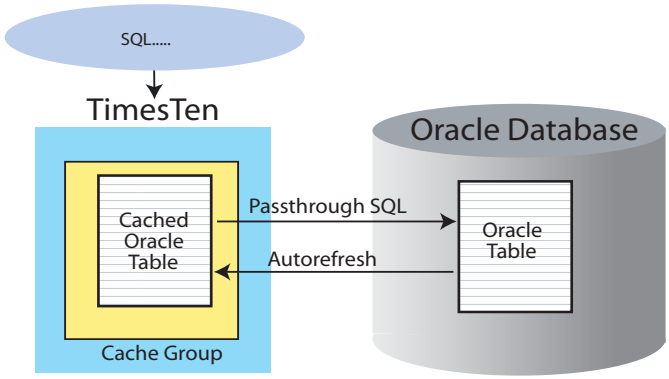
```
Command> call ttCacheStop;
```

Enabling the SQL passthrough feature

This section describes how to set the **PassThrough** attribute in your DSN to direct TimesTen to passthrough SQL to Oracle. For more information about the **PassThrough** attribute, see "Setting a passthrough level" on page 91.

Figure 2.5 shows SQL from an application being passed through to the Oracle table. The cached table receives the updates from Oracle by the autorefresh mechanism.

Figure 2.5 Passing SQL from Cache Group to Oracle



The steps for enabling the SQL passthrough feature are:

- Step 1: Create a new TimesTen DSN
- Step 2: Create a READONLY cache group
- Step 3: Load the cache group
- Step 4: Update the cache group table
- Step 5: Drop the cache group
- Step 6: Stop the cache agent

Step 1: Create a new TimesTen DSN

WINDOWS

On Windows, create a new TimesTen system data source (System DSN), named *cgPT2*, with the same attributes you specified for `cgDSN` in "Create an account on TimesTen" on page 16. In addition, set the **PassThrough** attribute value to '2' in order to pass queries and procedure calls for Oracle tables that are not in the cache group directly to Oracle.

For `cgPT2`, set:

- Data Store Path and Name: **c:\temp\cgPT2d**
- Permanent Data Sz (MB): **16**
- Temporary Data Sz (MB): **16**
- User ID: **testuser** (this ID is also used as the Oracle User ID)
- Oracle ID: **system1**
- Oracle Password: **mypsswrd**
- PassThrough: **2**
- DatabaseCharacterSet: The database character set must be the same as the Oracle database character set. To retrieve the Oracle database character set, enter the following query on the Oracle database:

```
SELECT value FROM nls_database_parameters
     WHERE parameter = 'NLS_CHARACTERSET';
```

Use defaults for all other settings.

UNIX

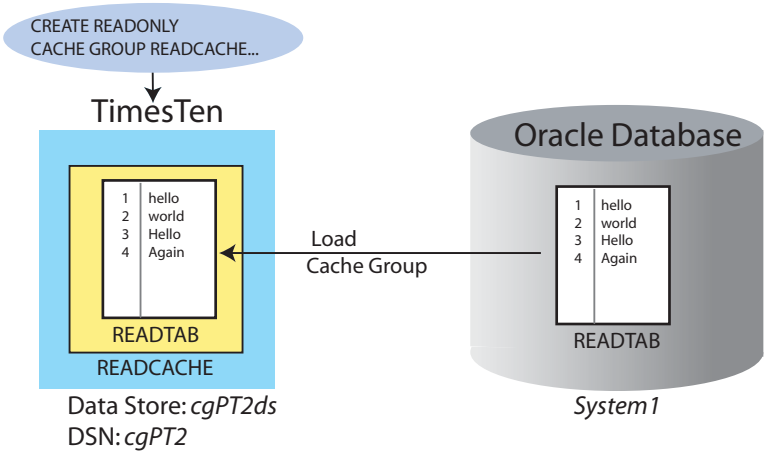
The `odbc.ini` file on UNIX platforms should have the following settings:

```
[cgPT2]
Datastore=/temp/cgPT2d
PermSize=16
TempSize=16
UID=testuser
OracleId=system1
OraclePwd=mypsswrd
Passthrough=2
DatabaseCharacterSet= Oracle database character set
     (see Windows explanation)
```

See "Defining DSNs for cached tables" on page 81 for more information on defining DSNs for cache groups.

Step 2: Create a READONLY cache group

Figure 2.6 Creating a READONLY cache group



In your TimesTen window, connect to DSN `cgPT2`, set the cache administration user ID and password, start the cache agent, and create the same READONLY cache group as you created in "Step 2: Create the cache group" on page 20:

```
ttIsql cgPT2
```

```
Command> call ttCacheUidPwdSet('testuser','mypsswrd');
Command> call ttCacheStart;
```

```
Command> CREATE READONLY CACHE GROUP readcache
> AUTOREFRESH INTERVAL 5 SECONDS
> FROM readtab
> (a NUMBER NOT NULL PRIMARY KEY, b VARCHAR2(31));
```

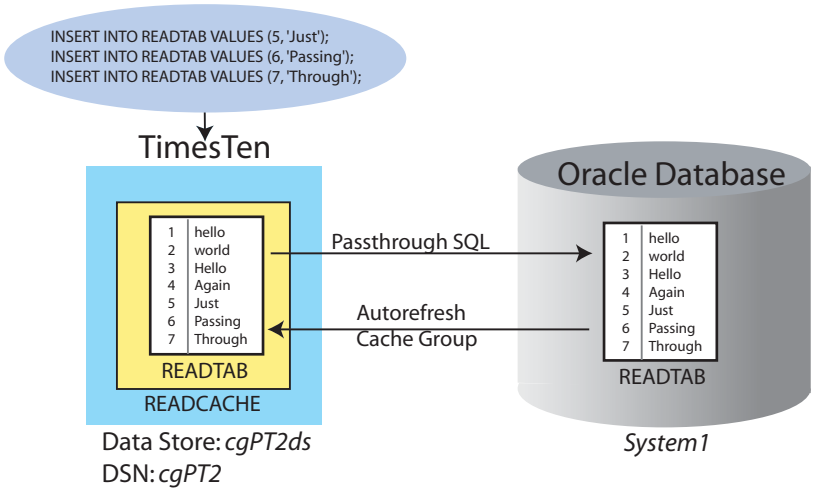
Step 3: Load the cache group

Load the contents of the Oracle table into the cache group table.

```
Command> LOAD CACHE GROUP readcache COMMIT EVERY 256 ROWS;
4 cache instances affected.
```

Step 4: Update the cache group table

Figure 2.7 Updating TimesTen Cache Group Table



Using **ttIsql**, insert some rows into `readtab`:

```
Command> INSERT INTO readtab VALUES (5, 'Just');
1 row inserted.
Command> INSERT INTO readtab VALUES (6, 'Passing');
1 row inserted.
Command> INSERT INTO readtab VALUES (7, 'Through');
1 row inserted.
```

Using SQL*Plus, check the contents of the `readtab` table on Oracle:

```
SQL> SELECT * FROM readtab;

         A B
---------- ------------------------------
         1 hello
         2 world
         3 Hello
         4 Again
         5 Just
         6 Passing
         7 Through

7 rows selected.
```

Then check the contents of the `readtab` table on TimesTen, using **ttIsql**:

```
Command> SELECT * FROM readtab;
< 1, hello                          >
< 2, world                          >
< 3, Hello                          >
< 4, Again                          >
< 5, Just                           >
< 6, Passing                        >
< 7, Through                        >
7 rows found
```

Step 5: Drop the cache group

Using **ttIsql**, enter the DROP CACHE GROUP statement to remove the cache group from the TimesTen data store:

```
Command> DROP CACHE GROUP readcache;
```

Step 6: Stop the cache agent

Call the **ttCacheStop** procedure to stop the cache agent for the data store:

```
Command> call ttCacheStop;
```

Checklist for creating a cache group

Table 2.1 Checklist for creating a cache group

Task Number	Task

1.	Ensure that Cache Connect to Oracle is installed. Use **ttIsql** to verify:
	`connect "uid=myuid;pwd=mypwd;OraclePWD=mypwd;passthrough=3";`
	`SELECT COUNT(*) FROM DUAL;`
	`exit`
	The query should return `1`. If it does not, then check the following:
	• The settings for the following environment variables: ORACLE_HOME, LD_LIBRARY_PATH, SHLIB_PATH
	• The cache administration user ID and password and the Oracle ID
	• The state of the Oracle server

2.	Design the cache group schema for all of the cache groups. You can use the Cache Administrator. See "Data type mappings for Cache Connect to Oracle" on page 70 for information about data mapping.

3.	Ensure that there are enough resources for loading all of the cache groups. Setting First Connection attributes: • **PermSize** - You can create the cache group first and use the **ttSize** utility to estimate the value for the **PermSize** attribute. You need an estimate of the number of rows in the cache to use the **ttSize** utility. • **TempSize** - There are no significant requirements. • **DatabaseCharacterSet** - Ensure that it matches the Oracle database character set. File system size recommendations: • The data store directory should be large enough to hold two checkpoint files. Each checkpoint file has a maximum size of 20 MB + **PermSize**. • The log directory should be large enough to hold the log files that accumulate between checkpoints. Note that an autorefresh transaction can be large if there are many changes in the Oracle tables during the autorefresh interval. A rule of thumb for log directory size is to make it 3 times the size of the data store plus 3 times **LogFileSize**. • The temp directory should be put on a fast file system to enhance the performance for large transactions. You can specify the temp directory be used for autorefresh operations by setting the TMPDIR environment variable (UNIX) or the TEMP environment variable (Windows). Restart the TimesTen daemon (UNIX) or restart the machine (Windows) after you set the environment variable. A large autorefresh transaction may require a large space in the temp directory.
4.	Set the cache administration user ID and password if the cache groups are autorefresh or asynchronous writethrough. The cache administration user ID must be an Oracle user and must have the required privileges. See "Set the cache administration user ID and password from the command line" on page 83 or "Set the cache administration user ID and password from a program" on page 86.
5.	Start the cache agent. Skip this step if you plan to load the cache groups with logging off (see Step 7). See "Starting and stopping the cache agent" on page 82.
6.	Create and commit all of the cache groups.

7. (Optional) Load cache groups with logging off.

For better performance and resource consumption, you can load the cache groups in no-logging mode. The disadvantages of no-logging mode are:

- All existing connections to the TimesTen data store must be stopped.
- The load operation cannot be replicated.

Perform the following tasks to load the cache groups with logging off:

a. Stop the cache agent, the replication agent, and the TimesTen server if they are running.

b. Disconnect all applications connected to the TimesTen data store.

c. Connect to the data store with First Connection attributes **Logging**=0; **DurableCommits**=0; **LockLevel**=1.

d. Issue the following SQL statement for each group: LOAD CACHE GROUP *cache_group_name* COMMIT EVERY 0 ROWS.

e. Commit the transaction and issue a checkpoint after each cache group load.

f. Reconnect the applications to the TimesTen data store with logging on.

g. Start the cache agent.

8. Create the TimesTen replication scheme on the cache group tables if you want them to be replicated.

9. If you are replicating the cache group tables or if the cache group is asynchronous writethrough (AWT), then start the replication agent. See "Starting the replication agent for an AWT cache group" on page 90.

Note: You cannot create or drop an AWT cache group while the replication agent is running.

10. Load the cache groups and commit (if you did not perform Step 7). Use LOAD CACHE GROUP *cache_group_name* COMMIT EVERY *n* ROWS. The recommended value for *n* is 256.

3

Defining a Cache Group

This chapter describes different types of cache groups and how to define them. It includes the following topics:

- Creating a cache group definition
- READONLY cache groups
- SYNCHRONOUS WRITETHROUGH (SWT) cache groups
- ASYNCHRONOUS WRITETHROUGH (AWT) cache groups
- USERMANAGED cache groups
- Defining cache group and table attributes
- AUTOREFRESH cache group attribute
- Cache table attributes
- Defining cache group tables
- Using WHERE clauses
- Implementing aging in a cache group
- Working with data types and type modes
- Data type mappings for Cache Connect to Oracle

Creating a cache group definition

You can create a cache group definition by using the CREATE CACHE GROUP statement, which includes a separate table definition for each Oracle table to be cached.

Note: If TimesTen was installed with Access Control enabled, you must have DDL privileges to the data store to use CREATE CACHE GROUP statement. In addition, the TimesTen user name must match the Oracle user name (this applies to either an internal or external user). See Chapter 1, "Access Control" in *Oracle TimesTen In-Memory Database Installation Guide* for details.

Table 3.1 shows the components of a simple cache group definition. Each portion of a cache group definition is described in the following sections.

Table 3.1 Components of a cache group definition

Component	See...
`CREATE` *`type`* `CACHE GROUP`	"Selecting a cache group type" on page 32
`owner.name`	"Naming a cache group" on page 32
`Cache group and table attributes`	"Defining cache group and table attributes" on page 47
`FROM` *`table definition`*	"Defining cache group tables" on page 53
`[WHERE ...]`	"Using WHERE clauses" on page 60
`[AGING ...]`	"Implementing aging in a cache group" on page 64

Complete syntax for the CREATE CACHE GROUPstatement is provided in *Oracle TimesTen In-Memory Database SQL Reference Guide*.

You can also use the Cache Administrator to create a cache group using your Web browser. See Chapter 6, "Cache Administrator" for details.

Naming a cache group

In the CREATE CACHE GROUP statement, identify the name of the cache group using the following form:

`owner.name`

If you exclude the owner from the name of your cache group, then the ID or the current session user is used as the owner.

Selecting a cache group type

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

- READONLY cache groups
- SYNCHRONOUS WRITETHROUGH (SWT) cache groups
- ASYNCHRONOUS WRITETHROUGH (AWT) cache groups

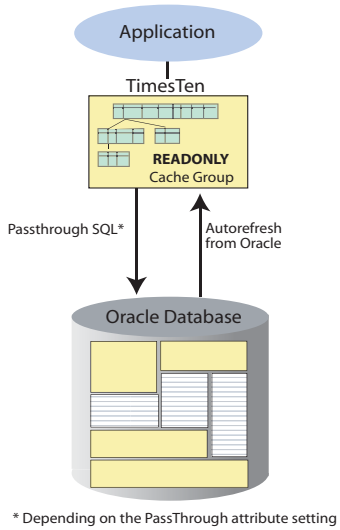
See "USERMANAGED cache groups" on page 43 for information about user-managed cache groups.

READONLY cache groups

A READONLY cache group enforces a caching behavior in which updates on Oracle tables are applied to TimesTen through the AUTOREFRESH mechanism. The cache cannot be updated directly. See "AUTOREFRESH cache group attribute" on page 48.

Figure 3.1 shows a READONLY cache group.

Figure 3.1 READONLY cache group



Use the CREATE READONLY CACHE GROUP statement to create a READONLY cache group. By default:

- The AUTOREFRESH cache group attribute is set to INCREMENTAL mode.
- The AUTOREFRESH INTERVAL value is 5 MINUTES.
- The AUTOREFRESH STATE is PAUSED.

You can use the ALTER CACHE GROUP statement to change any of the default settings of the AUTOREFRESH cache group attribute or to disable it entirely to prevent updates to the Oracle tables from being applied to the cache. See "AUTOREFRESH cache group attribute" on page 48 for details.

Before you create a READONLY cache group, set the cache administration user ID and password by using the **ttCacheUidPwdSet** built-in procedure or the **ttAdmin** utility with the `-cacheUidPwdSet` option. The cache administration user creates a trigger on Oracle, so it must have the Oracle privileges listed for CREATE READONLY CACHE GROUP in Table 4.1 on page 77.

The cache administration user ID and password need to be set only once for each data store. If a data store is overwritten or destroyed, then the cache administration user ID and password must be reset.

Attempts to update a cached table in a READONLY cache group result in TimesTen error 8225 “Table is read only”. However, if the **PassThrough** attribute is set to either 2 or 3, DML statements may be passed through the cache to Oracle and then propagated by AUTOREFRESH back into the cache group from Oracle. The effects of a passed-through statement on a READONLY cache group are not seen in the transaction that contains the statement. They can be seen only after the transaction has committed on Oracle and after the next AUTOREFRESH of the cache has completed. See “Setting a passthrough level” on page 91 for more information about the **PassThrough** attribute.

Restrictions on using READONLY cache groups

The following restrictions apply when using a READONLY cache group:

- The tables in a READONLY cache group cannot be updated directly.
- From the cache group and table attributes described in “Defining cache group and table attributes” on page 47, the following attributes can be used:
  - AUTOREFRESH
  - UNIQUE HASH ON
  - ON DELETE CASCADE
- Manual FLUSH operations described in “USERMANAGED cache groups” on page 43 are not allowed.
- The TRUNCATE TABLE statement cannot be autorefreshed.
- The cache group must be empty when you use the LOAD CACHE GROUP statement.
- The AUTOREFRESH STATE must be PAUSED when you use the LOAD CACHE GROUP or REFRESH CACHE GROUP statement.
- All fields (columns and tables) referenced in WHERE clauses must be fully qualified. For example: `user.table` and `user.table.column`.
- LOAD CACHE GROUP and REFRESH CACHE GROUP statements cannot contain a WHERE clause.
- Least recently used (LRU) aging cannot be specified. See “Implementing aging in a cache group” on page 64.

Example: Creating a READONLY cache group

This example creates a READONLY cache group, called *customer_orders*, for the *customer* and *ordertab* tables.

```
CREATE READONLY CACHE GROUP customer_orders
FROM
```

```
user1.customer (custid INTEGER NOT NULL,
       name VARCHAR2(100) NOT NULL,
       addr VARCHAR2(100),
       zip VARCHAR2(10),
       region VARCHAR2(10),
       PRIMARY KEY(custid)),
user1.ordertab (orderid NUMBER NOT NULL,
       custid INTEGER NOT NULL,
       PRIMARY KEY (orderid),
       FOREIGN KEY (custid) REFERENCES CUSTOMER(custid));
```

SYNCHRONOUS WRITETHROUGH (SWT) cache groups

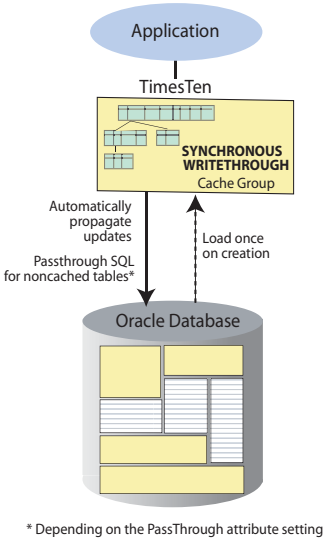
A SYNCHRONOUS WRITETHROUGH (SWT) cache group enforces a caching behavior in which cached data is updated in TimesTen and propagated to Oracle. When an SWT cache group is created by the CREATE SYNCHRONOUS WRITETHROUGH CACHE GROUP statement, the contents of the cache group are manually loaded from the Oracle tables after the cache group is created. The contents of an SWT cache group can be manually loaded, unloaded, or refreshed as often as desired.

Updates to a SWT cache group are committed synchronously. When the application commits a transaction, it is committed on Oracle before it is committed on TimesTen. The application is blocked and a lock is held on the table row until the transaction has committed on TimesTen.

If the Oracle transaction fails to commit, then the TimesTen transaction must be rolled back. If the Oracle transaction commits but the TimesTen transaction fails, then the cache group data becomes out of sync with the Oracle data. If this happens, you need to manually resynchronize the cache group with Oracle. This can be done by calling the **ttCachePropagateFlagSet** procedure to disable propagation and then reapplying the transaction on the TimesTen cache group. An alternative is to load the data from Oracle again.

Figure 3.2 shows a SYNCHRONOUS WRITETHROUGH cache group.

Figure 3.2 SYNCHRONOUS WRITETHROUGH cache group



Restrictions on using SWT cache groups

The following restrictions apply when using a SWT cache group:

- From the cache group and table attributes described in "Defining cache group and table attributes" on page 47, the UNIQUE HASH ON and ON DELETE CASCADE attributes can be used.
- The manual FLUSH operation described in "USERMANAGED cache groups" on page 43 is not allowed.
- WHERE clauses, as described in "Using WHERE clauses" on page 60, cannot appear in the table definitions of this type of cache group.
- The TRUNCATE TABLE statement cannot be applied to tables in cache groups.

Example: Creating an SWT cache group

This example creates a SYNCHRONOUS WRITETHROUGH cache group, called *vendors*, for a single Oracle table, *vendor*. Updates on the *vendor* table in the cache group are automatically propagated to Oracle.

```
CREATE SYNCHRONOUS WRITETHROUGH CACHE GROUP vendors
FROM
user1.vendor (vendor_id INTEGER NOT NULL,
       vendor_name VARCHAR2(100) NOT NULL,
       contact_name VARCHAR2(100) NOT NULL,
       phone VARCHAR2(15),
       street VARCHAR2(100),
```

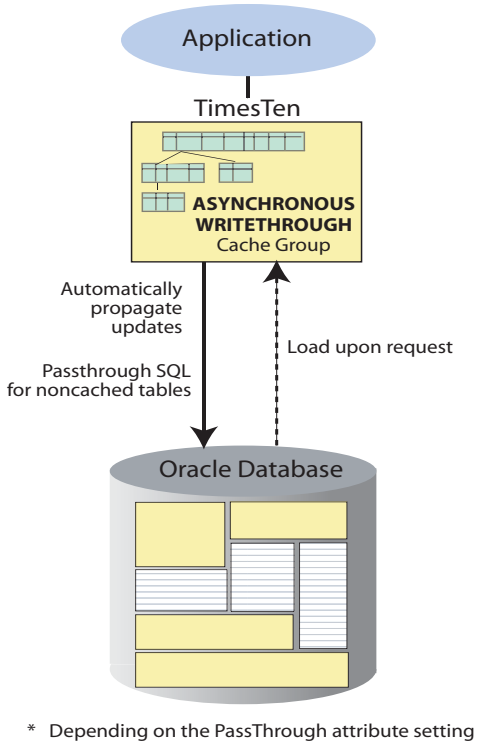
```
        city VARCHAR2(30),
        state VARCHAR2(30),
        zip VARCHAR2(10),
        PRIMARY KEY(vendor_id));
```

ASYNCHRONOUS WRITETHROUGH (AWT) cache groups

An ASYNCHRONOUS WRITETHROUGH (AWT) cache group enforces the same caching behavior as an SWT cache group, in which cached data is updated in TimesTen and propagated to Oracle. AWT cache groups provide better response times than SWT cache groups because the TimesTen commit occurs asynchronously from the Oracle commit. This allows an application to continue executing without having to wait for the Oracle transaction to commit. You can also update an AWT cache group when the Oracle database is down. When the Oracle database returns to operation, the updates will be applied to the Oracle database.

Figure 3.3 shows that updates from the TimesTen cache group are asynchronously replicated to Oracle.

Figure 3.3 ASYNCHRONOUS WRITETHROUGH cache group



An AWT cache group requires that you start both the cache agent and replication agent for the data store containing the cache group. The cache agent enables you to load and refresh the cache. The cache agent does not need to be running to unload the cache.

An AWT cache group is created by the CREATE ASYNCHRONOUS WRITETHROUGH CACHE GROUP statement.

To use an AWT cache group, you must have the Oracle privileges listed in "Create Oracle users and set privileges" on page 76 and set the cache administration user ID and password with the **ttCacheUidPwdSet** procedure before starting the cache agent and the replication agent. See "Starting the replication agent for an AWT cache group" on page 90 for more information.

When using an AWT cache group, transactions are committed on TimesTen and Oracle asynchronously. Thus your application cannot be sure when the changes will be committed to Oracle.

The rest of this section includes the following topics:

- What an AWT cache group guarantees
- What an AWT cache does not guarantee
- Restrictions on using AWT cache groups
- Example: Creating an AWT cache group
- Creating Oracle objects for AWT
- Setting up an AWT cache group

What an AWT cache group guarantees

An AWT cache group *can* guarantee the following:

- No transactions will be lost because of communication failures between TimesTen and Oracle.
  - If the replication agent loses its connection to Oracle, then AWT will resume after the agent is able to reconnect to Oracle.
  - If the replication agent on the data store with the cache group goes down, the agent will come back up and start transmitting again.
- Transactions committed against a single TimesTen data store are committed on Oracle in the same order in which they were committed on TimesTen.
- If a transaction fails on Oracle, the failure will be reported in the *dataStoreName*.`awterr` error file. (See "Permanent Oracle errors reported by TimesTen" in *Oracle TimesTen In-Memory Database Troubleshooting Procedures Guide*.)

What an AWT cache does not guarantee

An AWT cache group *cannot* guarantee:

- That all transaction executed in TimesTen will be applied to Oracle. Execution errors on Oracle cause the entire transaction to be rolled back. For example, an insert on Oracle may fail because of a unique constraint violation. Transactions with execution errors are not retried.

- That the absolute order of Oracle changes is preserved because AWT does not resolve update conflicts. Here are some examples:
  - A row is first updated on an AWT table and is committed. The same row is then updated through an Oracle passthrough operation and is committed. The AWT agent eventually applies the first update to the row and overwrites the passthrough operation.
  - On two separate data stores (DS1, DS2), there is an AWT table for the same Oracle base table. A row is updated on DS1 and is committed. A row is updated on DS2 and is committed. Because the cache group behavior is asynchronous, the change on DS2 may be applied to the Oracle database before the change on DS1, resulting in the DS1 change overwriting the DS2 change.

Restrictions on using AWT cache groups

The restrictions described for SWT cache groups in "Restrictions on using SWT cache groups" on page 36 also apply to AWT cache groups:

- From the cache group and table attributes described in "Defining cache group and table attributes" on page 47, the UNIQUE HASH ON and ON DELETE CASCADE attributes can be used.
- The manual FLUSH operation described in "USERMANAGED cache groups" on page 43 is not allowed.
- WHERE clauses, as described in "Using WHERE clauses" on page 60, cannot appear in the table definitions of this type of cache group.
- The TRUNCATE TABLE statement cannot be applied to tables in cache groups.
- VARCHAR2, NVARCHAR2, VARBINARY and TT_VARCHAR columns in AWT cached tables must be limited to 256K bytes.

The following additional restrictions apply *only* to AWT cache groups:

- The maximum length of the path name for a data store in an AWT cache group cannot exceed 248 characters.
- Errors and warnings generated during the propagation of changes to Oracle are recorded in a special error file and may be reported long after the commit to TimesTen occurred. See "Permanent Oracle errors reported by TimesTen" in *Oracle TimesTen In-Memory Database Troubleshooting Procedures Guide* for details.
- AWT cache groups are not allowed on data stores with logging turned off (**Logging**=0).
- The replication agent must be stopped before creating or dropping an AWT cache group. See "Setting up an AWT cache group" on page 41.
- Changes are not propagated from TimesTen to Oracle until the replication agent is started.

- Update conflicts on Oracle are not detected or resolved. Updates made directly on an Oracle base table will be overwritten when an update is propagated from TimesTen.
- Constraint checking is done on immediate operations associated with a single SQL statement. For example, suppose there is a unique index composed of an integer field on an AWT table. There are 10 records and the value of that field ranges from 1 to 10. An update statement is issued to increment that field. The statement succeeds on TimesTen but will likely fail when applied to Oracle. The reason is that TimesTen checks the unique index constraint at the end of the statement, but when the update is applied to Oracle, the constraint is checked after each row has been updated. Thus when the record with value 1 is set to 2, it will conflict with the row that already has value 2 for the field.
- You cannot include an AWT cache group in a replication scheme with the return twosafe service.

Example: Creating an AWT cache group

This example creates an ASYNCHRONOUS WRITETHROUGH cache group, called *customers*, for a single table, *customer*.

```
CREATE ASYNCHRONOUS WRITETHROUGH CACHE GROUP customers
FROM
user1.customer (custid INTEGER NOT NULL,
       name VARCHAR2(100) NOT NULL,
       addr VARCHAR2(100),
       zip VARCHAR2(10),
       PRIMARY KEY(custid));
```

Creating Oracle objects for AWT

TimesTen requires a state table to be created on Oracle to support AWT. The table is used to track the state and the last transaction that was applied to Oracle. This table is created automatically by the CREATE ASYNCHRONOUS WRITETHROUGH CACHE GROUP statement. See "Managing Oracle objects for READONLY, AUTOREFRESH, and AWT cache groups" on page 95.

Alternatively, the Oracle table can be created manually *before* an AWT cache group is created. See "Manually installing Oracle objects" on page 99.

Setting up an AWT cache group

When you set up an AWT cache group, perform the following tasks:

1. Set the cache administration user ID and password.

   Set the cache administration user ID and password by using the **ttCacheUidPwdSet** built-in procedure or the **ttAdmin** utility with the `-cacheUidPwdSet` option.

The cache administration user applies changes to Oracle. The cache administration user account must have the Oracle privileges listed for CREATE ASYNCHRONOUS WRITETHROUGH CACHE GROUP in Table 4.1 on page 77.

The cache administration user ID and password need to be set only once for each data store. If a data store is overwritten or destroyed, then the cache administration user ID and password must be reset.

2. Create the AWT cache group.

   Use the CREATE ASYNCHRONOUS WRITETHROUGH CACHE GROUP SQL statement.

3. Start the cache agent.

   See "Starting and stopping the cache agent" on page 82.

4. Start the replication agent.

   See "Starting the replication agent for an AWT cache group" on page 90.

   Creating an AWT cache group automatically creates a replication scheme that allows the data store to communicate with the Oracle database. The replication scheme is managed entirely by TimesTen and does not require user intervention. The replication scheme is dropped when you use DROP CACHE GROUP to drop the AWT cache group.

   The replication agent does not have to be started before loading the cache group, but better performance is achieved if the replication agent is started first.

5. Load the cache group.

   Use the LOAD CACHE GROUP statement.

USERMANAGED cache groups

If the system-managed cache groups (READONLY, AWT and SWT) do not meet your needs, you can use the CREATE USERMANAGED CACHE GROUP statement to create a cache group that implements customized caching behavior:

- You can define a USERMANAGED cache group to automatically refresh and propagate updates between Oracle and TimesTen by setting the AUTOREFRESH and PROPAGATE attributes. Setting both attributes enables *bidirectional propagation*, so that updates to either the TimesTen cache or Oracle are propagated to the other. See "Defining cache group and table attributes" on page 47.
- You can use SQL statements described in "Copying data between the Oracle database and cache groups" on page 108 to control the propagation of data between Oracle and TimesTen. For example, you can:
  - Use the FLUSH CACHE GROUP SQL statement in your application to flush updates made in the TimesTen cache group to Oracle.
  - Use the LOAD CACHE GROUP or REFRESH CACHE GROUP statements to load or refresh updates made in Oracle to the TimesTen cache.
- You can remove all data or selected data from a user-managed cache group by using the UNLOAD CACHE GROUP statement.
- You can specify either a PROPAGATE or READONLY attribute on each table in a user-managed cache group to define writethrough or read-only behavior at the table level. See "Cache table attributes" on page 50 for details.

Examples: Creating USERMANAGED cache groups

This section shows some CREATE USERMANAGED CACHE GROUP statements used to create customized cache groups.

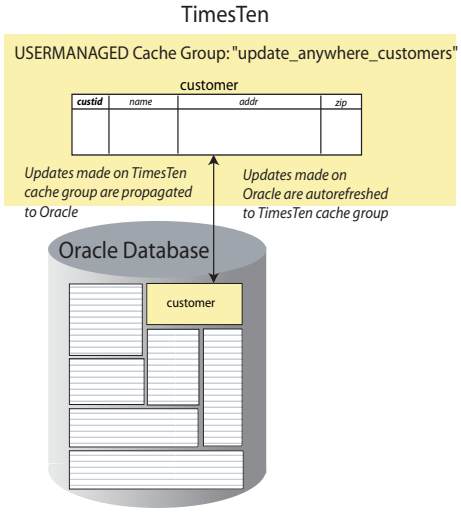
This section includes the following examples:

- Example 3.1 shows an example of a simple USERMANAGED cache group.
- Example 3.2 shows an example of a more complex USERMANAGED cache group.

Example 3.1

This example creates a USERMANAGED cache group, called *update_anywhere_customers*, for a single table, *customer*. Updates on the *customer* table in either TimesTen or Oracle are "bidirectionally" propagated to the other. Figure 3.4 shows the *update_anywhere_customers* USERMANAGED cache group.

Figure 3.4 Simple USERMANAGED cache group

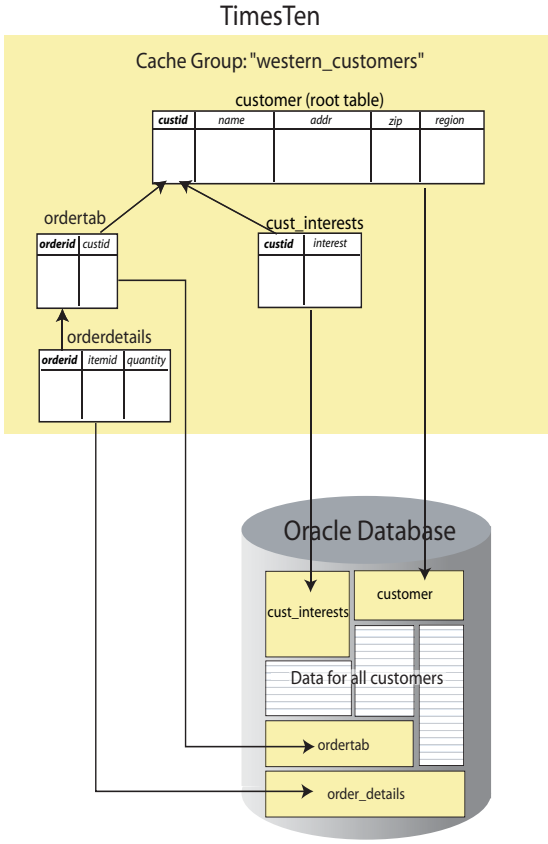


In this example, the AUTOREFRESH attribute is set so that the cache group is incrementally updated from Oracle every 30 seconds. The user is *user1*. The PROPAGATE attribute in the table description indicates that updates to the *customer* table in the TimesTen cache are propagated to Oracle:

```
CREATE USERMANAGED CACHE GROUP update_anywhere_customers
AUTOREFRESH
       MODE INCREMENTAL
       INTERVAL 30 SECONDS
FROM
user1.customer (custid INTEGER NOT NULL,
       name VARCHAR2(100) NOT NULL,
       addr VARCHAR2(100),
       zip VARCHAR2(10),
       PRIMARY KEY(custid),
       PROPAGATE);
```

Example 3.2 This example creates a USERMANAGED cache group, called *western_customers*, that contains four related tables: *customer*, *ordertab*, *order_details* and *cust_interest*. Figure 3.5 shows the cache group and its tables.

Figure 3.5 Complex USERMANAGED cache group



Each table within the *western_customers* cache group has a primary key. The tables in the cache group are linked together through a foreign key relationship. The *customer* table is the root table, so it does not refer to another table in the cache group. The root table includes a WHERE clause to restrict the rows to be cached in the table.

The PROPAGATE attribute specifies that any changes to the tables in the cache will be automatically propagated to the corresponding Oracle table at commit time.

```
CREATE USERMANAGED CACHE GROUP western_customers
    FROM
    user1.customer (
       custid INTEGER NOT NULL,
       name VARCHAR2(100) NOT NULL,
       addr VARCHAR2(100),
       zip VARCHAR2(10),
       region VARCHAR2(10),
```

```
   PRIMARY KEY(custid),
   PROPAGATE)
   WHERE (customer.region = 'Western'),
user1.ordertab(orderid NUMBER NOT NULL,
   custid INTEGER NOT NULL,
   PRIMARY KEY (orderid),
   FOREIGN KEY (custid) REFERENCES customer(custid),
   PROPAGATE),
user1.order_details(orderid NUMBER NOT NULL,
   itemit NUMBER NOT NULL,
   quantity NUMBER NOT NULL,
   PRIMARY KEY (orderid, itemid),
   FOREIGN KEY (orderid) REFERENCES ordertab(orderid),
   PROPAGATE),
user1.cust_interest(custid INTEGER NOT NULL,
   interest VARCHAR2(10) NOT NULL,
   PRIMARY KEY (custid, interest),
   FOREIGN KEY (custid) REFERENCES customer(custid),
   PROPAGATE);
```

Defining cache group and table attributes

The following tables summarize the attributes that can be used in the CREATE CACHE GROUP statement.

Table 3.2 summarizes the AUTOREFRESH cache group attribute. See "AUTOREFRESH cache group attribute" on page 48 for details.

Table 3.3 lists the attributes that can be applied to the individual table definitions in the cache group. Each attribute is described in detail in "Cache table attributes" on page 50.

Table 3.2 Attributes for the entire cache group

Cache Group Attribute	Description
AUTOREFRESH	Automatically applies changes made to the Oracle tables to the TimesTen cache. This attribute can be set on READONLY and USERMANAGED cache groups.

Table 3.3 Attributes for tables in a cache group

Cache Table Attribute	Description
UNIQUE HASH ON	Specifies that a hash index is to be created on this table. Can be set for tables in any type of cache group.
PROPAGATE	Automatically propagates changes made to the cached table to the corresponding Oracle table at commit time. Can be set for tables in USERMANAGED cache groups only.
READONLY	Specifies that the table in the cache group cannot be updated on TimesTen. Can be set for tables in USERMANAGED cache groups only.
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. Can be set for tables in any type of cache group. See "ON DELETE CASCADE" on page 52 for restrictions on: • Cache group tables with PROPAGATE attribute • SWT and AWT cache group tables

AUTOREFRESH cache group attribute

The AUTOREFRESH attribute is an optional attribute for READONLY and USERMANAGED cache groups.

AUTOREFRESH automatically applies changes from Oracle tables to the TimesTen cache.

TimesTen supports two AUTOREFRESH modes:

- FULL: The entire cache group is refreshed by unloading its contents and then reloading from the Oracle tables.
- INCREMENTAL: Oracle tracks changes and periodically updates only the rows in the cache group that have changed in Oracle. This mode uses special objects in Oracle to track the changes. See "Managing Oracle objects for READONLY, AUTOREFRESH, and AWT cache groups" on page 95 for details on how to create and manage these objects.

By default:

- The AUTOREFRESH MODE is INCREMENTAL.
- The AUTOREFRESH STATE is PAUSED.
- The AUTOREFRESH INTERVAL value is 5 MINUTES.

Cache groups with the same autorefresh interval are refreshed at the same time.

AUTOREFRESH in INCREMENTAL mode incurs some additional overhead to refresh the cache group for each update on Oracle. There is no additional overhead when using AUTOREFRESH in FULL mode.

Before you create a cache group with the AUTOREFRESH attribute, set the cache administration user ID and password by using the **ttCacheUidPwdSet** built-in procedure or the **ttAdmin** utility with the `-cacheUidPwdSet` option. The cache administration user creates a trigger on Oracle, so it must have the Oracle privileges listed in Table 4.1 on page 77 for READONLY cache groups and USERMANAGED cache groups with the AUTOREFRESH attribute.

The cache administration user ID and password need to be set only once for each data store. If a data store is overwritten or destroyed, then the cache administration user ID and password must be reset.

Specifying the AUTOREFRESH cache group attribute

You can specify AUTOREFRESH when you create a cache group using the CREATE CACHE GROUP statement. After the cache group has been created, you can use ALTER CACHE GROUP to change the MODE, STATE and INTERVAL settings. The ALTER CACHE GROUP statement cannot be used to instantiate AUTOREFRESH for a cache group that was created without the AUTOREFRESH attribute.

How often AUTOREFRESH occurs is determined by its INTERVAL value. You can use the CREATE CACHE GROUP or the ALTER CACHE GROUP statements to set the AUTOREFRESH STATE to ON, OFF or PAUSED. AUTOREFRESH is scheduled by TimesTen when the transaction that sets its STATE to ON is committed.

When the STATE is set to OFF, changes to the Oracle tables are not captured or recorded. When the state is PAUSED, changes to the Oracle tables are captured and recorded on Oracle, but are not applied to the tables in the TimesTen cache group.

When an AUTOREFRESH operation is in progress and you try to change the STATE to OFF or drop the cache group:

- The AUTOREFRESH operation stops if the **LockWait** general connection attribute is greater than 0. The ALTER CACHE GROUP SET AUTOREFRESH STATE OFF and DROP CACHE GROUP statements preempt the AUTOREFRESH operation.
- The AUTOREFRESH operation continues if the **LockWait** general connection attribute is 0. The ALTER CACHE GROUP SET AUTOREFRESH STATE OFF and DROP CACHE GROUP statements will fail with a lock timeout error.

When using AUTOREFRESH in INCREMENTAL mode, changes taking place on Oracle are maintained in a *change log table*. Under certain circumstances, it is possible for some of the records in a transaction to be purged from the change log table before they are applied to the TimesTen cache group. If this occurs, Cache Connect to Oracle initiates a full autorefresh of the cache group. See "Incremental autorefresh becomes full autorefresh" in *Oracle TimesTen In-Memory Database Troubleshooting Procedures Guide* for an explanation of why records may be purged from the change log table.

Restrictions on using AUTOREFRESH

The following restrictions apply when using AUTOREFRESH:

- Incremental AUTOREFRESH will not work if the TRUNCATE TABLE statement is used on an Oracle base table. If TRUNCATE is used on an Oracle base table, then you must reset AUTOREFRESH. Use the ALTER CACHE GROUP statement to set the AUTOREFRESH STATE to PAUSED, then manually refresh the cache group. Issue another ALTER CACHE GROUP to reset the AUTOREFRESH STATE to ON.
- To use the AUTOREFRESH feature, *all* of the cache group tables must be specified as either PROPAGATE or READONLY. You cannot specify AUTOREFRESH with the NOT PROPAGATE attribute.
- The AUTOREFRESH STATE must be PAUSED when you manually load or refresh a cache group.

- The cache group must be empty when you use the LOAD CACHE GROUP statement.
- All fields (columns and tables) referenced in WHERE clauses must be fully qualified. For example: `user.table` and `user.table.column`
- LOAD CACHE GROUP and REFRESH CACHE GROUP statements for AUTOREFRESH cache groups cannot have a WHERE clause.
- You cannot implement LRU aging on AUTOREFRESH cache groups. See "Implementing aging in a cache group" on page 64.

See Example 3.1 for an example USERMANAGED cache group with the AUTOREFRESH attribute.

Cache table attributes

This section includes the following cache table attributes:

- PROPAGATE
- READONLY
- ON DELETE CASCADE
- UNIQUE HASH ON

PROPAGATE

The PROPAGATE attribute can be specified only for tables in USERMANAGED cache groups. However, the propagation behavior described here is similar for synchronous writethrough (SWT) cache groups.

PROPAGATE specifies that any changes to a table in the TimesTen cache group are to be automatically propagated to the corresponding Oracle tables at commit time. NOT PROPAGATE disables propagation.

If your cache tables are defined with the PROPAGATE attribute, updates made to the cached table are propagated back to Oracle during TimesTen commit processing. This is done in a synchronous fashion:

1. The commit is first attempted in Oracle. If the commit fails in Oracle, the commit is not attempted in TimesTen and the TimesTen transaction is marked to be in need of rollback. This way, the Oracle database never misses data updates.
2. If the commit succeeds in Oracle, it is attempted in TimesTen. If the commit fails in TimesTen, you receive a TimesTen error message indicating the cause of the failure.

   Commits are done by connecting to Oracle on the application user's behalf. The Oracle login and password for the user must be specified in the DSN or on the connect string for TimesTen, with the **UID** and **OraclePWD** attributes, as described in "Defining DSNs for cached tables" on page 81.

For commits to be propagated back to Oracle, the following requirements must be met:

- The columns of the Oracle table being cached in TimesTen must include all the columns of at least one unique or primary key on that Oracle table. The columns making up the unique or primary key must be non-nullable in Oracle and they must be declared as primary keys in the TimesTen cache group table.

  The purpose of this requirement is to make sure that there is a one-to-one mapping between rows being updated in TimesTen and their original versions in Oracle. Given this one-to-one mapping, you can apply updates back without fear of affecting non-cached rows.
- The cached version of the table must have a primary key declared.

By default, a cache group table is created with the NOT PROPAGATE attribute so that updates to the table are not propagated to Oracle. When PROPAGATE is enabled, your application may occasionally need to make changes to a cached table that should not be committed to Oracle. Use the **ttCachePropagateFlagSet** built-in procedure to disable and then re-enable propagation.

When propagation is disabled, you can use the FLUSH CACHE GROUP statement to selectively propagate inserts and updates to Oracle, as described in "Flushing a USERMANAGED cache group" on page 115.

The following restrictions apply when using PROPAGATE:

- If you use PROPAGATE, it must be specified for all tables in the cache group.
- You cannot use both the PROPAGATE and READONLY cache table attributes in the same cache group.
- TimesTen does not do a conflict check to prevent the operation from overwriting more recently updated data on Oracle. For this reason, updates should be done in either the TimesTen cache or Oracle, but not both.
- NOT PROPAGATE cannot be set on a table if AUTOREFRESH is enabled for the cache group or on a replicated copy of an AUTOREFRESH cache group.
- After a cache group has been specified as PROPAGATE, you cannot change this attribute.

Note: For a cache group whose tables have the PROPAGATE attribute, TimesTen does not check whether inserts and updates to the tables are consistent with a WHERE clause that is part of the cache group definition. Such inserts and updates are not prevented and may also be propagated to Oracle.

See Example 3.2 for an example table definition that uses PROPAGATE.

READONLY

The READONLY table attribute can be specified only for tables in USERMANAGED cache groups.

Note: Do not confuse the READONLY table attribute with the READONLY cache group type. READONLY cache groups contain read-only tables, but READONLY tables can be explicitly specified only for USERMANAGED cache groups.

The READONLY table attribute can be specified for each cache group table to prohibit updates to that table by your TimesTen application.

The following restrictions apply when using the READONLY table attribute:

- If you use READONLY, it must be specified for all tables in the cache group.
- You cannot specify the READONLY table attribute with the PROPAGATE table attribute.

See "Setting a passthrough level" on page 91 for special considerations when using READONLY tables with the **PassThrough**=2 setting.

ON DELETE CASCADE

The ON DELETE CASCADE table attribute can be specified for tables in all types of cache groups when the cache group is created. It can also be specified for tables that are not in cache groups.

The ON DELETE CASCADE table attribute 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. The root table is the parent table of all other tables in the cache group.

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 ON DELETE CASCADE for child tables on a "delete" path.

The following restrictions apply when using ON DELETE CASCADE:

- For AWT and SWT cache groups and for cache group tables with the PROPAGATE attribute, foreign keys in TimesTen tables that have the ON DELETE CASCADE attribute must be a proper subset of the foreign keys in the Oracle tables that have the ON DELETE CASCADE attribute. ON DELETE CASCADE actions on the Oracle tables are applied to TimesTen as individual deletes. ON DELETE CASCADE actions on TimesTen are applied to Oracle as a cascaded operation.
- Matching of foreign keys between TimesTen and Oracle is enforced only at the time of cache group creation. ON DELETE CASCADE may not work correctly if the foreign key on Oracle is altered later.

See "CREATE CACHE GROUP" in *Oracle TimesTen In-Memory Database SQL Reference Guide*.

UNIQUE HASH ON

The UNIQUE HASH ON attribute can be specified for tables in all types of cache groups. It can also be specified for tables that are not in cache groups.

UNIQUE HASH ON specifies that a hash index is created on a table in a cache group. The columns specified in the hash index must be identical to the columns in the primary key. See "CREATE CACHE GROUP" in *Oracle TimesTen In-Memory Database SQL Reference Guide*.

Defining cache group tables

The most basic type of cache group is one that caches a single Oracle table in TimesTen. Caching multiple tables in a single cache group is more complex and requires that you understand additional Cache Connect to Oracle concepts.

In general, each Oracle table to be cached should have at least one primary or non-null unique index key. In addition, the primary keys and unique indexes defined in your TimesTen cache group tables should match those of the Oracle tables. For example, if the Oracle table has the primary key or unique index columns *C1*, *C2*, and *C3*, the corresponding TimesTen primary key on the cached table must also have columns *C1*, *C2*, and *C3*.

You can create any number of non-unique indexes for your TimesTen cache tables. Adding indexes to a cache table helps to speed up SQL in the same manner as any non-cache TimesTen table. Do not create unique indexes that do not match those on the Oracle tables, because they may cause false unique constraint failures.

Note: Oracle temporary tables cannot be cached.

This section includes the following topics:

- Defining a single cache group table
- Defining multiple cache group tables
- Caching Oracle partitioned tables
- About Oracle synonyms

Defining a single cache group table

Create a simple cache group definition to cache a single table as follows:

```
CREATE TYPE CACHE GROUP owner.name
FROM
  owner.root_table(
```

```
    column_list,
    PRIMARY KEY(primary_key_column_list));
```

where:

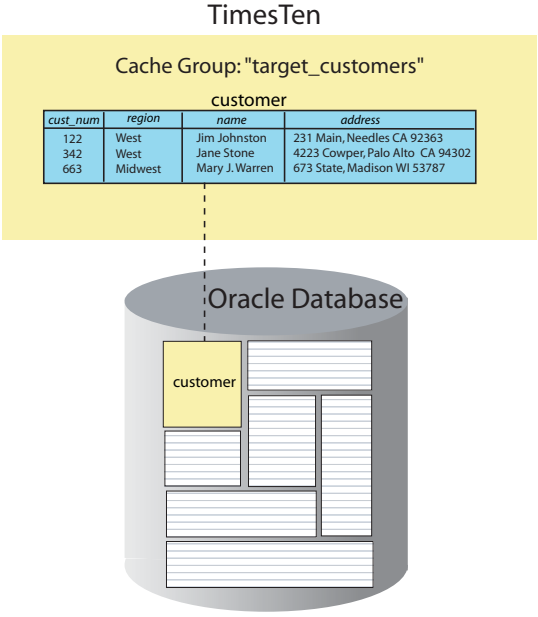
- *owner.root_table* is owner and table name of an Oracle table or a private synonym for an Oracle table.
- *column_list* is a list of columns and their data types for the table to be cached.
- *primary_key_column_list* is a list of columns that form the primary key.

When you choose data types for the columns, consider the data types in the Oracle columns and choose a properly mapped data type for the table in the cache group. See "Data type mappings for Cache Connect to Oracle" on page 70.

Example 3.3

Figure 3.6 shows a single-table READONLY cache group, named *target_customers*, to cache the *customer* Oracle table. The data in the TimesTen cache group is a subset of the much larger data set for all of the customers stored in the Oracle database

Figure 3.6 Caching a single Oracle table in TimesTen



The CREATE READONLY CACHE GROUP statement used to create the single-table *target_customers* cache group is:

```
CREATE READONLY CACHE GROUP target_customers
FROM
  user1.customer (
    cust_num   NUMBER NOT NULL PRIMARY KEY,
```

```
    region      VARCHAR2(5) NOT NULL,
    name        VARCHAR2(80),
    address     VARCHAR2(255) NOT NULL
  );
```

Defining multiple cache group tables

If multiple Oracle tables are cached in the same cache group, you must define a *root table* and one or more *child tables*. There can be only one root table in a cache group.

In a cache group with multiple tables, each child table in the cache group must be joined to the root table or another child table in the cache group by a foreign key constraint. Though the tables in a cache group must be joined by foreign key constraints in TimesTen, the tables do not necessarily need to be joined in the Oracle instance. The root table in the cache group does not refer to any other table in the cache group through a foreign key constraint. All other tables in the cache group are *child tables*.

Each table definition must include a primary key. Each child table must also include a foreign key reference to the primary key of its parent table. The table hierarchy in your cache group can designate child tables to be parents of other child tables, but no table in the cache group can be a child to more than one parent in the cache group. See Figure 3.7 for an example of a correct cache group table configuration. Figure 3.8 shows an incorrect table configuration and Figure 3.9 shows a work-around to the problem.

You must create a separate table definition in your cache group for each Oracle table to be included in the cache. The owner and name of the table definition in your cache group must match the owner and name of the Oracle table. You can create a table definition to cache all or a subset of the columns in the Oracle table.

Note: Each Oracle table can be defined in only one cache group. You cannot create multiple cache groups that cache the same Oracle table.

Example 3.4 Create a simple cache group definition to cache a root table and one child table as follows:

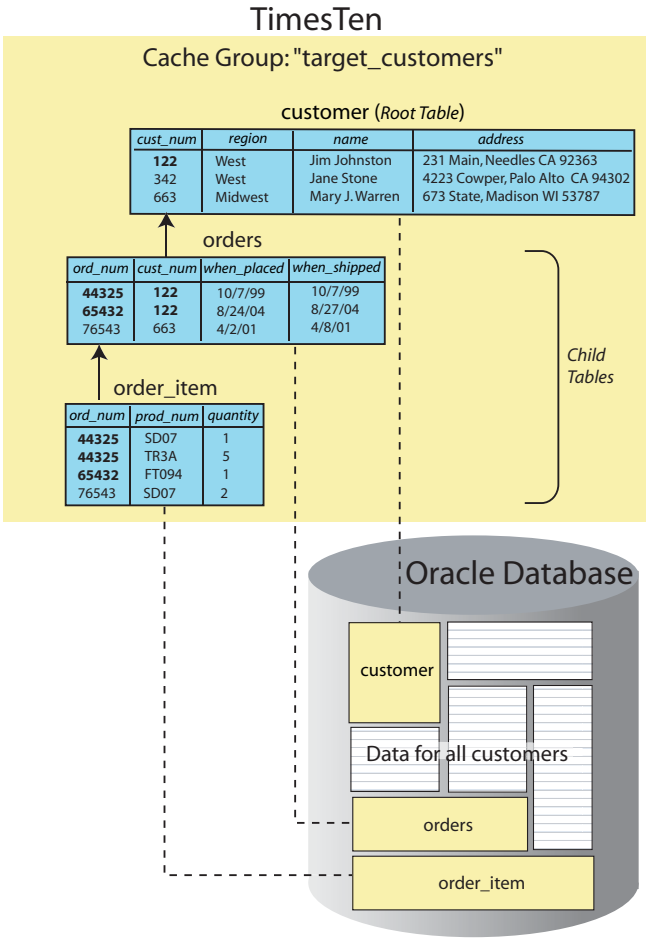
```
CREATE TYPE CACHE GROUP owner.name
FROM
  owner.root_table(
     column_list,
     PRIMARY KEY(primary_key_column_list)),
  owner.child_table(
     column_list,
     PRIMARY KEY(primary_key_column_list),
     FOREIGN KEY(reference_column_list)
        REFERENCES owner.root_table(primary_key_column_list));
```

where:

- *owner.root_table* is owner and table name of an Oracle table or a private synonym for an Oracle table.
- *owner.child_table* is owner and table name of an Oracle table or a private synonym for an Oracle table.
- *column_list* is a list of columns in the table to be cached.
- *primary_key_column_list* is a list of columns that form the primary key.
- *reference_column_list* is a list of columns in *child_table* that reference a foreign key.

Figure 3.7 shows a multi-table version of the *target_customers* cache group, previously shown in Figure 3.6.

Figure 3.7 Cache Group with Multiple Tables



In this example, the cache group has been expanded to cache three Oracle tables: *customer*, *orders*, and *order_item*. The data in the cache group is a subset of the much larger data set for all of the customers stored in the Oracle database. Each parent table within the *target_customers* cache group has a primary key (shown in **bold**) that is referenced by a child table through a foreign key relationship (shown by arrows). The *customer* table is the root table because it does not refer to any another table in the cache group. The primary key of the *customer* table is the primary key for the *target_customers* cache group. The *orders* and *order_item* tables are child tables.

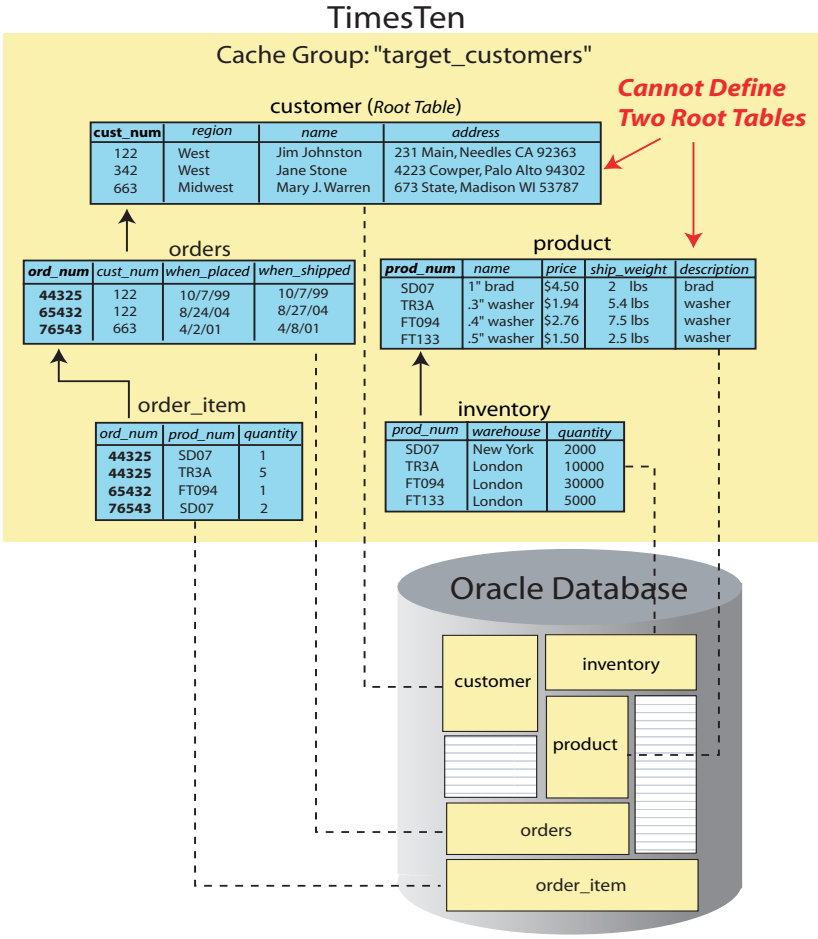
Example 3.5

Use the CREATE READONLY CACHE GROUP statement used to create the multitable *target_customers* cache group:

```
CREATE READONLY CACHE GROUP target_customers
FROM
user1.customer (
  cust_num  NUMBER NOT NULL PRIMARY KEY,
  region    VARCHAR2(10) NOT NULL,
  name      VARCHAR2(80),
  address   VARCHAR2(255) NOT NULL
),
user1.orders (
  ord_num       NUMBER NOT NULL PRIMARY KEY,
  cust_num      NUMBER NOT NULL,
  when_placed   TIMESTAMP NOT NULL,
  when_shipped  TIMESTAMP,
  FOREIGN KEY (cust_num) REFERENCES user1.customer (cust_num)
),
user1.order_item (
  ord_num   NUMBER NOT NULL,
  prod_num  NUMBER NOT NULL,
  quantity  NUMBER NOT NULL,
  PRIMARY KEY (ord_num, prod_num),
  FOREIGN KEY (ord_num) REFERENCES user1.orders (ord_num)
);
```

Figure 3.8 shows an example of an *incorrect* table configuration in a cache group.

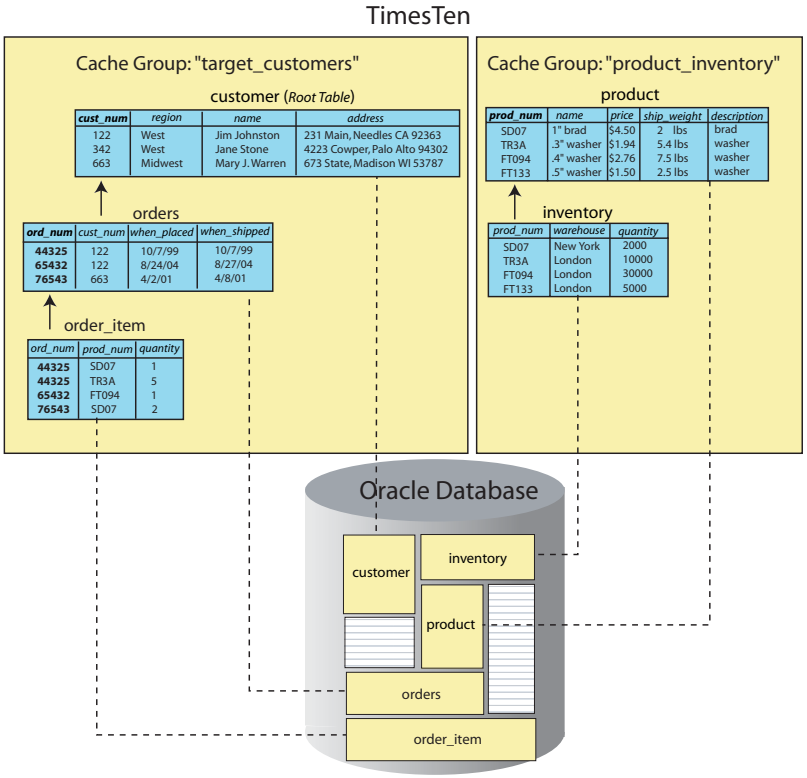
Figure 3.8 Problem: Two Root Tables



You cannot define the *products* and *inventory* tables in the same cache group as the *customer*, *orders*, and *order_item* tables. This is because the *products* table does not have a foreign key reference (direct or indirect) to the root table, *customers*. This means that the *products* table is also considered a root table, which is invalid because a cache group cannot have more than one root table.

To cache all of the tables, you can create a second cache group for the *products* and *inventory* tables, as shown in Figure 3.9.

Figure 3.9 Solution: Two Separate Cache Groups



Caching Oracle partitioned tables

You can define Oracle partitioned tables in a cache group by following the same conventions as you would when caching regular Oracle tables. For example, a partitioned table in a read-only cache group must have a corresponding unique, non-null index in Oracle. The partitioned table used in the cache group can be partitioned or subpartitioned in any manner (hash, list, range or composite).

The following restrictions apply when caching Oracle partitioned tables:

- DDL operations on partitions do not affect the cache group unless there is data loss. For example, if a partition with data is truncated, AUTOREFRESH does not delete the data from the corresponding cached table.
- WHERE clauses in cache group operations cannot reference individual partitions or subpartitions. For example, an attempt to define the following partitioned table, *user1.partitioned_table*, returns an error:

```
CREATE READONLY CACHE GROUP badcachegroup
FROM
```

```
     user1.partitioned_table(ii NUMBER NOT NULL PRIMARY KEY, jj
NUMBER)
    WHERE ii IN (SELECT ii
                 FROM user1.partitioned_table
                 PARTITION(F200402));
```

Note: Attempted cache operations (such as LOAD CACHE GROUP, UNLOAD CACHE GROUP, REFRESH CACHE GROUP) on an offline partition result in an ORA-00376 or ORA-01110 error. However, if the partition goes offline (for example, during a backup operation), then no error is generated unless you attempt to access the partition.

About Oracle synonyms

A table definition in a cache group can reference a private Oracle synonym for an Oracle base table. The actual Oracle base table can exist in another Oracle schema and have a different name, but it must reside on the same Oracle server as the synonym. The table name in the cache group must be the name of a private synonym, but the synonym can point to another synonym, which can be either a public or private synonym. The synonym must eventually point (either directly or indirectly) to a table, a partitioned table, or a materialized view.

A cache group that contains a table definition for an Oracle synonym can be a USERMANAGED, SWT, or AWT cache group. The USERMANAGED cache group can support the LOAD CACHE GROUP, UNLOAD CACHE GROUP, REFRESH CACHE GROUP, and FLUSH CACHE GROUP operations, but the cache group cannot be configured with AUTOREFRESH or READONLY attributes. You cannot have table definitions for Oracle synonyms in a READONLY cache group.

Using WHERE clauses

The table definitions in the CREATE CACHE GROUP statements for user-managed and READONLY cache groups can include a WHERE clause to specify search conditions when copying Oracle data into the cache.

In addition, you can specify WHERE clauses in LOAD CACHE GROUP, UNLOAD CACHE GROUP, MERGE and FLUSH CACHE GROUP statements. For more information about using these statements, see Chapter 5, “Using Cache Groups.” 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 group is evaluated before the WHERE clause in the statement. All WHERE clauses are parsed by TimesTen. Do not use Oracle SQL syntax that is not supported by TimesTen.

Example 3.6 This example creates a *western_customers* cache group, specifying a WHERE clause to cache the data related to only the customers having zip codes in the western region from the *customer* table on Oracle:

```
CREATE USERMANAGED CACHE GROUP western_customers
    FROM
    user1.customer (custid INTEGER NOT NULL,
        name VARCHAR(2100) NOT NULL,
        addr VARCHAR2(100),
        zip VARCHAR2(10),
        region VARCHAR2(10),
        PRIMARY KEY(custid),
        PROPAGATE)
    WHERE (user1.customer.region = 'Western');
```

We then specify an additional WHERE clause in the LOAD CACHE GROUP statement to cache only the customers from the western region with IDs of 100 or less:

```
LOAD CACHE GROUP western_customers WHERE (custid <= 100)
  COMMIT EVERY 256 ROWS;
```

The WHERE clauses in both the CREATE CACHE GROUP and LOAD CACHE GROUP statements are evaluated on Oracle before the data is loaded into the cache group.

Table 3.4 shows whether the WHERE clause is evaluated on TimesTen, Oracle, or both for the LOAD CACHE GROUP, UNLOAD CACHE GROUP, REFRESH CACHE GROUP and FLUSH CACHE GROUP statements. If the CREATE CACHE GROUP statement also includes a WHERE clause, the table shows when this additional WHERE clause is evaluated on Oracle.

Table 3.4 WHERE clause evaluation

SQL Statement	Statement WHERE clauses evaluated on...	CREATE CACHE GROUP WHERE clauses evaluated on...
LOAD	Oracle	Oracle
UNLOAD	TimesTen	-
REFRESH	TimesTen and Oracle	Oracle
FLUSH	TimesTen	-

Table 3.5 shows whether the WHERE clauses are parsed by TimesTen only, or both TimesTen and Oracle for each SQL statement.

Table 3.5 WHERE clause parsing

SQL Statement	Parsed by TimesTen	Parsed by Oracle
LOAD	Yes	Yes
UNLOAD	Yes	No
REFRESH	Yes	Yes
FLUSH	Yes	No
CREATE	Yes	Yes

With verbose logging, all of the SQL statements that are executed against Oracle will be logged in the Oracle server log. Look at this log to better understand the mechanics and performance of your cache group operations. See "Using the logs generated by the TimesTen daemon" in *Oracle TimesTen In-Memory Database Troubleshooting Procedures Guide*.

The following restrictions apply to WHERE clauses used with Cache Connect to Oracle:

- WHERE clauses in a CREATE CACHE GROUP statement cannot specify subqueries, so they cannot reference any table other the current table. WHERE clauses in LOAD CACHE GROUP, UNLOAD CACHE GROUP, REFRESH CACHE GROUP and FLUSH CACHE GROUP statements may specify subqueries.
- WHERE clauses cannot include Oracle PARTITION extended names. See "Caching Oracle partitioned tables" on page 59 for details.
- WHERE clauses in LOAD CACHE GROUP, REFRESH CACHE GROUP, FLUSH CACHE GROUP statements (for both tables and synonyms) can reference only the root table, unless the WHERE clause contains a subquery.
- When you create a cache group with more than one table, column names in all WHERE clauses must be fully qualified as follows: `user.table.column`.

Locale neutrality in WHERE clauses

Operations between a cache group and the Oracle Database inherit the values of TimesTen globalization support connection attributes used in the session. These operations include passthrough, propagate, load, refresh, flush, and synchronous writethrough.

Cache group operations that are not directly associated with a specific user session use the default settings of the globalization support connection attributes. These operations include autorefresh, aging, and asynchronous writethrough. For example, these operations use BINARY as the value for NLS_SORT.

Inconsistencies can result if cache group operations use different values for globalization support connection attributes. For example, consider the following statement:

```
CREATE CACHE GROUP cachegroup1 FROM table1(
     column1 NUMBER NOT NULL PRIMARY KEY,
     column2 NCHAR(30))
WHERE column2 < 'string';
```

Character strings in the WHERE clause are compared by using the binary values of the characters. This may produce unintended results in populating the cache group table, depending on the locale.

To ensure that character strings in the WHERE clause are compared by using their linguistic values, use a CREATE CACHE GROUP statement with a WHERE clause similar to the following:

```
CREATE CACHE GROUP cachegroup1 FROM table1(
     column1 NUMBER NOT NULL PRIMARY KEY,
     column2 NCHAR(30))
WHERE
     NLSSORT(column2,'NLS_SORT=TCHINESE_RADICAL')
   < NLSSORT('string','NLS_SORT=TCHINESE_RADICAL');
```

Implementing aging in a cache group

You can define an aging policy for the root table of a cache group. An aging policy refers to the type of aging and the aging attributes, as well as the aging state (ON or OFF). You can specify one of the following types of aging: usage-based or time-based. Usage-based aging removes least recently used (LRU) data within a specified data store usage range. Time-based aging removes data based on the specified data lifetime and frequency of the aging process. You can define both usage-based and time-based aging in the same data store, but you can define only one type of aging on a specific cache group or table.

Use the table definition in the CREATE CACHE GROUP statement to specify an aging policy for the root table in a new cache group. You can add an aging policy to the root table in an existing cache group by using the ALTER TABLE statement if the table does not already have an aging policy defined. You can change the aging policy by dropping aging and adding a new aging policy.

Aging can be defined only on the root table of a cache group because aging is based on the cache instance.

You can also define aging on tables that are not in cache groups. For more information about aging on tables that are not in cache groups, see "Implementing aging in your tables" in *Oracle TimesTen In-Memory Database Operations Guide*.

This section includes the following topics:

- Usage-based aging
- Time-based aging
- Aging and foreign keys
- Scheduling when aging starts
- Configuring a sliding window

Usage-based aging

Usage-based aging enables you to maintain the amount of memory used in a data store within a specified threshold by removing the least recently used (LRU) data. LRU aging can be used with all types of cache groups except cache groups with the AUTOREFRESH attribute.

Define LRU aging for a cache group by using the AGING LRU clause of the table definition of the CREATE CACHE GROUP statement. Aging begins automatically if the state is ON. Aging begins immediately if the root table is the first table in the data store to have an LRU aging policy defined. Otherwise, aging on the root table occurs at the same time that aging occurs on the tables that already have an LRU aging policy defined.

Use the **ttAgingLRUConfig** built-in procedure to specify the LRU aging attributes for all tables in the data store. The attribute values apply to all tables in

the data store that have an LRU aging policy. If you do not call the **ttAgingLRUConfig** built-in procedure, then the default values for the attributes are used.

The following table summarizes the LRU aging attributes:

LRU Aging Attribute	Description
LowUsageThreshhold	The percent of the data store **PermSize** at which LRU aging is deactivated.
HighUsageThreshhold	The percent of the data store **PermSize** at which LRU aging is activated.
AgingCycle	The number of minutes between aging cycles.

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

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

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

Use the ALTER TABLE statement to perform the following tasks:

- Enable or disable the aging state by using the ALTER TABLE statement with the SET AGING {ON|OFF} clause on the root table of the cache group.
- Add an LRU aging policy to an existing root table by using the ALTER TABLE statement with the ADD AGING LRU [ON|OFF] clause.
- Drop aging on the root table by using the ALTER TABLE statement with the DROP AGING clause.

Use the **ttAgingScheduleNow** built-in procedure to schedule when aging starts. For more information, see "Scheduling when aging starts" on page 67.

To change aging from LRU to time-based for a cache group, first drop aging on the root table by using the ALTER TABLE statement with the DROP AGING clause. Then add time-based aging to the root table by using the ALTER TABLE statement with the ADD AGING USE clause.

Time-based aging

Time-based aging removes data from the root table of the cache group based on the specified data lifetime and frequency of the aging process. Define time-based aging in the AGING USE clause of the table definition of the CREATE CACHE GROUP statement. Add a time-based aging policy to the root table of an existing cache group with the AGING USE clause of the ALTER TABLE statement.

The AGING USE clause has a *ColumnName* argument.*ColumnName* is the name of the column that is used for time-based aging. For brevity, we call this column the *timestamp column*. The timestamp column must be defined as follows:

- TIMESTAMP or DATE data type
- NOT NULL

Your application updates the values of the timestamp column. If the value of this column is unknown for some rows and you do not want the rows to be aged, then define the column with a large default value. You can create an index on the timestamp column for better performance of the aging process.

Note: You cannot add or modify a column in an existing table and then use that column as a timestamp column because you cannot add or modify a column and define it to be NOT NULL.

You cannot drop the timestamp column from a table that has a time-based aging policy.

Specify the lifetime in days, hours, or minutes in the LIFETIME clause of the CREATE CACHE GROUP statement.

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

Use the CYCLE clause to indicate how often the system should examine the rows to remove data that has exceeded the specified lifetime. If you do not specify CYCLE, aging occurs every five minutes. If you specify 0 for the cycle, then aging is continuous. Aging begins automatically if the state is ON.

Use the ALTER TABLE statement to perform the following tasks:

- Enable or disable the aging state on a root table with a time-based aging policy by using the SET AGING {ON|OFF} clause.
- Change the aging cycle on a root table with a time-based aging policy by using the SET AGING CYCLE clause.
- Change the lifetime in a root table by using the SET AGING LIFETIME clause.

- Add time-based aging to an existing root table with no aging policy by using the ADD AGING USE clause.
- Drop aging on the root table by using the DROP AGING clause.

Use the **ttAgingScheduleNow** built-in procedure to schedule when aging starts. For more information, see "Scheduling when aging starts" on page 67.

To change aging from time-based to LRU aging for a cache group, first drop aging on the root table. Then add LRU aging to the root table by using the ALTER TABLE statement with the ADD AGING LRU clause.

Note: You must stop the cache agent if you want to alter the root table of an AUTOREFRESH cache group. Stop the cache agent before you add, alter or drop aging on an AUTOREFRESH cache group.

Aging and foreign keys

Tables that are related by foreign keys must have the same aging policy.

If LRU aging is in effect and a row in a child table has been recently accessed, then neither the parent row nor the child row will be deleted.

If time-based aging is in effect and a row in a parent table is a candidate for aging out, then the parent row and all of its children will be deleted.

If a table has ON DELETE CASCADE enabled, the setting is ignored.

Scheduling when aging starts

Use the **ttAgingScheduleNow** built-in procedure to schedule the aging process. The aging process starts as soon as you call the procedure unless there is already an aging process in progress, in which case it will begin when the aging process that is in progress has completed.

When you call **ttAgingScheduleNow**, the aging process starts regardless of whether the state is ON or OFF. Specify the name of the root table when you call **ttAgingScheduleNow**. Otherwise **ttAgingScheduleNow** will start or reset aging on all of the tables in the data store that have aging defined, not just the root table of the cache group.

The aging process starts only once as a result of calling **ttAgingScheduleNow**. Calling **ttAgingScheduleNow** does not change the aging state. If the aging state is OFF when you call **ttAgingScheduleNow**, then the aging process starts, but it does not continue after the process is complete. To continue aging, you must call **ttAgingScheduleNow** again or change the aging state to ON.

If the aging state is already set to ON, then **ttAgingScheduleNow** resets the aging cycle based on the time **ttAgingScheduleNow** was called.

You can control aging externally. First disable aging by using the ALTER TABLE statement with the SET AGING OFF clause to disable aging. Then use **ttAgingScheduleNow** to start aging at the desired time.

You can use **ttAgingScheduleNow** to start or reset aging for an individual table by specifying its name when you call the procedure. If you do not specify a table name, then **ttAgingScheduleNow** will start or reset aging on all of the tables in the data store that have aging defined.

You can monitor aging by using the **ttTraceMon** utility. See "AGING tracing" in *Oracle TimesTen In-Memory Database Troubleshooting Procedures Guide*.

Configuring a sliding window

You can use time-based aging to achieve a sliding window for cache group data. In a cache group with a sliding window, new data is added and old data is removed on a regular schedule so that the cache contains only the data that satisfies a specific time interval.

You can configure a sliding window for cache group data by using the incremental AUTOREFRESH attribute and specifying a time-based aging policy. The AUTOREFRESH operation checks the timestamp on the Oracle table to determine whether new data should be brought into the cached table. SYSDATE and the time zone must be identical on Oracle and TimesTen.

You can configure a sliding window for other types of cache groups by using manual load, transparent load, refresh or inserts to bring new data into the cache group tables.

Example 3.7 The following statement creates a cache group with sliding window properties.

```
CREATE READONLY CACHE GROUP cg1
  AUTOREFRESH STATE ON INTERVAL 1 MINUTES
  FROM t1(i NUMBER NOT NULL PRIMARY KEY,
          ts TIMESTAMP NOT NULL,
          c VARCHAR2(50))
  AGING USE ts LIFETIME 3 HOURS CYCLE 10 MINUTES;
```

The *cg1* cache group has an autorefresh interval of 1 minute. Thus new data in the Oracle table is brought into the cache group as often as every minute.

Aging checks for old data every 10 minutes. Aging removes data that is more than 3 hours old (ts < SYSDATE - 3 HOURS).

The parameters that you set for the AUTOREFRESH interval and the LIFETIME interval determine how long the data remains in the cache. It is possible for data to be aged out of the cache in less than the full lifetime interval. For example, if the AUTOREFRESH interval is 3 days and the LIFETIME interval is 30 days, data may already be 3 days old when it enters the cache. Such data will be

removed from the cache after 27 days. This occurs because aging is based on the Oracle timestamp, not the TimesTen timestamp.

Working with data types and type modes

You must use Oracle type mode (**TypeMode**=0) if you are using the Cache Connect feature of Oracle TimesTen In-Memory Database, even if you have cache groups that were defined before release 7.0.

If you have cache groups that were defined before release 7.0, use the **ttMigrate** utility with the `-convertCGTypes` option to map the data types to the data types used in release 7.0 and later.

Note that if your application uses native integer data types and you want to continue using them, you must change the application to specify TT_INTEGER, TT_SMALLINT, TT_TINYINT, and TT_BIGINT before you use **ttMigrate**.

For more information about data type mapping in release 7.0 and later, see:

- "Data type mappings for Cache Connect to Oracle" on page 70
- "Data Types" in *Oracle TimesTen In-Memory Database SQL Reference Guide*
- "ttMigrate" in *Oracle TimesTen In-Memory Database API Reference Guide*

Floating point data types

BINARY_FLOAT and BINARY_DOUBLE were introduced in Oracle Database 10g Release 1. Cache Connect supports Oracle clients and servers from Oracle9i and Oracle Database 10g.

TimesTen recommends the following configuration if you want to use BINARY_FLOAT or BINARY_DOUBLE data types:

- Use an Oracle Database 10g client and server
- Map BINARY_FLOAT data in TimesTen to BINARY_FLOAT data in Oracle
- Map BINARY_DOUBLE data in TimesTen to BINARY_DOUBLE data in Oracle

You can use FLOAT(*n*) with Oracle9i and Oracle Database 10g.

If you want to conserve memory or enhance performance by mapping the TT_BINARY_FLOAT data type to Oracle's FLOAT(*n*), TimesTen recommends that your Oracle client version be the same or later release as your Oracle server version. Even if you follow this recommendation, the following behavior can occur:

- Data precision can be lost during conversion from FLOAT(*n*) to BINARY_FLOAT and BINARY_DOUBLE.

- When Inf and NaN are defined as BINARY_FLOAT and BINARY_DOUBLE data types in the cache, Inf can be interpreted as an overflow and NaN can be interpreted as 0.

For more information about mapping data types, see "Data type mappings for Cache Connect to Oracle" on page 70.

Data type mappings for Cache Connect to Oracle

When you choose data types for columns in cache group tables, consider the data types in the Oracle columns and choose an equivalent data type for tables in the cache group.

Primary and foreign key columns are distinguished from the non-key columns. The data type mappings allowed for key columns in a cache group table are shown in Table 3.6.

Table 3.6 Data type mappings allowed in key columns

Oracle Data Type	TimesTen Data Type
NUMBER(*p*,*s*)	NUMBER(*p*,*s*) **Note:** DECIMAL(*p*,*s*) or NUMERIC(*p*,*s*) can also be used. They are aliases for NUMBER(*p*,*s*).
NUMBER(*p*,0) INTEGER	TT_TINYINT TT_SMALLINT TT_INTEGER TT_BIGINT INTEGER NUMBER(*p*,0)
NUMBER	TT_TINYINT TT_SMALLINT TT_INTEGER TT_BIGINT NUMBER
CHAR(*m*)	CHAR(*m*)
VARCHAR2(*m*)	VARCHAR2(*m*)
RAW(*m*)	VARBINARY(*m*)
TIMESTAMP(*m*)	TIMESTAMP(*m*)
DATE	DATE

Oracle Data Type	TimesTen Data Type
NCHAR(*m*)	NCHAR(*m*)
NVARCHAR2(*m*)	NVARCHAR2(*m*)

Table 3.7 shows the data type mappings for non-key columns in a cache group table.

Table 3.7 Data type mappings allowed for non-key columns

Oracle Data Type	TimesTen Data Type
NUMBER(*p*,*s*)	NUMBER(*p*,*s*) REAL FLOAT BINARY_FLOAT DOUBLE BINARY_DOUBLE
NUMBER(*p*,0) INTEGER	TT_TINYINT TT_SMALLINT TT_INTEGER TT_BIGINT INTEGER NUMBER(*p*,0) FLOAT BINARY_FLOAT DOUBLE BINARY_DOUBLE
NUMBER	TT_TINYINT TT_SMALLINT TT_INTEGER TT_BIGINT NUMBER REAL FLOAT BINARY_FLOAT DOUBLE BINARY_DOUBLE
CHAR(*m*)	CHAR(*m*)
VARCHAR2(*m*)	VARCHAR2(*m*)

Oracle Data Type	TimesTen Data Type
RAW(*m*)	VARBINARY(*m*)
LONG	VARCHAR2(*m*) **Note:** *m* can be any valid value within the range defined for the data type.
LONG RAW	VARBINARY(*m*) **Note:** *m* can be any valid value within the range defined for the data type.
TIMESTAMP(*m*)	TIMESTAMP(*m*)
DATE	DATE TIMESTAMP(0)
FLOAT(*n*) **Note:** Includes DOUBLE PRECISION and FLOAT, which are equivalent to FLOAT(126). Also includes REAL, which is equivalent to FLOAT(63).	FLOAT(*n*) BINARY_DOUBLE **Note:** FLOAT(126) can be declared as DOUBLE PRECISION. FLOAT(63) can be declared as REAL.
BINARY_FLOAT	BINARY_FLOAT
BINARY_DOUBLE	BINARY_DOUBLE
NCHAR(*m*)	NCHAR(*m*)
NVARCHAR2(*m*)	NVARCHAR2(*m*)

See Chapter 1, "Data Types" in *Oracle TimesTen In-Memory Database SQL Reference Guide* for complete information about TimesTen data types.

4

Managing Cache Groups

This chapter describes how to manage cache groups. It includes the following topics:

- Configuring your system for cache groups
- Set-up tasks on the Oracle database
- Defining DSNs for cached tables
- Starting and stopping the cache agent
- Checking the status of the cache agent
- Starting the replication agent for an AWT cache group
- Applying a cache group definition to a data store
- Setting a passthrough level
- Monitoring cache groups
- Managing Oracle objects for READONLY, AUTOREFRESH, and AWT cache groups

Configuring your system for cache groups

The Cache Connect to Oracle installation and configuration information is described in *Oracle TimesTen In-Memory Database Installation Guide*.

This section summarizes the steps specific to configuring your TimesTen host system as an Oracle Client and to use the Cache Connect to Oracle features to operate with a remote Oracle server.

1. Install the Oracle client or Oracle database on your TimesTen host, as described in "Install the Oracle Client on the TimesTen host" on page 15. The Oracle client or database must be installed before TimesTen to prevent problems with Cache Connect features.
2. Configure the environment variables for your particular operating system, as described in "Configuring Cache Connect to Oracle on a UNIX platform" on page 75 or "Configuring Cache Connect to Oracle on a Windows platform" on page 75.
3. Install TimesTen as described in the *Oracle TimesTen In-Memory Database Installation Guide*.

4. If you are going to use the Web-based Cache Administrator, configure the embedded web server as described in "Web server configuration" in the *Oracle TimesTen In-Memory Database Installation Guide*.
5. If you define the Oracle Service Names in a TNSNAMES.ORA file, use the system-level TNSNAMES.ORA file. The TimesTen main daemon, the cache agent, the web server, and the replication agent use information provided in the system-level TNSNAMES.ORA file. Do not configure the Oracle client to use a different TNSNAMES.ORA file.

Note: TimesTen does not support Oracle Name Server for Windows clients.

Configuring Cache Connect to Oracle on a UNIX platform

Set these environment variables for the TimesTen user environment and for the environment of the user that starts the TimesTen daemon (root):

- Set the ORACLE_HOME environment variable to the path of the Oracle Client installation directory. For example:

  ```
  $ORACLE_HOME = /oracle/ora10g
  ```

  For details, see "ORACLE_HOME environment variable" in the *Oracle TimesTen In-Memory Database Installation Guide*.

- The LD_LIBRARY_PATH or SHLIB_PATH environment variable for 32- and 64-bit Oracle/TimesTen installations should include:

  ```
  $ORACLE_HOME/lib
  $ORACLE_HOME/network/lib
  install_dir/lib
  ```

  For example:

  ```
  LD_LIBRARY_PATH = $ORACLE_HOME/lib:$ORACLE_HOME/network/lib:
  /timesten/myinstance/lib
  ```

Note: If you are configuring a 64-bit Oracle server with a 32-bit TimesTen installation, then the library path is `$ORACLE_HOME/lib32`.

  For details, see "Shared library path environment variable" in *Oracle TimesTen In-Memory Database Installation Guide*.

- The PATH environment variable should include:

  ```
  $ORACLE_HOME/bin
  install_dir/bin
  ```

  For example:

  ```
  PATH = $ORACLE_HOME/bin:/timesten/myinstance/bin
  ```

Configuring Cache Connect to Oracle on a Windows platform

You must set the PATH system environment variable to include:

```
Oracle_install_dir\bin
install_dir\lib
install_dir\bin
```

For example:

```
PATH = C:\Oracle\Ora10g\bin;C:\timesten\myinstance\lib;
C:\timesten\myinstance\bin;
```

Set-up tasks on the Oracle database

This section describes the operations an Oracle DBA must perform on the Oracle database using the *system* account. The topics include:

- Create Oracle users and set privileges
- Create a separate tablespace for the cache administration user

Create Oracle users and set privileges

As described in "Defining DSNs for cached tables" on page 81, you must specify an Oracle user and password in order to access Oracle tables and create cache groups. All Oracle user accounts must be granted CREATE SESSION privilege to use Cache Connect to Oracle. Some Cache Connect to Oracle operations require additional privileges for Oracle users.

Some Cache Connect to Oracle operations require a separate user with additional Oracle privileges. Because you may want to grant these additional cache administration user privileges only to select users, Cache Connect to Oracle allows you to create a separate cache administration user account. (See "Starting and stopping the cache agent" on page 82 for details on how to start a cache agent as a cache administration user.)

The minimum privileges for both Oracle users and the cache administration user to use each Cache Connect to Oracle operation are listed in Table 4.1. The privileges for the cache administration user must include all Oracle tables to be cached in the data store, while the Oracle user needs only privileges for the specific Oracle tables specified in the cache group created by the user. See Example 4.1 on page 79.

Table 4.1 Oracle privileges required for cache group operations

Cache group operation	Minimum privileges to be granted to Cache Connect to Oracle users	Minimum privileges to be granted to the cache administration user
All operations	• CREATE SESSION on the Oracle database • SELECT TABLE	None necessary
CREATE READONLY CACHE GROUP	• CREATE SESSION on the Oracle database • SELECT TABLE	• CREATE SESSION on the Oracle database • SELECT TABLE • UNLIMITED TABLESPACE (or adequate space quota on the tablespace) • CREATE TABLE[1] • CREATE ANY TRIGGER[1]
CREATE SYNCHRONOUS WRITETHROUGH CACHE GROUP	• CREATE SESSION on the Oracle database • SELECT TABLE • INSERT • UPDATE • DELETE	None necessary
CREATE ASYNCHRONOUS WRITETHROUGH CACHE GROUP	• CREATE SESSION on the Oracle database • SELECT TABLE • INSERT [2] • UPDATE [2] • DELETE [2] • QUERY REWRITE [3]	• CREATE SESSION on the Oracle database • SELECT TABLE • INSERT • UPDATE • DELETE • UNLIMITED TABLESPACE (or adequate space quota on the tablespace) • CREATE TABLE[1]

Cache group operation	Minimum privileges to be granted to Cache Connect to Oracle users	Minimum privileges to be granted to the cache administration user
CREATE USERMANAGED CACHE GROUP (see variants in subsequent rows)	• CREATE SESSION on the Oracle database • SELECT TABLE	None necessary
CREATE USERMANAGED CACHE GROUP .....with PROPAGATE	• CREATE SESSION on the Oracle database • SELECT TABLE • INSERT • UPDATE • DELETE	None necessary
CREATE USERMANAGED CACHE GROUP .....with AUTOREFRESH INCREMENTAL	• CREATE SESSION on the Oracle database • SELECT TABLE • QUERY REWRITE [3]	• CREATE SESSION on the Oracle database • SELECT TABLE • UNLIMITED TABLESPACE (or adequate space quota on the tablespace) • CREATE TABLE[1] • CREATE ANY TRIGGER[1]
LOAD CACHE GROUP	• CREATE SESSION on the Oracle database • SELECT TABLE	• CREATE SESSION on the Oracle database • SELECT TABLE
REFRESH CACHE GROUP	• CREATE SESSION on the Oracle database • SELECT TABLE	• CREATE SESSION on the Oracle database • SELECT TABLE

Cache group operation	Minimum privileges to be granted to Cache Connect to Oracle users	Minimum privileges to be granted to the cache administration user
FLUSH CACHE GROUP	• CREATE SESSION on Oracle • SELECT TABLE • INSERT TABLE • UPDATE TABLE	-
All operations in a configuration with Real Application Clusters	• CREATE SESSION on the Oracle database • SELECT TABLE	SELECT on Oracle GV$SESSION dynamic performance view

1 Not required to create the cache group if Oracle objects were manually installed. See **ttCacheSqlGet**.

2 Not enforced from TimesTen

3 Required only for Oracle server version 9.2

Example 4.1

You need to establish the privileges for two users, *Sam* and *John*, so each user can issue a CREATE READONLY CACHE GROUP statement to create separate read-only cache groups. *Sam* is to create a cache group to cache the Oracle table *TTUSER.TABLEA* and *John* is to cache Oracle *TTUSER.TABLEB* in another cache group.

To provide *Sam* and *John* with sufficient Oracle privileges to CREATE READONLY CACHE GROUPs, you need to create two Oracle user accounts and one cache administration user with extended privileges. The cache administration user is named *User2*:

- oracleUID = Sam
- oracleUID = John
- cacheUid = User2

Create the accounts on Oracle:

```
SQL> CREATE USER Sam IDENTIFIED BY Samspwd DEFAULT TABLESPACE users;
SQL> CREATE USER John IDENTIFIED BY Johnspwd DEFAULT TABLESPACE
users;
SQL> CREATE USER User2 IDENTIFIED BY User2pwd DEFAULT TABLESPACE
users;
```

Then assign each user the following privileges:

```
SQL> GRANT CREATE SESSION, SELECT ON TTUSER.TABLEA TO Sam;
SQL> GRANT CREATE SESSION, SELECT ON TTUSER.TABLEB TO John;
```

```
SQL> GRANT
> CREATE SESSION,
> SELECT ON TTUSER.TABLEA, TTUSER.TABLEB,
> UNLIMITED TABLESPACE,
> CREATE TABLE,
> CREATE ANY TRIGGER
> TO User2;
SQL>
```

Create a separate tablespace for the cache administration user

TimesTen strongly recommends creating a separate tablespace for the cache administration user. This table space is used as the cache administration user's default tablespace. The tablespace contains autorefresh triggers for each Oracle table, change log tables for each Oracle table, and other objects that TimesTen needs for each cache administration user. See "Managing Oracle objects for READONLY, AUTOREFRESH, and AWT cache groups" on page 95. If you do not specify a separate tablespace, these objects are placed in the Oracle system tablespace.

Specify the tablespace when you create the cache administration user on Oracle. You can also specify the tablespace after user creation with the DEFAULT TABLESPACE clause of the Oracle ALTER USER statement.

Example 4.2

Example 4.1 showed how to create and grant privileges to the cache administration user *User2*. To specify a separate tablespace called *cacheuser* for *User2*, enter the following statement on Oracle:

```
SQL> ALTER USER User2 IDENTIFIED BY User2pwd
     DEFAULT TABLESPACE cacheuser;
```

Change log tables for each of the cached Oracle tables reside in the cache administration user tablespace. For each update on an Oracle table, one row (a change log record) is inserted into the change log table for that Oracle table. The size of a change log record in bytes is as follows:

```
size of change log record = size of primary key on Oracle table + 250
```

The number of records in a change log table depends on the update rate on the Oracle table and on the autorefresh interval on TimesTen. Every 20 seconds, TimesTen removes change log records that have been applied to all data stores that cache the associated Oracle table.

When the cache administration user tablespace gets full

When the cache administration user tablespace gets full, the autorefresh trigger makes space for new change log records by deleting existing change log records.

This can cause an automatic full refresh for some tables on some TimesTen data stores.

Check for the following conditions if the tablespace is full:

- Is a cache group being created or is a data store being duplicated? Both of these operations temporarily stop clean-up operations on the change log table.
- Are the cache agents on all TimesTen data stores running? If a cache agent is not running, change log records accumulate.
- Has a data store been abandoned without dropping an autorefresh cache group in the data store? Change log records accumulate in this situation.

For more information, see "Diagnosing a full cache administration user tablespace" in *Oracle TimesTen In-Memory Database Troubleshooting Procedures Guide*.

Defining DSNs for cached tables

Data stores that cache Oracle data must be defined as System DSNs and not as User DSNs. If TimesTen caches are not defined as System DSNs, then the TimesTen Cache Administrator cannot work with them and **ttAdmin** cannot start the TimesTen cache agent. For information on creating DSNs, see Chapter 1, "Creating TimesTen Data Stores" in *Oracle TimesTen In-Memory Database Operations Guide*.

The DSN path name must be absolute. See "Starting and stopping the cache agent" on page 82 for problems associated with relative DSN path names.

When creating DSNs for cached Oracle tables, the DSN attributes must be set as follows:

- **Logging** must be enabled as disk-based.
- **LockLevel** must enable row-level locking. (Cache Connect to Oracle does not support database-level locking.)
- **Isolation** can be any isolation mode.
- **OracleID** must be set to the Oracle Service Name for Oracle 9i and later Oracle releases.
- **PassThrough** can be set to control which procedures and SQL statements are to be executed locally in the TimesTen cache or passed through to Oracle, as described in "Setting a passthrough level" on page 91.
- **DatabaseCharacterSet** must be the same as the Oracle database character set.
- **TypeMode** must be set to 0 (Oracle).

The following attributes can be specified as DSN attributes or as connection attributes:

- Oracle User ID identifies the Oracle user ID. This string is indirectly specified by setting the **UID** connection attribute. The corresponding DSN attribute on UNIX is UID. On Windows, the corresponding DSN attribute is User ID.
- **OraclePWD** identifies the password for the Oracle user.

The user name and password used to connect to Oracle are taken either from the attributes in the DSN definition described above or from the ODBC connection string. The values given in the connection string override those specified in the DSN.

For example, you can specify the Oracle UID and **OraclePWD** in the **ttIsql** connect string:

```
ttIsql -connStr "DSN=cgDSN; UID=testuser; OraclePWD=mypsswrd"
```

If you are issuing updates and commits through an ODBC application, you must specify the user name and password in the connect string. For example:

```
SQLDriverConnect(hdbc, ..., "DSN=cgDSN; UID=testuser;
OraclePWD=mypsswrd", ...);
```

UNIX

For example, the DSN definition for *myOraCache* for a UNIX platform might look like the following:

```
[myOraCache]
DataStore=/users/OracleCache/hotData
Authenticate=0
Logging=1
LockLevel=0
PermSize=40
TypeMode=0
DurableCommits=1
OracleID=system1
UID=testuser
OraclePWD=mypsswrd
DatabaseCharacterSet=WE8ISO1559P1
```

WINDOWS

On Windows, specify the *OracleID* in the *Oracle Data Loading* screen of the ODBC TimesTen Setup dialog.

Starting and stopping the cache agent

A TimesTen process called a *cache agent* performs asynchronous cache operations such as loading or refreshing a cache group. Many Cache Connect to Oracle operations can be done directly by TimesTen without the assistance of a cache agent.

You must start a *separate* cache agent for each data store that contains a cache group if one or more of the following is true:

- The cache group is READONLY.

- The cache group is USERMANAGED and includes an AUTOREFRESH clause.
- The application loads or refreshes the cache group.

Note: On UNIX platforms, the cache agent requires that the ORACLE_HOME and the LD_LIBRARY_PATH or SHLIB_PATH environment variables be set as described in "Configuring Cache Connect to Oracle on a UNIX platform" on page 75.

You can start the cache agent for a DSN from either the command line or a program, as described in the following sections:

- Controlling the cache agent from the command line
- Controlling the cache agent from a program

You can also start a cache agent from a browser using the Cache Administrator. See Chapter 6, "Cache Administrator."

Note: If TimesTen was installed with Access Control enabled, you must have ADMIN privileges to the data store to start or stop the cache agent. In addition, the TimesTen user name must match the Oracle user name (this applies to both internal and external users). See Chapter 1, "Access Control" in *Oracle TimesTen In-Memory Database Installation Guide* for details.

Controlling the cache agent from the command line

Use the **ttAdmin** utility to control the cache agent from the command line.You can use **ttAdmin** to:

- Set the cache administration user ID and password from the command line
- Start the cache agent from the command line
- Stop the cache agent from the command line
- Set the cache agent start policy from the command line

For more information about using the **ttAdmin** utility, see *Oracle TimesTen In-Memory Database API Reference Guide*.

Set the cache administration user ID and password from the command line

You must have a cache administration user account that has the Oracle privileges described in "Create Oracle users and set privileges" on page 76 if you are using one of the following types of cache groups:

- ASYNCHRONOUS WRITETHROUGH cache group
- READONLY cache group
- USERMANAGED cache group with the AUTOREFRESH attribute

The cache administration user ID must conform to the following rules:

- Its length is from 1 to 30 bytes.
- It must begin with an alphabetic character.
- It can contain only ASCII alphanumeric characters, underscores (_), dollar signs ($) and pound signs (#).

The cache administration password must conform to the following rules:

- Its length is from 1 to 30 bytes long.
- It can contain any ASCII character except the semicolon (;).

Set the cache administration user ID and password before you start the cache agent. Use the following syntax:

```
ttAdmin -cacheUidPwdSet -cacheUid cacheUid -cachePwd cachePwd DSN
```

Note: The cache administration user ID and password cannot be reset if there are autorefresh or AWT cache group in the data store. Those cache groups must be dropped before resetting the cache administration user ID and password.

Example 4.3

```
ttAdmin -cacheUidPwdSet -cacheUid testuser -cachePwd mypass
myOraCache
```

If you change the cache administration user ID or password, you must restart the cache agent.

Note: Do not set the cache administration user ID and password if there will be no cache groups in the data store. If they are set when there will be no cache groups, the cache administration user ID and password will be unnecessarily used and verified against the Oracle database.

Start the cache agent from the command line

To start a cache agent from the command line, use the following syntax:

```
ttAdmin -cacheStart DSN
```

Example 4.4

```
ttAdmin -cacheStart myOraCache
```

If you try to start the cache agent for a data store identified in the DSN with a relative path, TimesTen looks for the data store relative to where TimesTen is running and fails. For example, on Windows, if you specify the path for the data store in the DSN as `DataStore=.\dsn1` and attempt to start the cache agent with the following command:

```
ttAdmin -cacheStart dsn1
```

The cache agent looks for the data store in *install_dir*`\srv\dsn1`. It cannot find the data store at that location and cannot start. For UNIX, the cache agent looks in:

```
/var/TimesTen/ttversion/bits
```

Stop the cache agent from the command line

To stop the agent, use the following syntax:

```
ttAdmin -cacheStop DSN
```

Example 4.5

```
ttAdmin -cacheStop myOraCache
```

Note: When using the AUTOREFRESH feature, do not stop the cache agent immediately after dropping or altering a cache group. Instead, wait for at least two minutes. The cache agent needs this time to clean up the Oracle objects used by AUTOREFRESH.

Set the cache agent start policy from the command line

You can set the cache agent start policy by calling the **ttCachePolicySet** procedure. Use the **ttCachePolicyGet** procedure to return the current policy.

The default cache agent start policy is *manual*. If you want the cache agent to restart automatically when the TimesTen daemon restarts, set the cache agent start policy to *always*. The cache agent starts immediately when you set the policy to *always*.

Example 4.6

Set the cache agent start policy to *always*:

```
CALL ttCachePolicySet ('always');
```

Controlling the cache agent from a program

To control the cache agent for a data store from your program, first connect to the data store. Use the **ttCacheUidPwdSet**, **ttCacheStart**, **ttCacheStop** or **ttCachePolicySet** procedures to perform the following tasks:

- Set the cache administration user ID and password from a program
- Start the cache agent from a program
- Stop the cache agent from a program
- Set the cache agent start policy from a program

Note: If Access Control is enabled, then ADMIN privileges are required to start, stop, and set policies for the cache agent and the replication agent. ADMIN privileges are also required to set the cache administration user ID and password.

See Chapter 2, “Built-In Procedures” in *Oracle TimesTen In-Memory Database API Reference Guide* for more information about the **ttCacheUidPwdSet**, **ttCacheStart**, **ttCacheStop** and **ttCachePolicySet** procedures.

Set the cache administration user ID and password from a program

You must have a cache administration user account that has the Oracle privileges described in "Create Oracle users and set privileges" on page 76 if you are using one of the following types of cache groups:

- ASYNCHRONOUS WRITETHROUGH cache group
- READONLY cache group
- USERMANAGED cache group with the AUTOREFRESH attributes

The cache administration user ID must conform to the following rules:

- Its length is from 1 to 30 bytes.
- It must begin with an alphabetic character.
- It can contain only ASCII alphanumeric characters, underscores (_), dollar signs ($) and pound signs (#).

The cache administration password must conform to the following rules:

- Its length is from 1 to 30 bytes long.
- It can contain all ASCII characters except the semicolon (;).

Set the cache administration user ID and password before you start the cache agent. Use the **ttCacheUidPwdSet** procedure.

Note: Do not set the cache administration user ID and password if there will be no cache groups in the data store. If they are set when there will be no cache groups, the cache administration user ID and password will be unnecessarily used and verified against the Oracle database.

Example 4.7

In this example, the data store is identified by the connection handle *hdbc*. The cache administration user ID is *testuser*, and the cache administration user password is *mypass*.

```
sprintf( stmt, "CALL ttCacheUidPwdSet('testuser','mypass')");
       rc = SQLAllocStmt( hdbc, &hstmt );
       rc = SQLExecDirect( hstmt, (SQLCHAR *) stmt, SQL_NTS );

sprintf( stmt, "CALL ttCacheStart()" );
       rc = SQLAllocStmt( hdbc, &hstmt2 );
       rc = SQLExecDirect( hstmt2, (SQLCHAR *) stmt, SQL_NTS );
```

Start the cache agent from a program

Use the **ttCacheStart** procedure to start the cache agent.

Example 4.8

In this example, the data store is identified by the connection handle *hdbc*.

```
sprintf( stmt, "CALL ttCacheStart()" );
       rc = SQLAllocStmt( hdbc, &hstmt );
       rc = SQLExecDirect( hstmt, (SQLCHAR *) stmt, SQL_NTS );
```

Note: If you have autocommit disabled, you must commit *before* and *after* calling **ttCacheStart**.

Stop the cache agent from a program

Use the **ttCacheStop** procedure to stop the cache agent.

Example 4.9

```
sprintf( stmt, "CALL ttCacheStop()" );
       rc = SQLAllocStmt( hdbc, &hstmt );
       rc = SQLExecDirect( hstmt, (SQLCHAR *) stmt, SQL_NTS );
```

The **ttCacheStop** procedure has an optional parameter, *stopTimeout*, that specifies how long the TimesTen daemon waits for the cache agent to stop. If the cache agent does not stop in the specified time frame, then the TimesTen daemon stops the cache agent. The default value of *stopTimeout* is 100 seconds. A value of 0 indicates to wait forever.

Example 4.10

To stop the cache agent and set *stopTimeout* to 160 seconds:

```
sprintf( stmt, "CALL ttCacheStop(160)" );
       rc = SQLAllocStmt( hdbc, &hstmt );
       rc = SQLExecDirect( hstmt, (SQLCHAR *) stmt, SQL_NTS );
```

Note: If you have autocommit disabled, you must commit *before* and *after* calling **ttCacheStop**.

Note: When using the AUTOREFRESH feature, do not stop the cache agent immediately after dropping or altering a cache group. Instead, wait for at least two minutes. The cache agent uses this time to clean up the Oracle objects used by AUTOREFRESH.

Set the cache agent start policy from a program

You can set the cache agent start policy by calling the **ttCachePolicySet** procedure. Use the **ttCachePolicyGet** procedure to return the current policy.

The default cache agent start policy is *manual*. If you want the cache agent to restart automatically when the TimesTen daemon restarts, set the cache agent start policy to *always*. The cache agent starts immediately when you set the policy to *always*.

Example 4.11

Set the cache agent start policy to *always* for the data store identified by the *hdbc* connection handle:

```
sprintf( stmt, "CALL ttCachePolicySet ('always')");
       rc = SQLAllocStmt( hdbc, &hstmt );
```

```
rc = SQLExecDirect( hstmt, (SQLCHAR *) stmt, SQL_NTS );
```

Checking the status of the cache agent

You can use either the **ttStatus** utility or the **ttDataStoreStatus** procedure to check that the TimesTen cache agent is running.

Example 4.12

To use the **ttStatus** utility confirm that the cache agent for the data store located in *c:\temp\cgds* is running, enter:

```
C:\>ttStatus
TimesTen status report as of Wed Apr 07 15:04:45 2004

Daemon pid 484 port 15000 instance MYCOMPUTER
No TimesTen server running

----------------------------------------------------------------
Data store c:\temp\cgds
There are 4 connections to the data store
Data store is in shared mode
Shared Memory KEY Global\DBI3cd02077.0.SHM.26 HANDLE 0x380
Subdaemon pid 964 context 0x5d82d0 connected (KEY
Global\DBI3cd02077.0.SHM.26)
Process pid 2040 context 0x97b200 connected (KEY
Global\DBI3cd02077.0.SHM.26)
Process pid 2040 context 0x1427c70 connected (KEY
Global\DBI3cd02077.0.SHM.26)
Process pid 2040 context 0x144bd70 connected (KEY
Global\DBI3cd02077.0.SHM.26)
TimesTen's cache agent is running for this data store
cache agent restart policy: manual
```

You can use the **ttAdmin** utility with the `-query` option to confirm the policy settings for a data store, including the cache agent restart policy.

Example 4.13

```
C:\>ttAdmin -query cgDSN
RAM Residence Policy            : inUse
Replication Agent Policy        : manual
Replication Manually Started    : False
cache agent Policy             : manual
cache agent Manually Started   : True
```

Example 4.14

If you have created an AWT cache group and have started the replication agent for the data store, the **ttStatus** utility also reports the status of the Oracle and replication agents. See "Starting the replication agent for an AWT cache group" on page 90.

In this example, the replication and the cache agent are running.

```
C:\> ttStatus
TimesTen status report as of Wed May  4 13:44:30 2005
Daemon pid 25337 port 16000 instance -
No TimesTen server running
No TimesTen webserver running
----------------------------------------------------------------
------- Data store /datastore/cache There are 15 connections to
the data store Data store is in shared mode Shared Memory KEY
0x260e30c2 ID 521502725
Cache agent    pid 25545 context 0x82fd820 name timestenorad
connid 2 connected (KEY 0x260e30c2)
Cache agent    pid 25545 context 0x832d798 name timestenorad
connid 4 connected (KEY 0x260e30c2)
Cache agent    pid 25545 context 0x8335198 name timestenorad
connid 3 connected (KEY 0x260e30c2)
Cache agent    pid 25545 context 0xaf27ad08 name timestenorad
connid 6 connected (KEY 0x260e30c2)
Process         pid 25485 context 0x80bd8b0 name cache connid 8
connected (KEY 0x260e30c2)
Replication     pid 25559 context 0x8193110 name  connid 7
connected (KEY 0x260e30c2)
Replication     pid 25559 context 0x81bddb0 name  connid 9
connected (KEY 0x260e30c2)
Replication     pid 25559 context 0x81e53f8 name  connid 12
connected (KEY 0x260e30c2)
Replication     pid 25559 context 0x82040d8 name  connid 5
connected (KEY 0x260e30c2)
Replication     pid 25559 context 0x8229148 name  connid 10
connected (KEY 0x260e30c2)
Replication     pid 25559 context 0x824e1b8 name  connid 11
connected (KEY 0x260e30c2)
Subdaemon       pid 25350 context 0x8092960 name Worker connid
2044 connected (KEY 0x260e30c2)
Subdaemon       pid 25350 context 0x8110140 name Flusher connid
2045 connected (KEY 0x260e30c2)
Subdaemon       pid 25350 context 0x813a838 name Checkpoint
connid 2047 connected (KEY 0x260e30c2)
Subdaemon       pid 25350 context 0x81608f8 name Monitor connid
2046 connected (KEY 0x260e30c2)
Replication policy  : Manual
Replication agent is running.
Cache agent policy : Manual
TimesTen's Cache agent is running for this data store
```

Starting the replication agent for an AWT cache group

You must start the replication agent if you are using an asynchronous writethrough (AWT) cache group.

The CREATE ASYNCHRONOUS WRITETHROUGH CACHE GROUP statement creates a replication scheme that enables the data store to communicate with the Oracle database. The replication scheme is managed entirely by TimesTen and does not require user intervention. The replication scheme is dropped when you use the DROP CACHE GROUP statement to drop the AWT cache group.

You can start the replication agent from the command line by using the **ttAdmin** utility with the `-repStart` option.

Example 4.15 Start the replication agent for the *AWTdsn* data store.

```
ttAdmin -repStart AWTdsn
```

You can start the replication agent from a program by using the **ttRepStart** procedure.

Example 4.16 A user with a cache administration user ID of *testuser* and a password of *mypass* can use:

- **ttCacheUidPwdSet** procedure to set the cache administration user ID and password
- **ttCacheStart** procedure to start the cache agent
- **ttRepStart** procedure to start the replication agent for the data store *AWTdsn*

```
sprintf( stmt, "CALL ttCacheUidPwdSet('testuser','mypass')");
       rc = SQLAllocStmt( hdbc, &hstmt );
       rc = SQLExecDirect( hstmt, (SQLCHAR *) stmt, SQL_NTS );

sprintf( stmt, "CALL ttCacheStart()" );
       rc = SQLAllocStmt( hdbc, &hstmt2 );
       rc = SQLExecDirect( hstmt2, (SQLCHAR *) stmt, SQL_NTS );

sprintf( stmt, "CALL ttRepStart()" );
       rc = SQLAllocStmt( hdbc, &hstmt3 );
       rc = SQLExecDirect( hstmt3, (SQLCHAR *) stmt, SQL_NTS );
```

Note: Stop the replication agent (for example, **ttAdmin** `-repStop` *`DSN`*) before issuing a CREATE ASYNCHRONOUS WRITETHROUGH CACHE GROUP or a DROP CACHE GROUP statement for an AWT cache group.

Applying a cache group definition to a data store

After you have described the cache groups in a SQL file, you can execute the SQL on the TimesTen data store using the `-f` option to the **ttIsql** utility. The syntax is:

```
ttIsql -f file.sql DSN
```

Example 4.17

If your cache group is described in a file called, *CG1.sql*, you can execute the file on a DSN, called *cgDSN*, by entering:

```
C:\> ttIsql -f CG1.sql cgDSN
```

You can also execute the SQL file containing your cache group definition from the **ttIsql** command line. For example:

```
Command> run CG1.sql
```

Setting a passthrough level

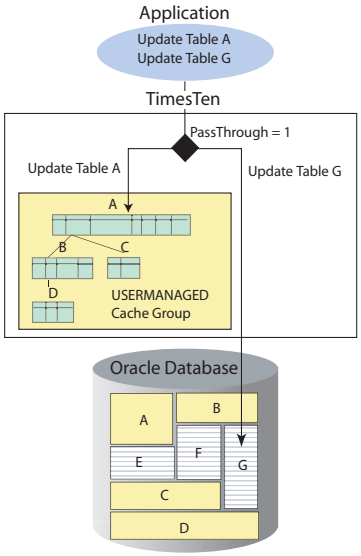
Your application can send SQL statements to either TimesTen or Oracle over a single TimesTen connection.

Whether SQL is directed to TimesTen or Oracle depends on the composition of SQL statement and the setting of the **PassThrough** connection attribute. You can set the **PassThrough** attribute to define which types of SQL statements are to be handled locally by TimesTen and which are to be redirected to Oracle.

For example, you can set **PassThrough**=0 (the default) to specify that all SQL is to be executed in TimesTen or **PassThrough**=3 to specify that all SQL is to be passed to Oracle for execution.

As shown in Figure 4.1, you can set **PassThrough**=1 to pass through SQL for a target table that is not in TimesTen or that generates a syntax error on TimesTen. For example, an UPDATE into a cached table (*Table A*) is handled in the TimesTen cache, while an UPDATE into a table that is not found in TimesTen (*Table G*) is passed through to Oracle. Similarly, TimesTen-specific procedures, such as **ttCacheStart** or **ttCkpt**, are handled by TimesTen, while nonsupported procedures are passed through to Oracle.

Figure 4.1 PassThrough=1



At the **PassThrough**=1 setting, no DDL statements are passed through to Oracle. Procedures that do not exist in TimesTen are passed through to Oracle. If a DML statement for a cache group table contains bad syntax, it is not sent to Oracle. However, all SELECT statements containing bad syntax are sent to Oracle.

At all passthrough levels, if a transaction has passed any DDL or DML statements to Oracle, then commits and rollbacks are executed in both Oracle and TimesTen. Otherwise, they are executed only in TimesTen.

A READONLY cache group prohibits updates by DML statements on the cache group. If you plan to use a READONLY cache group, you can set **PassThrough**=2 to direct all DML statements for tables in the cache group to Oracle. Otherwise, this setting is identical to **PassThrough**=1, so that all DDL statements and syntactically compatible non-DML statements and procedures are directed to TimesTen. The same passthrough behavior applies to READONLY tables in a USERMANAGED cache group. For example, consider the example cache group shown in Figure 4.1. If you have the READONLY attribute set on *Table A* and **PassThrough** = 2, an UPDATE into *Table A* is passed through to Oracle.

The passthrough feature is not recommended for DML operations on tables in AWT and SWT cache groups. Updates in AWT groups are by definition asynchronous through the cache, but passthrough renders them synchronous, which may have unintended results. Updates in SWT cache groups can result in self-deadlocks if the passthrough feature is implemented.

For a complete description of how the **PassThrough** attribute setting affects which SQL statements are executed in TimesTen or passed through to Oracle and under what circumstances, see "PassThrough" in *Oracle TimesTen In-Memory Database API Reference Guide*.

Passthrough and parameter binding

Ensure that your application supplies the correct SQL data types for passthrough statements. The ODBC driver converts the C and SQL types and presents the converted data and the SQL type code to TimesTen. TimesTen passes the information to Oracle.

TimesTen handles parameters of passthrough statements differently from parameters of statements that execute in TimesTen itself. The TimesTen query optimizer assigns data types to parameters of TimesTen statements. Applications can obtain these data type assignments using the **SQLDescribeParam** ODBC function. In contrast, parameters of passthrough statements do not have data types assigned to them. TimesTen reports the data types of parameters of passthrough statements as VARCHAR2(4000). This data type is just a placeholder. It does not mean that TimesTen expects to receive the parameter values as VARCHAR2 data. Instead, applications must inform TimesTen about the data types of parameters of passthrough statements when they call the **SQLBindParameter** ODBC function. The fSqlType argument of **SQLBindParameter** indicates the parameter's SQL data type. The cbColDef and ibScale arguments indicate its precision and scale, if applicable.

For example, to bind a parameter of SQL data type INTEGER to a NUMBER(3) parameter, call the **SQLBindParameter** ODBC function with the following arguments:

```
fCType=SQL_C_SLONG
fSqlType=SQL_DECIMAL
cbColDef=3
ibScale=0
```

Alternatively, you can use the following arguments:

```
fCType=SQL_C_SLONG
fSqlType=SQL_INTEGER
cbColDef=0
ibScale=0
```

cbColDef and ibScale are ignored for SQL_INTEGER.

Changing the passthrough level

You can override the **PassThrough** setting in the DSN with the `passthrough` command in a **ttIsql** session. You can also programmatically reset the passthrough setting for a specific transaction by calling the **ttOptSetFlag** procedure and specifying 'passthrough' as the *optflag* parameter and the new

passthrough setting as the *optval* parameter. The new passthrough setting takes effect immediately. At the end of the transaction, passthrough is reset to the original value specified by the **PassThrough** attribute.

Note: **PassThrough** behaves like any other optimizer flag. The **PassThrough** level established when a statement is prepared is the level used when the statement is executed. If the **PassThrough** level has changed between the time a statement is prepared and when it is executed, the original **PassThrough** level is used for execution.

Monitoring cache groups

The following sections describe how to obtain configuration information on your cache groups and how to monitor the status of cache operations:

- Using the ttIsql cachegroup command
- Monitoring autorefresh cache groups

Using the ttIsql cachegroup command

You can obtain information about the cache groups currently in use by a data store by using the **ttIsql** `cachegroup` command, as shown below in Example 4.18.

Example 4.18

```
Command> cachegroup;

Cache Group TESTUSER.CUSTOMERORDERS:

  AutoRefresh Mode: Incremental
  AutoRefresh State: On
  AutoRefresh Interval: 5 Minutes

  Root Table: TESTUSER.CUSTOMER
  Where Clause: (none)
  Type: Read Only

  Child Table: TESTUSER.ORDERTAB
  Where Clause: (none)
  Type: Read Only

Cache Group TESTUSER.TABCACHE:

  Root Table: TESTUSER.TAB3
  Where Clause: (none)
  Type: Propagate
```

```
2 cache groups found.
```

Monitoring autorefresh cache groups

TimesTen provides the following tools for monitoring autorefresh cache groups:

- The **ttCacheAutorefreshStatsGet** built-in procedure
- Messages in the support log
- A SQL script that can be executed on the Oracle database to display information about change log tables
- The AUTOREFRESH component of the **ttTraceMon** utility
- SNMP traps

For more information about these tools, see "AUTOREFRESH tracing" and "Monitoring autorefresh cache groups" in *Oracle TimesTen In-Memory Database Troubleshooting Procedures Guide*.

Managing Oracle objects for READONLY, AUTOREFRESH, and AWT cache groups

READONLY cache groups and some USERMANAGED cache groups use the AUTOREFRESH feature to propagate Oracle updates automatically to the TimesTen cache.

As described in "AUTOREFRESH cache group attribute" on page 48, you can establish either a full or incremental autorefresh for your cache group. When configured for incremental autorefresh, TimesTen requires that a trigger, log table, and other objects be created for each Oracle *base table* specified in the cache group. The trigger is fired with each insert, update, and delete operation on the Oracle base table. The trigger records the primary key of the updated record in the log table. The TimesTen cache agent periodically searches the log table for updated keys and joins the updated Oracle base tables to get a snapshot of the latest updates.

The Oracle objects needed for autorefresh are automatically installed by TimesTen when you create a cache group with the AUTOREFRESH INCREMENTAL attribute, as described in "Automatically installing Oracle objects" on page 98. Alternatively, you can manually install the Oracle objects, as described in "Manually installing Oracle objects" on page 99, to allow users with lesser Oracle privileges to create cache groups that use triggers.

Before you can install the Oracle objects, you must:

- Have a cache administration user account that grants the extended cache administration user privileges listed for CREATE READONLY CACHE GROUP and CREATE USERMANAGED CACHE GROUP with

AUTOREFRESH INCREMENTAL in "Create Oracle users and set privileges" on page 76.

- Set the cache administration user ID and password with **ttCacheUidPwdSet**.
- Start the cache agent, as described in "Starting and stopping the cache agent" on page 82.

For each cache administration user, TimesTen creates the following tables, where *version* is an internal TimesTen version number:

- TT_*version*_USER_COUNT
- TT_*version*_AGENT_STATUS
- TT_*version*_SYNC_OBJS

For each table in the cache group, TimesTen creates the following objects, where *number* is the object ID of the original Oracle table and *version* is an internal TimesTen version number

Object	Description
TT_*version_number*_L	Change log table for recording the changes to the Oracle table
TT_*version_number*_T	Trigger to record changes in the change log table. The trigger is fired when there are inserts, deletes, and updates on the cached table.

AWT cache groups require an additional Oracle object. A table called TT_*version*_REPPPEERS is created on Oracle to track the state and the last statement applied to Oracle. This table is created automatically by the CREATE ASYNCHRONOUS WRITETHROUGH CACHE GROUP statement. Alternatively, the Oracle tables can be created manually *before* an AWT cache group is created. See "Manually installing Oracle objects for AWT cache groups" on page 100.

If an AWT cache group is part of an active standby pair replication scheme, then a table called TT_*version*_REPACTIVESTANDBY is created automatically on Oracle when the active standby pair is created.

TimesTen grants SELECT, INSERT, UPDATE, and DELETE privileges to the user PUBLIC on the following tables:

- TT_*version_number*_L
- TT_*version*_USER_COUNT
- TT_*version*_REPPEERS (created for AWT cache groups)
- TT_*version*_REPACTIVESTANDBY (created for AWT cache groups in an active standby replication scheme)

This rest of this section the following topics:

- Automatically installing Oracle objects
- Manually installing Oracle objects
- Manually installing Oracle objects for AWT cache groups
- Confirming installation of Oracle objects
- Removing Oracle objects
- Manually removing Oracle objects for AWT cache groups

Automatically installing Oracle objects

To direct TimesTen to automatically install the Oracle objects, ensure that the cache administration user has all the required privileges for automatic creation of Oracle objects. See "Create Oracle users and set privileges" on page 76. The Oracle objects are automatically created when the cache group is created in the PAUSED or ON state or when the cache group state is altered to PAUSED or ON.

Example 4.19 shows how to direct TimesTen to install the Oracle objects automatically for a READONLY cache group.

1. Use **ttIsql** to connect to the `cgDSN` data store.
2. Call the **ttCacheUidPwdSet** procedure to set the cache administration user ID (`testuser`) and password (`mypsswrd`) for the data store.
3. Call the **ttCacheStart** procedure to start the cache agent.
4. Use CREATE READONLY CACHE GROUP to create a READONLY cache group, which has the AUTOREFRESH INCREMENTAL and STATE set to PAUSED by default.

Example 4.19

```
> ttIsql cgDSN
Command> call ttCacheUidPwdSet('testuser','mypsswrd');
Command> call ttCacheStart();
Command> CREATE READONLY CACHE GROUP readcache
         FROM
         user1.readtab
         (a NUMBER NOT NULL PRIMARY KEY, b VARCHAR2(31));
```

Example 4.20 shows how to direct TimesTen to install the Oracle objects automatically for an AWT cache group.

Example 4.20

```
> ttIsql cgDSN
Command> call ttCacheUidPwdSet('testuser','mypsswrd');
Command> call ttCacheStart();
Command> CREATE ASYNCHRONOUS WRITETHROUGH CACHE GROUP writecache
         FROM user1.writetab (a NUMBER NOT NULL PRIMARY KEY,
                              b VARCHAR2(31));
```

Manually installing Oracle objects

If you do not want to grant users the full set of CREATE privileges required to install Oracle objects automatically, you can manually install the Oracle objects and allow users with fewer privileges to create AWT cache groups and cache groups with the AUTOREFRESH INCREMENTAL attribute. See "Manually installing Oracle objects for AWT cache groups" on page 100.

See "Create Oracle users and set privileges" on page 76 for the extended Oracle privileges required to create the objects on Oracle. The installer of the objects must be granted the same Oracle privileges as the user who selects the automatic installation option.

Obtain the SQL code needed to create the Oracle objects from TimesTen by one of the following methods:

- From a **ttIsql** prompt, enter a `cachesqlget` command with the INSTALL flag.
- From a program, call the **ttCacheSqlGet** built-in procedure and specify '1' for the *install_flag* parameter.

Note: When obtaining the SQL code to install the Oracle objects, you should also generate the script to obtain the SQL code to remove the objects, as described in "Removing Oracle objects" on page 102, and save it for later use.

After you have obtained the SQL code, you can manually create the Oracle objects by starting SQL*Plus on the Oracle host machine and logging in:

`sqlplus` *`cacheUid/cachePwd@machine_name`*

At the `SQL>` prompt, enter the SQL code to create the objects.

Example 4.21

Use **ttIsql** to connect to the data store. Then call the **ttCacheStart** procedure to start the cache agent for the data store. The cache administration user ID is *testuser* and the password is *mypsswrd*. Next, use CREATE CACHE GROUP to create a READONLY cache group with the AUTOREFRESH STATE set to OFF, and run the `cachesqlget` command to obtain the SQL necessary to install the Oracle objects:

```
> ttIsql cgDSN
Command> call ttCacheUidPwdSet('testuser', 'mypsswrd');
Command> call ttCacheStart();
Command> CREATE READONLY CACHE GROUP readcache
  AUTOREFRESH
       MODE INCREMENTAL
       INTERVAL 30 SECONDS
       STATE OFF
  FROM user1.readtab
    (a NUMBER NOT NULL PRIMARY KEY, b VARCHAR2(31));
Command> cachesqlget readcache install;
```

The output from `cachesqlget` looks similar to the following:

```
CREATE TABLE system.tt_version_user_count(tableName VARCHAR2(65),
userCount
.... more code .....
END;
.
RUN;
```

You can cut and paste the SQL from TimesTen to the SQL*Plus prompt on Oracle:

```
SQL> CREATE TABLE system.tt_version_user_count(tableName
VARCHAR2(65), userCount
.... more code .....
END;
.
RUN;
```

Alternatively, you can ask **ttIsql** to save the `cachesqlget` output to a file by providing the output file name on the command line. Then you can copy the file to the Oracle system and execute it there to install the Oracle objects.

Note: If you manually install Oracle objects, you must remove them manually. See "Removing Oracle objects" on page 102.

Manually installing Oracle objects for AWT cache groups

When you manually install Oracle objects for AWT cache groups, the objects must be installed *before* creating the AWT cache group for the Oracle instance. You need to install the objects only once for each Oracle instance and cache administration user ID and password.

Perform the following tasks to install the Oracle objects for an AWT cache group:

1. Use the **ttCacheUidGet** procedure within a **ttIsql** session to check whether the cache administration user ID and password are already set. In this example, the DSN is `cgDSN`. The NULL result indicates that no cache administration user ID or password are set for `cgDSN`.

   ```
   > ttIsql cgDSN
   Command> call ttCacheUidGet();
   < <NULL> >
   1 row found.
   ```

2. Set the cache administration user ID and password by using **ttCacheUidPwdSet**(*uid*,*pwd*). The following example sets the cache administration user ID to `testuser` and the password to `mypsswrd` from within a **ttIsql** session and then verifies that it has been set.

```
Command> call ttCacheUidPwdSet('testuser','mypsswrd');
Command> call ttCacheUidGet();
< testuser >
```

3. Obtain the SQL to install the objects by using the **ttIsql** `cachesqlget` command with the ASYNCHRONOUS_WRITETHROUGH and INSTALL flags.

   ```
   Command> cachesqlget ASYNCHRONOUS_WRITETHROUGH INSTALL;
   ```

   The output is similar to the following:

   ```
   CREATE TABLE testuser.TT_03_RepPeers(
     replication_name CHAR(31) NOT NULL,
     replication_owner CHAR(31) NOT NULL,
     tt_store_id NUMBER(19,0) NOT NULL,
     subscriber_id NUMBER(19,0) NOT NULL,
     commit_timestamp NUMBER(19,0),
     commit_seqnum NUMBER(19,0),
     timerecv NUMBER(10,0),
     protocol NUMBER(10,0),
     PRIMARY KEY(tt_store_id));

   GRANT INSERT, UPDATE, DELETE, SELECT
     ON testuser.TT_03_RepPeers TO PUBLIC;

   CREATE TABLE testuser.TT_03_RepActiveStandby(
     tt_store_pair INTEGER NOT NULL,
     tt_store_id1 NUMBER(19,0) NOT NULL,
     tt_store_id2 NUMBER(19,0) NOT NULL,
     ts1 NUMBER(19,0),
     ts2 NUMBER(19,0),
     role1 CHAR(1),
     role2 CHAR(1),
     state NUMBER(10,0),
     rep_checksum NUMBER(19,0),
     PRIMARY KEY(tt_store_pair));

   CREATE INDEX testuser.TT_03_RepActiveStandby_ix
     ON testuser.TT_03_RepActiveStandby
     ( tt_store_id1, tt_store_id2 );

   GRANT INSERT, UPDATE, DELETE, SELECT
     ON testuser.TT_03_RepActiveStandby TO PUBLIC;
   ```

4. Cut and paste SQL code into a SQL*Plus session to create TT_*version*_REPPEERS. Alternatively, you can execute the code from an ODBC or JDBC program on a TimesTen connection with the **PassThrough** attribute set to 3.

   Create TT_*version*_REPACTIVESTANDBY and TT_*version*_REPACTIVESTANDBY_IX if the AWT cache group will be replicated in an active standby pair.

Note: If you install Oracle objects manually, you must remove them manually. See "Manually removing Oracle objects for AWT cache groups" on page 102.

Confirming installation of Oracle objects

To confirm the objects are installed in Oracle, log into your Oracle account and execute the following query from the Oracle SQL *Plus command prompt:

```
SQL> select owner, object_name, object_type from all_objects where
object_name like 'TT\___\_%' escape '\';
```

The output should include objects similar to the following:

```
OWNER                OBJECT_NAME                  OBJECT_TYPE
------------- ------------------------------ -----------------
TESTUSER               TT_03_34520_L                TABLE
TESTUSER               TT_03_34520_T                TRIGGER
TESTUSER               TT_03_USER_COUNT             TABLE
TESTUSER               TT_03_REPPEERS               TABLE
```

Removing Oracle objects

If the Oracle objects were automatically installed by Cache Connect to Oracle, as described in "Automatically installing Oracle objects" on page 98, they will be automatically removed when you set AUTOREFRESH to OFF or use DROP CACHE GROUP to remove the cache group.

If you manually installed the Oracle objects, as described in "Manually installing Oracle objects" on page 99, you can remove the objects from your Oracle system by executing the code to uninstall them. To obtain the uninstall code from TimesTen, use one of the following methods:

- From a **ttIsql** prompt, enter a `cachesqlget` command, specifying `uninstall`.
- From a program, call the **ttCacheSqlGet** procedure and set the *install_flag* to 0.

Note: If the cache agent is shut down immediately after dropping or altering an autorefresh cache group, then it *may* not uninstall the Oracle objects. When the cache agent is restarted, it will uninstall the Oracle objects of the dropped or altered autorefresh cache group.

Manually removing Oracle objects for AWT cache groups

Remove Oracle objects for AWT cache groups after dropping *all* AWT cache groups from *all* data stores that use an Oracle instance.

1. From a **ttIsql** session, execute the `cachesqlget` command to obtain the SQL to remove the objects. In this example, the DSN is `cgDSN` and the user is `testuser`.

```
> ttIsql cgDSN
Command> cachesqlget ASYNCHRONOUS_WRITETHROUGH UNINSTALL;
```

The output looks similar to the following:

```
DROP TABLE testuser.TT_03_RepPeers;

DROP TABLE testuser.TT_03_RepActiveStandby;
```

2. Cut and paste the SQL into a SQL*Plus session to remove the objects. Alternatively, you can execute the code from an ODBC or JDBC program on a TimesTen connection with the **PassThrough** attribute set to 3.

5

Using Cache Groups

This chapter describes how to use cache groups. It includes the following topics:

- Altering a cache group
- Dropping a cache group
- Copying data between the Oracle database and cache groups
- Loading and refreshing a cache group
- Using transparent loading
- Flushing a USERMANAGED cache group
- Unloading a cache group
- Replicating cache group tables
- Changing the Oracle schema

Altering a cache group

Use the ALTER CACHE GROUP statement to make changes to the AUTOREFRESH STATE, INTERVAL, and MODE settings. Any values or states set by ALTER CACHE GROUP are persistent; they are stored in the data store and survive TimesTen daemon and cache agent restarts.

Note: If TimesTen was installed with Access Control enabled, you must have DDL privileges to the data store to use the ALTER CACHE GROUP statement. See Chapter 1, "Access Control" in the *Oracle TimesTen In-Memory Database Installation Guide* for details.

When an AUTOREFRESH operation is in progress and you change the STATE to OFF:

- The AUTOREFRESH operation stops if the **LockWait** general connection attribute is greater than 0. The ALTER CACHE GROUP SET AUTOREFRESH STATE OFF statement preempts the AUTOREFRESH operation.
- The AUTOREFRESH operation continues if the **LockWait** general connection attribute is 0. The ALTER CACHE GROUP SET AUTOREFRESH STATE OFF statement will fail with a lock timeout error.

Example 5.1

This example activates AUTOREFRESH on the *AutorefreshCustomers* cache group by changing its state to ON:

```
ALTER CACHE GROUP AutorefreshCustomers
   SET AUTOREFRESH STATE ON;
```

Dropping a cache group

Use the DROP CACHE GROUP SQL statement to delete a cache group and all of its related tables from TimesTen.

Note: If TimesTen was installed with Access Control enabled, you must have DDL privileges to the data store to use the DROP CACHE GROUP statement. See Chapter 1, "Access Control" in the *Oracle TimesTen In-Memory Database Installation Guide* for details.

If you are dropping a READONLY cache group or a USERMANAGED cache group that uses AUTOREFRESH in INCREMENTAL mode and you have *manually* installed the objects on Oracle, as described in "Manually installing Oracle objects" on page 99, you must manually remove the Oracle objects before dropping the cache group. See "Removing Oracle objects" on page 102 for details.

If you enter a DROP CACHE GROUP statement when an AUTOREFRESH operation is in progress:

- The AUTOREFRESH operation stops if the **LockWait** general connection attribute is greater than 0. The DROP CACHE GROUP statement preempts the AUTOREFRESH operation.
- The AUTOREFRESH operation continues if the **LockWait** general connection attribute is 0. The DROP CACHE GROUP statement will fail with a lock timeout error.

To drop the *WesternCustomers* cache group:

```
DROP CACHE GROUP WesternCustomers
```

All the tables within the *WesternCustomers* cache group will be immediately dropped from TimesTen and the definition of the cache group will be removed from the TimesTen system tables.

Note: Any AWT setting will remain in effect until the DROP CACHE GROUP transaction is committed. If the cache group is AWT, make sure that all pending updates have been applied to Oracle before dropping the cache group. Use the **ttRepSubscriberWait** procedure.

Copying data between the Oracle database and cache groups

TimesTen provides several mechanisms to copy data from Oracle to TimesTen and from TimesTen to Oracle.

An application can use the following SQL statements to copy data between Oracle tables and a cache group:

SQL Statement	Description
LOAD CACHE GROUP	Load cache instances not already in the cache from Oracle data
REFRESH CACHE GROUP	Replace cache instances with current Oracle data
FLUSH CACHE GROUP	Copy rows from cache group tables to Oracle tables. The cache group must be USERMANAGED without PROPAGATE enabled.

SWT cache groups, AWT cache groups, and USERMANAGED cache groups with PROPAGATE enabled provide automatic ways to apply changes from a cache group to Oracle tables. See the cache group descriptions in Chapter 3, "Defining a Cache Group"and "PROPAGATE" on page 50.

The AUTOREFRESH cache group attribute can be used in READONLY and USERMANAGED cache groups to automatically apply changes from Oracle tables to a cache group. See "AUTOREFRESH cache group attribute" on page 48.

You can use the transparent load feature to load data from Oracle tables into cache group tables when a SELECT query does not find data in the cache group tables. You can use transparent load for any type of cache group except cache groups with the AUTOREFRESH attribute. See "Using transparent loading" on page 113.

AUTOREFRESH is part of a cache group definition. Transparent load can be implemented after a cache group has been defined and is invoked as needed. There are fewer restrictions on cache groups when transparent load is configured than on AUTOREFRESH cache groups.

AUTOREFRESH is appropriate for relatively static data such as product catalogs or flight schedules. Transparent loading is useful for dynamic cache content. Applications can use the transparent load feature to load data into the cache and let TimesTen aging remove the data automatically when the data is no longer being used.

Loading and refreshing a cache group

You can load data from Oracle into a cache group using either a LOAD CACHE GROUP or REFRESH CACHE GROUP statement. Both SQL statements copy data from Oracle into the cache group with or without WHERE clauses or WITH ID clauses. Loading and refreshing differ as follows:

- LOAD CACHE GROUP does not update instances that are already present in the cache group.
- REFRESH CACHE GROUP replaces all or specified instances in the cache group with the most current Oracle data, even if an instance is already present in the cache group. A refresh operation is equivalent to an UNLOAD CACHE GROUP statement followed by a LOAD CACHE GROUP statement. All changes to the Oracle data, including inserts, updates, and deletes in the root and child tables, are reflected in the cache after a refresh operation.

Example 5.2 To load data from Oracle to the *WesternCustomers* cache group:

```
LOAD CACHE GROUP WesternCustomers
    COMMIT EVERY 256 ROWS;
```

Example 5.3 To load only the cache group instances where the customers are in Zip Code 94022 from Oracle to the *WesternCustomers* cache group:

```
LOAD CACHE GROUP WesternCustomers
    WHERE (user1.customer.zip = 94022)
    COMMIT EVERY 256 ROWS;
```

Example 5.4 To refresh the entire *WesternCustomers* cache group from the Oracle tables:

```
REFRESH CACHE GROUP WesternCustomers
    COMMIT EVERY 256 ROWS;
```

Example 5.5 To refresh the only the cache group instances associated with the order item 2353:

```
REFRESH CACHE GROUP WesternCustomers
    WHERE (user1.orderdetails.itemid = 2353)
    COMMIT EVERY 256 ROWS;
```

For more information, see "LOAD CACHE GROUP" and "REFRESH CACHE GROUP" in *Oracle TimesTen In-Memory Database SQL Reference Guide*.

The rest of this section includes the following topics:

- Loading and refreshing AUTOREFRESH and READONLY cache groups
- Loading and refreshing a cache group WITH ID
- Loading and refreshing cache groups with multiple tables
- Improving performance of loading and refreshing large tables

Loading and refreshing AUTOREFRESH and READONLY cache groups

The following restrictions apply to loading and refreshing AUTOREFRESH and READONLY cache groups:

- When you load an AUTOREFRESH or READONLY cache group, the cache group must be empty.
- The AUTOREFRESH state must be PAUSED.
- The LOAD or REFRESH statement cannot contain a WHERE clause or a WITH ID clause.

Perform the following tasks to load an AUTOREFRESH cache group:

1. Set the AUTOREFRESH state to PAUSED.

```
ALTER CACHE GROUP testcache SET AUTOREFRESH STATE PAUSED;
COMMIT;
```

2. Unload the cache group.

```
UNLOAD CACHE GROUP testcache;
COMMIT;
```

3. Load the cache group.

```
LOAD CACHE GROUP testcache COMMIT EVERY 256 ROWS;
COMMIT;
```

4. Set the AUTOREFRESH state to ON.

```
ALTER CACHE GROUP testcache SET AUTOREFRESH STATE ON;
COMMIT;
```

See "Unloading a cache group" on page 116.

Loading and refreshing a cache group WITH ID

The WITH ID clause enables you to load or refresh a cache group based on values of the primary key without using a WHERE clause. You can represent the primary key values as literals or as binding parameters. Using the WITH ID clause is faster than using the equivalent WHERE clause. It also enables you to roll back the load transaction if it fails.

The WITH ID clause cannot be used for AUTOREFRESH or READONLY cache groups.

The WITH ID clause allows you to load one cache instance at a time. Suppose you have an *orders* table with a primary key of *order_id*. If a customer calls about a specific order, the information can be loaded by loading the cache instance for the specified *order_id*.

Example 5.6 On Oracle, create a table called *sample*:

```
CREATE TABLE sample (a NUMBER, b NUMBER, c NUMBER,
    PRIMARY KEY (a,b));
```

Populate the table so that it contains the following rows:

```
         A          B          C
---------- ---------- ----------
         1          2          3
         4          5          6
         7          8          9
```

Create a cache group on TimesTen:

```
CREATE SYNCHRONOUS WRITETHROUGH CACHE GROUP testcache
    FROM sample
      (a NUMBER, b NUMBER, c NUMBER, PRIMARY KEY (a,b));
```

Load the cache group with the row that has (1,2) as a primary key:

```
LOAD CACHE GROUP testcache WITH ID (1,2);
1 cache instance affected.

SELECT * FROM sample;
< 1, 2, 3>
1 row found.
```

Example 5.7

Refresh the *testcache* cache group with the row that has (4,5) as a primary key:

```
REFRESH CACHE GROUP testcache WITH ID (4,5);
1 cache instance affected.

SELECT * FROM sample;
< 1, 2, 3 >
< 4, 5, 6 >
2 rows found.
```

Loading and refreshing cache groups with multiple tables

If your cache group contains multiple tables and Oracle is currently updating the tables to be loaded or refreshed, you may want to set the TimesTen isolation level to SERIALIZABLE before submitting LOAD CACHE GROUP or REFRESH CACHE GROUP statement. This causes TimesTen to query the Oracle tables in a serializable manner and guarantees that the loaded data is transactionally consistent. After you have loaded or refreshed the cache group, you can reset the isolation level back to READ_COMMITTED for better concurrency.

Example 5.8

During a **ttIsql** session you can reset the isolation level before loading a cache group. This applies to nologging mode when you want to load or refresh a cache group.

```
Command> isolation SERIALIZABLE;
Command> LOAD CACHE GROUP WesternCustomers
>COMMIT EVERY 1000 ROWS;
Command> isolation READ_COMMITTED;
```

Improving performance of loading and refreshing large tables

You can improve the performance of loading or refreshing large tables by using the PARALLEL clause of the LOAD CACHE GROUP or REFRESH CACHE GROUP statement. Specify the number of threads to be used. One thread fetches rows from Oracle, and the other threads insert data into the cache group. Do not specify more threads than the number of CPUs available for loading.

Example 5.9

```
REFRESH CACHE GROUP WesterCustomers
    COMMIT EVERY 256 ROWS
    PARALLEL 2;
```

Using transparent loading

You can configure TimesTen to load data automatically from Oracle tables into cache group tables when a SELECT query does not find data in the cache group tables. When the SELECT statement is executed, the selected or related rows are loaded into the root table and related or selected rows are loaded into the child tables. Only rows that satisfy the cache group definition are loaded. If the cache group has a time-based aging policy defined, the rows must satisfy the aging policy.

You can configure all types of cache groups for transparent loading *except* READONLY cache groups and USERMANAGED cache groups with the AUTOREFRESH attribute.

Transparent load is particularly useful when cache content is dynamic. For example, you can use it to load data that has been removed by aging. See "Implementing aging in a cache group" on page 64.

This section includes the following topics:

- Types of SELECT statements
- Configuring transparent loading
- Overriding transparent loading for a transaction

Types of SELECT statements

Transparent load is available for the following types of SELECT statements:

- SELECT using an equality condition on a primary key on a single table.The equality condition must include a constant or a parameter. For example:

  ```
  SELECT * FROM table1 WHERE primkey=1;
  ```

  If the primary key is compound, the SELECT statement must include the equality condition on all of the primary key columns. For example:

  ```
  SELECT * FROM table2 WHERE pkcol1=10 AND pkcol2=10;
  ```

- SELECT using an equality condition on a foreign key on a single table. The equality condition must include a constant or a parameter. For example:

  ```
  SELECT * FROM table2 WHERE foreignkey=1;
  ```

  If the foreign key is compound, the SELECT statement must include the equality condition on all of the foreign key columns. For example:

  ```
  SELECT * FROM table2 WHERE fkcol1=10 AND fkcol2=10;
  ```

- Select a single subtree of a cache group instance, using an equality condition. For example:

  ```
  SELECT * FROM table1,table2 where table1.primkey=1 and
  table2.foreignkey=table1.primkey;
  ```

The SELECT query must meet the following conditions:

- The SELECT query must be the outermost query in the statement.
- The SELECT query cannot include UNION, INTERSECT, or MINUS set operators.
- Tables of only one cache group can be specified in the outermost SELECT query, but other tables that are not in a cache group can be specified in the statement.

When the SELECT query returns selected rows, the entire cache instance will be loaded in order to maintain the relationship between primary keys and foreign keys.

Configuring transparent loading

To load data transparently, set the **TransparentLoad** Cache Connect attribute to 1. This enables a SELECT statement to be executed on the Oracle tables. The resulting data is loaded into the cache group tables. The data is then returned transparently by the original SELECT statement that was executed on the cache group tables.

The following table summarizes the settings for **TransparentLoad**:

Setting	Description
`0`	Do not use transparent loading. (Default)
`1`	Run the SELECT statement in TimesTen without issuing error or warning messages.
`2`	Returns an error if the SELECT operation cannot use transparent load. The SELECT operation will be executed based on the data available in TimesTen.

An alternative way to enable transparent loading is to use the **SQLSetConnectOption** ODBC function to set the TT_TRANSPARENT_LOAD ODBC connection option. The setting applies to the entire connection.

Overriding transparent loading for a transaction

You can override the **TransparentLoad** attribute or the TT_TRANSPARENT_LOAD ODBC option for a specific transaction by using the `TransparentLoad` flag of the **ttOptSetFlag** built-in procedure. The `TransparentLoad` flag value takes effect when the statement is prepared and cannot be changed at runtime. After the transaction has been committed or rolled back, the connection setting takes effect again.

Flushing a USERMANAGED cache group

The FLUSH CACHE GROUP statement flushes inserts and updates from USERMANAGED cache groups to Oracle. It does not flush deletes. Records from tables that are specified as READONLY or PROPAGATE cannot be flushed to Oracle.

Use FLUSH CACHE GROUP when commit propagation from TimesTen to Oracle is turned off. Rather than propagating every transaction upon commit, many transactions can be committed on TimesTen before changes are propagated to Oracle. For each cache instance, if the cache instance exists in Oracle, the operation in Oracle consists of an update. If the cache instance does not exist in Oracle, TimesTen inserts it.

The FLUSH CACHE GROUP statement can be specified with a WHERE or a WITH ID clause to control what data is flushed to Oracle. See"FLUSH CACHE GROUP" in *Oracle TimesTen In-Memory Database SQL Reference Guide* for more information.

Example 5.10 To flush the changes from the *WesternCustomers* cache group to Oracle:

```
FLUSH CACHE GROUP WesternCustomers;
```

Unloading a cache group

You can remove some or all instances from a cache group with the UNLOAD CACHE GROUP SQL statement. Unlike the DROP CACHE GROUP statement, the tables themselves are not dropped.

Use the UNLOAD CACHE GROUP statement carefully with cache groups that have the AUTOREFRESH attribute. A row that is unloaded can reappear in the cache group as the result of an autorefresh operation if the row or its child rows are updated in Oracle.

Example 5.11

To remove all instances from the *WesternCustomers* cache:

```
UNLOAD CACHE GROUP WesternCustomers;
```

To remove the customer with *CustId = 1* from the *WesternCustomers* cache:

```
UNLOAD CACHE GROUP WesternCustomers WITH ID(1);
```

or

```
UNLOAD CACHE GROUP WesternCustomers
  WHERE (User1.Customer.CustId = 1);
```

Replicating cache group tables

You can replicate cache group tables in one data store to cache group tables in another data store or to standard TimesTen tables in another data store. The cache group type in the master data store must be the same as the cache group type in the subscriber data store. For example, tables in a READONLY cache group can be replicated only to tables in a READONLY cache group in another data store or to standard tables in another data store.

All tables in a cache group must included in the replication scheme.

You can achieve high availability of cache instances by configuring a type of replication scheme called an *active standby pair* for the following types of cache groups:

- A READONLY cache group
- An ASYNCHRONOUS WRITETHROUGH cache group

An active standby pair that replicates one of these types of cache groups can change the role of a cache group automatically 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 "Active standby pairs with cache groups" in *TimesTen to TimesTen Replication Guide*.

You can also configure replication of cache group tables to achieve a balanced workload. The following configurations are examples:

- Load balancing — Configure unidirectional replication from tables in a cache group with the AUTOREFRESH attribute to standard tables. Updates on Oracle are automatically refreshed to the TimesTen cache group. The updates are then replicated to TimesTen tables.
- Split workload — Configure bidirectional replication between cache group tables in WRITETHROUGH cache groups in different data stores. Design your applications so that transactions for a specific purpose (for example, a geographical region) updates only particular cache group tables and that the same cache group table cannot be updated directly in both stores. Updates to a cache group table in one store are replicated to the corresponding cache group table in the other data store. Updates made directly by applications are propagated to Oracle, but replicated updates are not.

Do not use active/active configurations, especially when you replicate cache groups. Unintended consequences of such configurations include unresolvable deadlocks, divergent contents of the cache groups, and recovery problems.

See "Cache groups and replication" and "Replicating cache groups" in *TimesTen to TimesTen Replication Guide* for more information.

Changing the Oracle schema

Cache Connect to Oracle does not have a mechanism to detect schema changes in the Oracle database caused by actions such as CREATE, DROP, or ALTER. If you need to make changes to the Oracle schema, then perform the following tasks:

1. Use DROP CACHE GROUP to drop all cache groups that cache the Oracle tables to be changed.
2. Stop the cache agent.
3. Make the desired changes to the Oracle schema.
4. Use CREATE CACHE GROUP to re-create the cache groups.

If you do not drop the cache group before the schema is modified in the Oracle database, then operations on the original cache group, including AUTOREFRESH, may fail or may succeed with the wrong semantics.

Incremental AUTOREFRESH cannot detect truncation of an Oracle table. If you want to truncate an Oracle base table, then perform the following tasks:

1. Use the ALTER CACHE GROUP statement to set the AUTOREFRESH STATE to PAUSED.
2. Truncate the Oracle base table.
3. Refresh the cache group by using the REFRESH CACHE GROUP statement *without* the WHERE or WITH ID clauses.

Autorefresh operations will resume after you refresh the cache group.

Note: Truncating a TimesTen table that is in a cache group is not permitted.

Note: If the cache group is AWT, make sure that all pending updates have been applied to Oracle before dropping the cache group. Use the **ttRepSubscriberWait** procedure.

6

Cache Administrator

This chapter includes the following topics:

- About the Cache Administrator
- Security
- Starting the Cache Administrator
- Connecting to a data store
- Using the Cache Administrator

About the Cache Administrator

TimesTen provides a Web-based tool called the *Cache Administrator* for creating and working with cache group definitions. The Cache Administrator enables you to define one or more cache groups by navigating through your Oracle schema with a Web browser located on the same machine as your TimesTen installation. You can set up your Web server to enable access to Cache Connect to Oracle. See "Web server configuration" in Chapter 2, "TimesTen Installation" in the *Oracle TimesTen In-Memory Database Installation Guide*.

The following Web browsers are supported for the Cache Administrator:

- Internet Explorer 6.0
- Firefox 1.5 and later

Security

Cache group definitions are stored in the `ws/cgi-bin/cache/cachedata` file in the daemon home directory. You can obtain the daemon home directory by entering the `ttVersion -m` command.

The cache group definitions include the definitions of queries and indexes as well as connection information. If Access Control is enabled, the connection information includes the Oracle user and the TimesTen user.

The Oracle user account must have the following privileges:

- CREATE SESSION
- SELECT for the tables you need to access for cache group creation

When Access Control is enabled for a TimesTen instance, the user must be an internal TimesTen user. The TimesTen user must have the following privileges:

- DDL
- SELECT

When Access Control is enabled, the TimesTen user must have the ADMIN privilege so it can start and stop the cache agent from the command line.

See Table 4.1, "Oracle privileges required for cache group operations," on page 77 for a detailed list of Oracle privileges.

Starting the Cache Administrator

Before starting the Cache Administrator, make sure that:

- The Oracle client is installed on the machine where you want to start the Cache Administrator.
- The ORACLE_HOME environment variable is set correctly. It must be set before you start the TimesTen daemon. See "Configuring Cache Connect to Oracle on a UNIX platform" on page 75 or "Configuring Cache Connect to Oracle on a Windows platform" on page 75 for more information about environment variables.
- LD_LIBRARY_PATH on UNIX or PATH on Windows is set correctly.
- Cache Connect to Oracle is successfully installed.
- The TimesTen daemon is running.
- The Web server is running. You can verify this by using **ttStatus**. The instance administrator can start it by using the **ttDaemonAdmin** utility with the `-startwebserver` option.

Start the Cache Administrator by entering the following URL into the browser:

`http://localhost:port/cache`

The local host is the only supported location from which the Web server can be started. This means that the Cache Administrator must run on the machine that TimesTen is installed on. Attempting to view the Cache Administrator from other machines will fail.

`port` is the port number of the daemon's Web server. The port number was specified during installation. If you do not know the port number, look in the daemon's Web server log, `webserver.log`, in the daemon's directory.

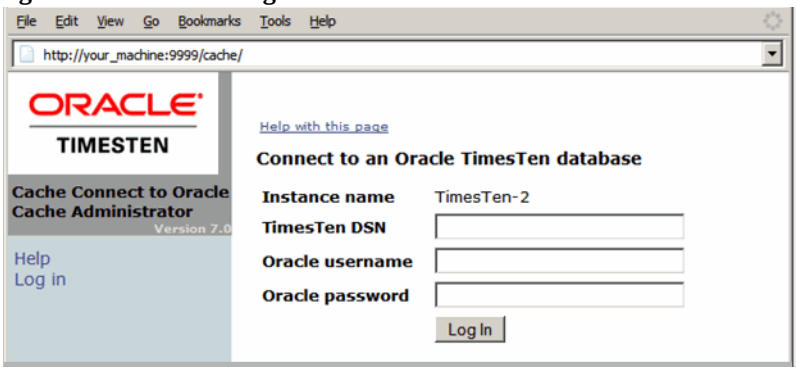
WINDOWS

On Windows platforms, you can start the Cache Administrator from the **Start** > **Programs** > **TimesTen** menu.

Connecting to a data store

After you have started the cache agent for a data store, choose **Login** to connect to the data store. Figure 6.1 shows the **Connect to a Data Store** screen that appears if Access Control is *not* enabled on the data store.

Figure 6.1 Connecting to a data store without Access Control



The following fields must be completed:

- **TimesTen DSN** — Indicates which TimesTen data store is used for the cache. You can enter a simple string (the name of a DSN) or the full connection string.

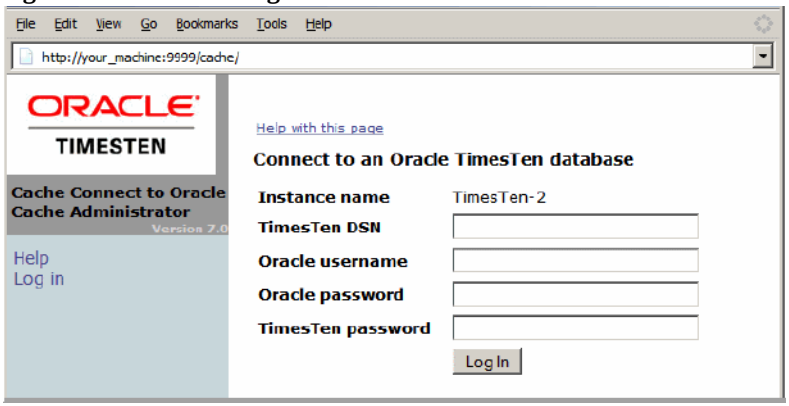
  The simple string is the name of a DSN that has been created either in the ODBC panel on Windows or in the SYS.ODBC.INI file on UNIX. For more information about creating a DSN, see Chapter 1, "Creating TimesTen Data Stores" in *Oracle TimesTen In-Memory Database Operations Guide*.

  The following is an example of a full connection string:

  ```
  DSN=CacheData;DurableCommits=0;DatabaseCharacterSet=US7ASCII;
  ```

- **Oracle username** — A user account on the Oracle database. It retrieves information concerning the tables and indexes stored on Oracle.
- **Oracle password** — The password of the specified Oracle user.
- **TimesTen password** — If Access Control is enabled on the data store, you must also enter your TimesTen Password, as shown in Figure 6.2. Password entries are case-sensitive.

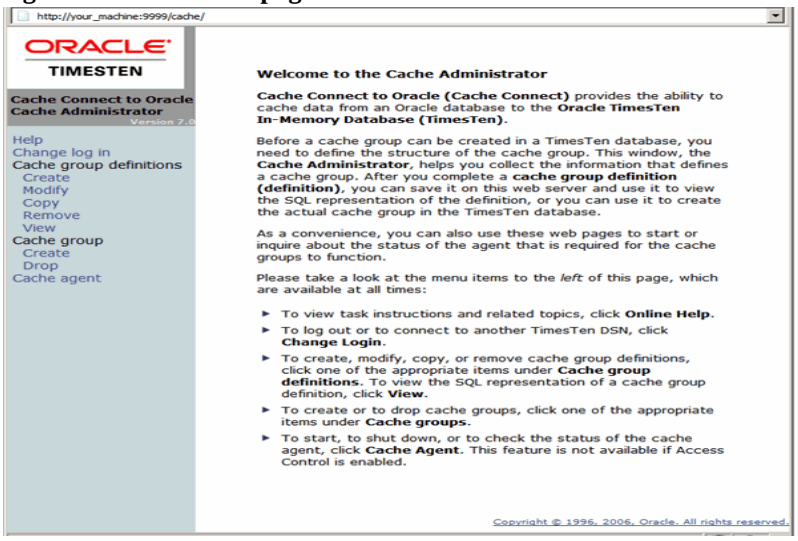
Figure 6.2 Connecting to a data store with Access Control enabled



After making your entries, click **Log in**.

When you have successfully connected to a data store, the Cache Administrator options appear as shown in Figure 6.3.

Figure 6.3 Welcome page and full menu



Using the Cache Administrator

You can perform the following tasks with the Cache Administrator:

- Manage cache group definitions (create, copy, modify, remove).
- View the SQL statement that represents a cache group.
- Generate the SQL statement necessary for creating the cache group and send the information to the data store where it will be created.

- Create cache groups.
- Manage the cache agent (start, stop, status).

The online help for the Cache Administrator explains how to perform these tasks.

7

Implementing Cache Connect in a RAC Environment

This chapter describes how to implement Cache Connect to Oracle in a RAC environment. It includes the following topics:

- How Cache Connect works in a RAC environment
- Restrictions on using Cache Connect in a RAC environment
- Setting up Cache Connect in a RAC environment

How Cache Connect works in a RAC environment

Oracle Real Application Clusters (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. RAC handles read/write consistency and load balancing while providing high availability.

Fast Application Notification (FAN) is a RAC feature that was integrated with Oracle Call Interface (OCI) in Oracle Database 10g Release 2. See *Oracle Database Oracle Clusterware and Oracle Real Application Clusters Administration and Deployment Guide* for more information about RAC and FAN. FAN publishes information about changes in the cluster to applications that subscribe to FAN events. FAN eliminates inefficiencies 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

Without FAN, it can take several minutes for Cache Connect to receive notification of an Oracle failure.

Transparent application failover (TAF) is a feature of Oracle Net Services that enables you to specify how you want applications to reconnect after a failure. Cache Connect uses OCI integrated with FAN to receive notification of Oracle events. See *Oracle Database Net Services Administrator's Guide* for more information about TAF. OCI applications can use one of the following types of Oracle Net failover functionality:

- Session: If a user's connection is lost, a new session is automatically created for the user. This type of failover does not attempt to recover selects.
- Select: This type of failover enables users with open cursors to continue fetching on them after failure.
- None: This is the default. No failover functionality is used. This can also be explicitly specified to prevent failover from happening.

Cache Connect can recover quickly from Oracle failures without intervention from the user. Cache Connect can be used in a RAC environment with the Oracle Database 10g Release 2 server. The Oracle client can be Oracle9i Release 2 or Oracle Database 10g Release 2. See "Restrictions on using Cache Connect in a RAC environment" on page 129 for more information about Oracle database releases.

Cache Connect behavior depends on the actions of TAF and how TAF is configured. The **RACCallback** Cache Connect attribute registers TAF and FAN callbacks by default. If you do not want TAF and FAN capabilities, then set **RACCallback** to 0 in the connection string.

Table 7.1 shows the behaviors of Cache Connect features and operations in a RAC environment with different TAF failover types. These behaviors are the same for Oracle9i Release 2 and Oracle Database 10g Release 2 clients except that Cache Connect receives immediate notification of node failure when FAN is enabled in Oracle Database 10g Release 2.

Table 7.1 Behavior of Cache Connect Features and Operations in a RAC Environment

Feature or Operation	TAF Failover Type	Cache Connect Behavior After a Failed Connection to the Oracle Database
Autorefresh	None	The cache agent automatically shuts down, restarts, and waits until a connection can be established to the Oracle service. This behavior is the same as in Cache Connect to Oracle release 6.0.1 and for Cache Connect in a non-RAC environment.
Autorefresh	Session	One of the following occurs: • All failed connections are recovered. Autorefresh operations that were in progress will be rolled back and tried again. • If TAF times out or cannot recover the connection, the cache agent automatically shuts down, restarts, and waits until a connection can be established to the Oracle service. This behavior is the same as in Cache Connect to Oracle release 6.0.1 and for Cache Connect in a non-RAC environment.

Feature or Operation	TAF Failover Type	Cache Connect Behavior After a Failed Connection to the Oracle Database
Autorefresh	Select	One of the following occurs: • Autorefresh resumes from the point of connection failure. • Autorefresh operations that were in progress will be rolled back and tried again. • If TAF times out or cannot recover the connection, the cache agent automatically shuts down, restarts, and waits until a connection can be established to the Oracle service. This behavior is the same as in Cache Connect to Oracle release 6.0.1.
Passthrough, SWT, propagate, and flush	None	The application is notified of the loss of connection and must roll back the TimesTen connection. During the next passthrough operation, Cache Connect tries to reconnect to the Oracle database. All modified session attributes are lost. This behavior is the same as in Cache Connect to Oracle release 6.0.1 and for Cache Connect in a non-RAC environment.
Passthrough, SWT, propagate, and flush	Session	One of the following occurs: • The connection to the Oracle database is recovered. If there were open cursors, DML or lock operations on the lost connection, then an error occurs and the user must roll back the transaction. Otherwise the user can continue without rolling back. • If TAF times out or cannot recover the connection, the application is notified of the loss of connection and must roll back the TimesTen connection. During the next passthrough operation, Cache Connect tries to reconnect to the Oracle database. All modified session attributes are lost. This behavior is the same as in Cache Connect to Oracle release 6.0.1 and for Cache Connect in a non-RAC environment.
Passthrough	Select	The connection to the Oracle database is recovered. If there were DML or lock operations on the lost connection, then an error occurs and the user must roll back the transaction. Otherwise the user can continue without rolling back.

Feature or Operation	TAF Failover Type	Cache Connect Behavior After a Failed Connection to the Oracle Database
SWT, propagate, and flush	Select	One of the following occurs: • The connection to the Oracle database is recovered. If there were DML or lock operations on the lost connection, then an error occurs and the user must roll back the transaction. Otherwise the user can continue without rolling back. • If TAF times out or cannot recover the connection, the application is notified of the loss of connection and must roll back the TimesTen connection. During the next passthrough operation, Cache Connect tries to reconnect to the Oracle database. All modified session attributes are lost. This behavior is the same as in Cache Connect to Oracle release 6.0.1 and for Cache Connect in a non-RAC environment.
Load and refresh	None	The application receives a loss of connection error.
Load and refresh	Session	One of the following occurs: • The load succeeds. • An error is returned, saying that an Oracle fetch cannot be executed.
Load and refresh	Select	One of the following occurs: • If the Oracle cursor is open and the cursor is recovered or if the Oracle cursor is not open, then the load succeeds. • An error is returned if TAF was unable to recover either the session or open Oracle cursors. **Note:** An error is less likely to be returned than if the TAF failover type is Session.
AWT	None	The replication receiver thread for AWT exits. A new thread is spawned and tries to connect to the Oracle database.
AWT	Session, select	One of the following occurs: • If the connection is recovered and there are uncommitted DML operations on the connection, then the batch operation or transaction will be rolled back and executed again. • If the connection is recovered and there are no uncommitted DML operations, then work continues.

Restrictions on using Cache Connect in a RAC environment

You can use Cache Connect to Oracle in a RAC environment for the following Oracle client and server releases:

- Oracle Database 10*g* Release 2. FAN is available only if both the client and the server are Oracle Database 10*g* Release 2.
- Oracle Database 10g Release 1
- Oracle9i Release 9.2

Use the latest version of the Oracle client for each release. For more information about Oracle client/server interoperability, see "Cache Connect support for Oracle client/server releases" on page 132.

Cache Connect support of RAC has the following restrictions:

- Cache Connect behavior is limited to RAC, FAN, and TAF capabilities. For example, if all nodes for a service fail, the service will not be restarted. Cache Connect 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.
- Cache Connect uses the Oracle Client Interface (OCI) integrated for FAN events. This interface automatically spawns a thread to wait for an Oracle event. This is the only TimesTen feature that spawns a thread in a TimesTen direct link ODBC application. Adapt your ODBC application to account for this thread creation. If you do not want the extra thread, set the **RACCallback** Cache Connect attribute to 0. Then TAF and FAN will not be used.

Setting up Cache Connect in a RAC environment

Install Cache Connect and RAC. Ensure that the cache administration user has SELECT privileges on the Oracle GV$SESSION dynamic performance view.

There are two TimesTen environment variables that control TAF timeouts:

- `TT_ORA_FAILOVER_TIMEOUT`: TAF timeout in minutes for the user application (SWT cache groups, cache groups with the propagate option, and cache groups using the passthrough feature). The default is 5 minutes.
- `TT_AGENT_ORA_FAILOVER_TIMEOUT`: TAF timeout in minutes for the replication agent (for AWT cache groups). The default is 5 hours.

Make sure that the TimesTen Data Manager and cache agent are started. The following Oracle components should also be started:

- Oracle instances
- Oracle listeners
- Oracle service that will be used for Cache Connect

Then perform the following tasks:

1. Verify that the **RACCallback** connection attribute is set to 1 (the default).
2. Use the `DBMS_SERVICE.MODIFY_SERVICE` Oracle PL/SQL package or Oracle Enterprise Manager to enable publishing of FAN events. This changes the `AQ_HA_NOTIFICATIONS` column in the Oracle `ALL_SERVICES` view to `YES`. (This applies to Oracle Database 10g Release 2 clients.)

   See *Oracle Database PL/SQL Packages and Types Reference* for more information about `DBMS_SERVICE.MODIFY_SERVICE`.
3. Enable TAF on the service by *one* of the following methods:
   - Create a service for Cache Connect in the Oracle `tnsnames.ora` file with the following characteristics:
     - `LOAD_BALANCE=ON` (optional)
     - `FAILOVER_MODE=(TYPE=SELECT)` or `FAILOVER_MODE=(TYPE=SESSION)`
   - Use `DBMS_SERVICE.MODIFY_SERVICE` to set the TAF failover type.

   See *Oracle Database Net Services Administrator's Guide.*
4. If your application is an ODBC direct link application, link it with a thread library so that it will receive FAN notifications. FAN spawns a thread to monitor for failures. (This applies to Oracle Database 10g Release 2 clients.)

8

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.

This chapter includes the following topics:

- Summary of compatibility issues
- Cache Connect support for Oracle client/server releases
- Transaction semantics
- JDBC compatibility
- ODBC compatibility
- SQL compatibility

Note: TimesTen does not support Oracle Call Interface (OCI).

Summary of compatibility issues

Consider the following differences between TimesTen and Oracle:

- TimesTen and Oracle database metadata are stored differently. See"ODBC compatibility" on page 136 and "JDBC compatibility" on page 132 for more information.
- TimesTen and Oracle have different transaction isolation models. See "Transaction semantics" on page 132 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.
- TimesTen and Oracle have different types and lengths for ROWID. TimesTen does not support the Oracle ROWID data type. See "Data type support" on page 140 for more information.
- Sequences are not cached and synchronized between the cache and the Oracle database. See "SQL expressions" on page 142 for more information.

- Side effects of triggers and procedures are not propagated to the cache until after an AUTOREFRESH or a manual REFRESH CACHE GROUP operation.
- The XA and JTA APIs are not supported in cache groups.

Cache Connect support for Oracle client/server releases

Cache Connect to Oracle respects all supported interoperability combinations of Oracle client/server configurations. For example, using Cache Connect with a 9.2.0.8 Oracle client installation on the local TimesTen server and connecting to a 10.2.0.3 Oracle database on a different server is supported. Customers should check Metalink Documentation Note 207303.1, “Client/Server/Interoperability Support Between Different Oracle Versions”, for current information on supported Oracle client/server configurations. Customers should also check both Oracle and TimesTen release notes for known interoperability problems.

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. 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 isolation levels and transaction semantics, see Chapter 7, “Transaction Management and Recovery” in *Oracle TimesTen In-Memory Database Operations Guide*.

JDBC compatibility

Compatibility issues that apply to JDBC include the following:

- JDBC database metadata functions return TimesTen metadata. If you want Oracle metadata, either use Cache Connect with the **PassThrough** attribute or connect to the Oracle Database directly.
- The set/get connection and statement attributes are executed on TimesTen.
- All Oracle **java.sql.ResultSet** metadata (length, type, label) is returned in TimesTen data type lengths. The column labels that are returned are TimesTen column labels.

- Output parameters (OUT and INOUT) for **java.sql.CallableStatement** methods are not supported in TimesTen. See "java.sql.CallableStatement" on page 135 for more information.
- Oracle extensions (`oracle.sql` and `oracle.jdbc` packages) are not supported.
- Java stored procedures are not supported in TimesTen.

Note: TimesTen does not support asynchronous execution of statements and does not support canceling an executing statement from a different thread.

---

java.sql.Connection

The following **Connection** methods have no compatibility issues:

close
commit
createStatement
setAutoCommit
prepareCall
prepareStatement
rollback

The following methods are executed locally in TimesTen:

getCatalog
getMetaData
get/setTransactionIsolation (See "Transaction semantics" on page 132 for restrictions.)
isReadOnly
isClosed - only returns TimesTen connection status
nativeSQL
setCatalog
setReadOnly

java.sql.Statement

The following **Statement** methods have compatibility issues:

close
execute
executeUpdate
executeQuery
getResultSet
getUpdateCount
getWarnings
executeBatch

clearBatch
addBatch

The following methods are executed locally in TimesTen:

get/setMaxFieldSize
get/setMaxRows
get/setQueryTimeout
getMoreResults
setEscapeProcessing
setCursorName
cancel

java.sql.ResultSet

The following **ResultSet** methods have no compatibility issues:

close
findColumn(*int*) **and findColumn**(*string*)
getXXX(*number*) and **getXXX**(*name*)
getXXXStream(*int*) and **getXXXStream**(*string*)
getMetaData

java.sql.PreparedStatement

The following **PreparedStatement** methods have no compatibility issues:

execute
executeUpdate
executeQuery
setXXX
setXXXStream
close
getResultSet
getUpdateCount
addBatch

The following methods are executed locally in TimesTen:

get/setMaxFieldSize
get/setMaxRows
get/setQueryTimeout
getMoreResults
setEscapeProccessing
setCursorName
cancel

java.sql.CallableStatement

The same restrictions as shown for the **java.sql.Statement** and **java.sql.PreparedStatement** interfaces apply to **CallableStatement**. In addition, CallableStatements with any form of output are not supported. All parameters are taken as input parameters. Callable statements with output parameters may return an error at prepare time, return an error through registerOutparameter or the application may get unexpected results.

- In a WRITETHROUGH cache group, if **PassThrough**=1, indirect DML operations that are hidden in stored procedures or induced by triggers may be passed through without being detected by Cache Connect to Oracle.
- Stored procedures that update, insert, or delete from READONLY cache group tables will be autorefreshed within another transaction in an asynchronous fashion. Thus the changes do not appear within the same transaction that the stored procedure was executed within and there may be some time lapse before the changes are autorefreshed into the cache table.

java.sql.ResultSetMetadata

The following **ResultSetMetadata** methods have no compatibility issues:

getColumnCount
getColumnType
getColumnLabel
getColumnName
getTableName
isNullable

The following methods are executed locally in TimesTen:

getSchemaName
getCatalogName
getColumnDisplaySize
getColumnType
getColumnTypeName
getPrecision
getScale
isAutoIncrement
isCaseSensitive
isCurrency
isDefinitelyWritable
isReadOnly
isSearchable
isSigned
isWritable

Stream support

The compatibility issues related to streams are:

- The JDBC driver fully fetches the data into an in-memory buffer during a call to the **executeQuery** or **next** method. The **getXXXStream** entry points return a stream that reads data from this buffer.
- Oracle supports up to 2GB of long or long raw data.
- Oracle always streams long/long raw data even if the application does not call **getXXXStream**.
- TimesTen does not have a **close** method. It invalidates the stream when it moves to the next row. Oracle closes the stream when it moves to the next column.
- TimesTen does not support the **mark**, **markSupported**, and **reset** methods.

ODBC compatibility

The following table lists the ODBC 2.5 API calls and specifies whether the function has a compatibility issue.

Function Name	Compatibility
SQLAllocConnect	-
SQLAllocEnv	-
SQLAllocStmt	-
SQLBindCol	-
SQLBindParameter	When a SQL statement has duplicate parameter names, TimesTen considers them to be multiple occurrences of the same parameter and does not allow them to be bound separately. Oracle considers them to be distinct parameters and does allow them to be bound separately.
SQLBrowseConnect	**Not supported**
SQLCancel	TimesTen does not support asynchronous execution of statements and does not support canceling an executing statement from a different thread.
SQLColAttributes	-

Function Name	Compatibility
SQLColumnPrivileges	**Not supported**
SQLColumns	Metadata functions return metadata for TimesTen tables.
SQLConnect	-
SQLDataSource	-
SQLDescribeCol	-
SQLDescribeParam	-
SQLDisconnect	-
SQLDriverConnect	-
SQLDrivers	-
SQLError	Native error codes are TimesTen errors. Users may receive generic errors like “execution at Oracle failed, Oracle error code *nnn*.”
SQLExecDirect	-
SQLExecute	-
SQLExtendedFetch	**Not supported**
SQLFetch	-
SQLForeignKeys	Metadata functions return metadata for TimesTen tables.
SQLFreeConnect	-
SQLFreeEnv	-
SQLFreeStmt	-
SQLGetConnectOption	-
SQLGetCursorName	TimesTen supports get/set cursor name but does not support the cursor delete/update statement.
SQLGetData	-

Function Name	Compatibility
SQLGetFunctions	-
SQLGetInfo	-
SQLGetStmtOption	-
SQLGetTypeInfo	Metadata functions return metadata for TimesTen tables.
SQLMoreResults	**Not supported**
SQLNativeSql	-
SQLNumParams	-
SQLNumResultCols	-
SQLParamData	-
SQLParamOptions	-
SQLPrepare	-
SQLPrimaryKeys	Metadata functions return metadata for TimesTen tables.
SQLProcedureColumns	Metadata functions return metadata for TimesTen tables.
SQLProcedures	Metadata functions return metadata for TimesTen tables.
SQLPutData	-
SQLRowCount	-
SQLSetConnectOption	Set functions set TimesTen options or metadata.
SQLSetCursorName	Set functions set TimesTen options or metadata.
SQLSetPos	**Not supported**
SQLSetScrollOptions	**Not supported**
SQLSetStmtOption	Set functions set TimesTen options or metadata.

Function Name	Compatibility
SQLSpecialColumns	Metadata functions return metadata for TimesTen tables.
SQLStatistics	Metadata functions return metadata for TimesTen tables.
SQLTablePrivileges	**Not supported**
SQLTables	Metadata functions return metadata for TimesTen tables.
SQLTransact	-

See "Supported ODBC functions" in *Oracle TimesTen In-Memory Database C Developer's and Reference 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 a different semantic. This section compares Oracle Database 10g Release 2 and Oracle TimesTen In-Memory Database 7.0.0.0.0

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:

- Stored procedure-related objects:
  - Stored procedures
  - Stored functions
  - Packages
  - Triggers
- 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. Statements that include views and materialized views are passed through to Oracle when there is no local object with the same name in TimesTen.

Differences between Oracle and TimesTen tables

Some of 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
ROWID
UROWID
CLOB
NCLOB
BLOB
BFILE
REF
“Any” types
XML types
Spatial types
Media types

The following TimesTen data types are not supported by Oracle:

TT_TINYINT
TT_SMALLINT

TT_INTEGER
TT_BIGINT
TT_VARBINARY

Note: TimesTen NCHAR/NVARCHAR2 data types are encoded as UTF-16. Oracle NCHAR/NVARCHR2 are encoded as either UTF-16 or UTF-8.

SQL operators

TimesTen supports these Oracle operators and predicates:

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

SQL functions

TimesTen supports these Oracle functions:

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

SQL expressions

TimesTen supports these Oracle expressions:

- Column Reference
- Sequence
- NULL
- ()
- Binding parameters
- Case expression
- CAST

SQL subqueries

TimesTen supports Oracle subqueries:

- IN (subquery)
- >,<,= ANY (subquery)
- >,=,< SOME (subquery)
- EXISTS (subquery)

- >,=,< (scalar subquery)
- In WHERE clause of DELETE/UPDATE
- In FROM clause

Note: A non-verifiable scalar subquery is a scalar subquery whose 'single-row-result-set' property cannot be determined until execution time. TimesTen allows at most one non-verifiable scalar subquery in the entire query and the subquery cannot be specified in an OR expression.

SQL queries

TimesTen supports these Oracle queries:

- 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 Oracle DML statements:

- INSERT INTO ... VALUES
- INSERT INTO ... SELECT
- UPDATE WHERE expression (expression may contain a subquery)
- DELETE WHERE expression (expression may contain a subquery)
- MERGE (TimesTen does not support ODBC batch execution of MERGE statements)

TimesTen-only SQL and built-in procedures

This section lists TimesTen SQL statements and built-in procedures that are not supported in Oracle. With **PassThrough= 3**, these statements are passed to Oracle 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, FLUSH CACHE GROUP, and REFRESH CACHE GROUP statements.
- All TimesTen replication management DDL statements, including CREATE REPLICATION, DROP REPLICATION, ALTER REPLICATION,

CREATE ACTIVE STANDBY PAIR, ALTER ACTIVE STANDBY PAIR, DROP ACTIVE STANDBY PAIR statements.

- All TimesTen built-in procedures. (See "Built-In Procedures" in *Oracle TimesTen In-Memory Database API Reference Guide*.)

Other SQL issues

- The data type for the Oracle ROWID pseudocolumn is ROWID. The data type for TimesTen's ROWID is BINARY(16). They are different lengths.
- PL/SQL is an Oracle language and is not supported by TimesTen.
- Oracle optimizer hints are embedded in SQL comments. TimesTen uses built-in procedures to set optimizer hints.
- Oracle and TimesTen system tables are different.
- TimesTen does not support the Oracle RETURNING clause in INSERT, UPDATE and DELETE statements.
- Oracle stored functions can return cursor references through output parameters or return values. TimesTen does not support this feature.
- When a SQL statement has duplicate parameter names, TimesTen considers them to be multiple occurrences of the same parameter and does not allow them to be bound separately. Oracle considers them to be distinct parameters and does allow them to be bound separately.

Glossary

age out

To remove cache instances from the cache group after a specified period of time (time-based) or when a specified level of data store usage is reached (usage-based).

asynchronous writethrough cache group

A cache group in which cached data is updated in TimesTen and asynchronously propagated to Oracle.

autorefresh

Updates on Oracle are automatically propagated to the TimesTen cache group by means of Oracle triggers.

AWT cache group

See asynchronous writethrough cache group.

bidirectional propagation

To copy updates made to either Oracle or the TimesTen cache group to the other.

cache agent

A TimesTen process that enables cache group operations, such as AUTOREFRESH, and COMMIT EVERY *n* ROWS.

cache group

The data from the Oracle tables cached in a TimesTen data store. A cache group can be created to cache all or part of a single Oracle table or a set of related Oracle tables. If multiple Oracle tables are cached, each table in the cache group must be related to another table in the cache group through a one-to-many relationship.

cache group primary key

The primary key of the root table in the cache group.

cache instance

A specific row of data identified by a primary key in the cache group root table. If there are multiple tables in the cache group, the cache instance consists of the set of rows associated by foreign key relationships with the row in the root table.

cache instance key

The primary key for a specific row in the root table, which identifies the root table row and all of the rows in the cache group child tables that reference that root table row.

child table

A table in the cache group that has a foreign key reference to the primary key of the root table or that of 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 table in the cache group can be a child to more than one parent in the cache group.

flush

To copy inserts or updates in the cache group from TimesTen to Oracle.

load

To copy only new instances from Oracle. Does not update or delete instances that are already present in the cache.

propagate

To copy data between Oracle and the TimesTen cache group.

refresh

To replace all cache instances in the TimesTen cache group with the most current Oracle data

replication

The process of maintaining duplicate copies of data in multiple data stores.

replication agent

Replication at each master and subscriber data store is controlled by a *replication agent.* The replication agent on the master data store reads the records from the transaction log and forwards any detected changes to replicated elements to the replication agent on the subscriber data store. The replication agent on the subscriber then applies the updates to its data store.

root table

The main table in the cache group that does not refer to 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 cache group

A cache group in which cached data is updated in TimesTen and synchronously propagated to Oracle.

system-managed cache group

System-managed cache groups enforce specific behaviors. The types of system-managed cache groups are READONLY, SYNCHRONOUS WRITETHROUGH, and ASYNCHRONOUS WRITETHROUGH.

A READONLY cache group enforces a caching behavior in which updates on Oracle tables are applied to TimesTen through the AUTOREFRESH mechanism. You cannot update tables in a READONLY cache group directly.

ASYNCHRONOUS WRITETHROUGH (SWT) cache group enforces a caching behavior in which cached data is updated in TimesTen and propagated to Oracle. When your application commits a transaction, it is committed on Oracle before it is committed on TimesTen.

An ASYNCHRONOUS WRITETHROUGH (AWT) cache group enforces the same caching behavior as an SWT cache group, in which cached data is updated in TimesTen and propagated to Oracle, but the TimesTen commit occurs asynchronously from the Oracle commit.

user-managed cache group

A user-managed cache group implements customized behavior such as bidirectional propagation.

unload

To remove some or all cache instances from a cache group.

Index

A

active/active replication schemes
    with cache groups 117
aging
    adding to existing cache group 64
    attributes 64
    foreign keys 67
    in cache groups 64
    LRU 64
    monitoring 68
    new cache group 64
    ON DELETE CASCADE 67
    parent and child tables 67
    performance 66
    scheduling 67
    sliding window 68
    state 64
    time-based 66
    usage-based 64
aging policy 64
ALTER CACHE GROUP 106
    AUTOREFRESH operation 106
AQ_HA_NOTIFICATIONS 130
ASYNCHRONOUS WRITETHROUGH cache group 38
    example of 41
    restrictions 40, 107
    set-up tasks 41
    starting replication agent 90
AUTOREFRESH 48, 83, 145
    FULL vs. INCREMENTAL 48
    INCREMENTAL mode objects 95
    transparent load comparison 108
    use in READONLY cache group 33
autorefresh
    change log table 96
AUTOREFRESH cache group attribute 48
AUTOREFRESH cache groups
    loading and refreshing 110
autorefresh cache groups
    monitoring 95
AUTOREFRESH operation
    ALTER CACHE GROUP 106
    completion when disabled 49
    DROP CACHE GROUP 107
availability
    replicating cache groups 117
AWT cache group
    set-up tasks 41
AWT cache group, see "ASYNCHRONOUS WRITETHROUGH cache group"
AWT cache groups
    manually installing Oracle objects 100
    removing Oracle objects 102

B

bidirectional propagation 43, 145
BINARY_DOUBLE data type 69
BINARY_FLOAT data type 69

C

cache administration password
    rules 84, 86
cache administration user
    setting user ID and password 33, 41, 48
cache administration user ID
    rules 84, 86
Cache Administrator 119
    security 119
    starting 120
cache agent
    start policy 85, 87
    starting 82
    stopping 82
cache group
    creating 18, 31
    data types 72
    defined 8
    dropping 107
    load performance 112
    load WITH ID 110
    modifying 106
    obtaining information about 94
    parallel load 112
    refresh performance 112
    refresh WITH ID 110
    replicating 117
    time-based aging 66
cache group definition

applying to DSNs 91
cache group table
defining 53
indexes on 53
primary keys 53
cache groups
Oracle temporary tables 53
cache instance
definition 9
loading into cache 109
unloading from cache 116
cache table attributes
ON DELETE CASCADE 52
PROPAGATE 50
READONLY 52
UNIQUE HASH ON 53
cachegroup ttIsql command 94
cachesqlget
sample output 101
change log table 80, 96
child table
defining 8, 55
COMMIT EVERY 145
COMMIT EVERY n ROWS
recommended value 30
connection string 82
CREATE CACHE GROUP 20, 31

D

data type mapping
for key columns 70
non-key columns 71
data type mappings
Cache Connect to Oracle 70
data types
cache group 72
choosing 54, 70
floating point recommendations 69
database character set
retrieving Oracle setting 16, 24
DatabaseCharacterSet attribute 81
DBMS_SERVICE.MODIFY_SERVICE Oracle PL/SQL package 130
DROP CACHE GROUP 22, 27, 107
AUTOREFRESH operation 107
DSN
creating 16, 24, 81
path name 84

F

FAN 125
Fast Application Notification 125
floating point data types
recommendations 69
FLUSH CACHE GROUP statement 115

G

GUI 119
GV$SESSION dynamic performance view 129

H

high availability
replicating cache groups 117

I

incremental autorefreshes 48
index
aging performance 66
indexes 53
instance data store 16
Isolation DSN attribute 81

J

JDBC restrictions 132

L

least recently used aging 64
load balancing 117
LOAD CACHE GROUP 109
load cache group
performance 112
load on SELECT 113
loading
WITH ID 110
loading AUTOREFRESH and READONLY cache groups 110
loading cache tables
SERIALIZABLE isolation level 111
LockLevel attribute 81
Logging attribute 81
LRU aging 64
LRU aging attributes 64

M

mapping
data types for Cache Connect 70

materialized view 60

O

ODBC connection string 82
ODBC restrictions 136
ON DELETE CASCADE table attribute 52
Oracle
    tablespace 80
Oracle account
    connecting to 19, 98, 121
    creating 15
    password 81
    user id 81
Oracle client
    installing 15
Oracle client/server support 132
Oracle objects
    manually installing for AWT cache groups 100
    removing 102
    removing for AWT cache groups 102
Oracle objects for AUTOREFRESH INCREMENTAL 95
Oracle Password 121
Oracle Real Application Clusters 125
Oracle schema, making changes to 118
Oracle synonyms 60
Oracle transparency issues 131
Oracle User 121
Oracle User ID attribute 82
ORACLE_HOME environment variable 15, 75, 83
OracleID attribute 81
OraclePWD attribute 82

P

parallel loading 112
passthrough
    AWT and SWT cache groups 92
    commits and rollbacks 92
    data types 93
    parameter binding 93
    READONLY cache groups 92
    READONLY cache table attribute 92
PassThrough attribute 81
    setting 91
phantom data store
    creating 38
    status of 88
policies
    check settings for a data store 88
policy
    cache agent start 85, 87
primary keys 55
PROPAGATE table attribute 44, 45, 50, 51
propagation, bidirectional 43
PUBLIC user privileges granted by TimesTen 96

R

RAC 125
    Oracle releases 129
    restrictions 129
RACCallback Cache Connect attribute 126, 130
READONLY cache group 33
READONLY cache groups
    loading and refreshing 110
READONLY table attribute 52
Real Application Clusters 125
REFRESH CACHE GROUP 109
refresh cache group
    performance 112
refreshing
    WITH ID 110
refreshing AUTOREFRESH and READONLY cache groups 110
refreshing cache tables
    SERIALIZABLE isolation level 111
removing Oracle objects 102
replicating cache group tables
    load balancing 117
root table
    defining 8, 55

S

security 119
SQL restrictions 139
SQLBindParameter ODBC function 93
SQLDescribeParam ODBC function 93
starting the cache agent
    policy 85, 87
SYCHRONOUS WRITETHROUGH cache group 35, 38
SYNCHRONOUS WRITETHROUGH cache group 35
    example of 36
synonyms 60
system-managed cache groups
    definition 32

T

tablespace
    cache administration user 80
    on Oracle 80
TAF 125
temporary tables
    cache groups 53
time-based aging 66
TimesTen Password 121
TNSNAMES.ORA 74
transparent application failover 125
transparent load 113
    AUTOREFRESH comparison 108
TransparentLoad attribute 114
    overriding 114
truncate and autorefresh 49, 59, 118
TT_AGENT_ORA_FAILOVER_TIMEOUT environment variable 129
TT_ORA_FAILOVER_TIMEOUT environment variable 129
TT_TRANSPARENT_LOAD ODBC connection option 113, 114
TT_version_AGENT_STATUS 96
TT_version_number_L 96
TT_version_number_T 96
TT_version_REPACTIVESTANDBY 96
TT_version_SYNC_OBJS 96
TT_version_USER_COUNT 96
ttAdmin utility 81
    -cacheUidPwdSet option 33, 41, 48
ttAgingLRUConfig procedure 64
ttCachePolicySet procedure 85
ttCachePropagateFlag procedure 51
ttCacheStart procedure 85
ttCacheStop procedure 85
ttCacheUidPwdSet built-in procedure 41
ttCacheUidPwdSet procedure 33, 48, 85
ttCcachePolicySet procedure
    command line example 85
    programmatic example 87
ttDataStoreStatus procedure 88
ttIsql -f, use of 91
ttOptSetFlag procedure
    using to override TransparentLoad attribute 114
ttStatus utility 88
TypeMode DSN attribute 81

U

UNIQUE HASH ON table attribute 53
UNLOAD CACHE GROUP 116
unloading a cache group
    WITH ID 116
unloading an AUTOREFRESH cache group 116
usage-based aging 64
USERMANAGED cache group
    example of 20, 25, 32, 43, 44, 108

W

WHERE clause 60
WITH ID
    loading a cache group 110
    refreshing a cache group 110