Oracle® TimesTen Application-Tier Database Cache

User's Guide

Release 18.1

E61196-06

July 2020

Oracle TimesTen Application-Tier Database Cache User's Guide, Release 18.1

E61196-06

Contents

1 TimesTen Application-Tier Database Cache Concepts

2 Getting Started

3 Setting Up a Caching Infrastructure

4 Defining Cache Groups

5 Cache Group Operations

6 Managing a Caching Environment

7 Cache Performance

8 Cleaning up the Caching Environment

9 Using TimesTen Cache in an Oracle RAC Environment

10 Using TimesTen Cache with Data Guard

A Procedure and Privileges for Caching Oracle Database Data in TimesTen Classic

B SQL*Plus Scripts for TimesTen Cache

C Compatibility Between TimesTen and Oracle Databases

Index

Preface

Oracle TimesTen In-Memory Database (TimesTen) is a relational database that is memory-optimized for fast response and throughput. The database resides entirely in memory at runtime and is persisted to the file system.

- Oracle TimesTen In-Memory Database in classic mode, or TimesTen Classic, refers to single-instance and replicated databases (as in previous releases).
- Oracle TimesTen In-Memory Database in grid mode, or TimesTen Scaleout, refers to a multiple-instance distributed database. TimesTen Scaleout is a grid of interconnected hosts running instances that work together to provide fast access, fault tolerance, and high availability for in-memory data.
- TimesTen alone refers to both classic and grid modes (such as in references to TimesTen utilities, releases, distributions, installations, actions taken by the database, and functionality within the database).
- TimesTen Application-Tier Database Cache, or TimesTen Cache, is an Oracle Database Enterprise Edition option. TimesTen Cache is ideal for caching performance-critical subsets of an Oracle database into cache tables within a TimesTen database for improved response time in the application tier. Cache tables can be read-only or updatable. Applications read and update the cache tables using standard Structured Query Language (SQL) while data synchronization between the TimesTen database and the Oracle database is performed automatically. TimesTen Cache offers all of the functionality and performance of TimesTen Classic, plus the additional functionality for caching Oracle Database tables.
- TimesTen Replication features, available with TimesTen Classic or TimesTen Cache, enable high availability.

TimesTen supports standard application interfaces JDBC, ODBC, and ODP.NET; Oracle interfaces PL/SQL, OCI, and Pro*C/C++; and the TimesTen TTClasses library for C++.

Audience

This guide is for application developers who use and administer TimesTen, and for system administrators who configure and manage TimesTen databases that cache data from Oracle databases. To work with this guide, you should understand how relational database systems work. You should also have knowledge of SQL, and either Open Database Connectivity (ODBC), Java Database Connectivity (JDBC), or Oracle Call Interface (OCI).

Related documents

TimesTen documentation is available at:

`https://docs.oracle.com/database/timesten-18.1`

Oracle Database documentation is also available on the Oracle documentation website. This may be especially useful for Oracle Database features that TimesTen supports but does not attempt to fully document, such as OCI and Pro*C/C++.

Conventions

TimesTen Classic is supported on multiple platforms. Unless otherwise indicated, the information in this guide applies to all supported platforms. The term Windows refers to all supported Windows platforms. The term UNIX applies to all supported UNIX platforms. The term Linux is used separately. See "Platforms and Compilers" in *Oracle TimesTen In-Memory Database Release Notes* (`README.html`) in your installation directory for specific platform versions supported by TimesTen.

Note: In TimesTen documentation, the terms "data store" and "database" are equivalent. Both terms refer to the TimesTen database.

This document uses the following text conventions:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.
`italic monospace`	Italic monospace type indicates a placeholder or a variable in a code example for which you specify or use a particular value. For example: `LIBS = -Ltimesten_home/install/lib -ltten` Replace *`timesten_home`* with the path to the TimesTen instance home directory.
[ ]	Square brackets indicate that an item in a command line is optional.
{ }	Curly braces indicate that you must choose one of the items separated by a vertical bar (|) in a command line.
|	A vertical bar separates alternative arguments.
. . .	An ellipsis (. . .) after an argument indicates that you may use more than one argument on a single command line.
% or $	The percent sign or dollar sign indicates the UNIX shell prompt, depending on the shell that is used.
#	The number (or pound) sign indicates the prompt for the UNIX root user.

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

Convention	Meaning
installation_dir	The path that represents the directory where the current release of TimesTen is installed.
timesten_home	The path that represents the home directory of a TimesTen instance.
release or *rr*	The first two parts in a release number, with or without the dot. The first two parts of a release number represent a major TimesTen release. For example, 181 or 18.1 represents TimesTen Release 18.1.
DSN	The data source name.

Note: TimesTen release numbers are reflected in items such as TimesTen utility output, file names and directory names. These details are subject to change with every minor or patch release, and the documentation cannot always be up to date. The documentation seeks primarily to show the basic form of output, file names, directory names and other code that may include release numbers. You can confirm the current release number by looking at the Release Notes or executing the `ttVersion` utility.

Documentation Accessibility

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

Access to Oracle Support

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

What's New

This section summarizes new features and functionality of TimesTen Release 18.1 that are documented in this guide, providing links into the guide for more information.

New features in Release 18.1.4.1.0

You can now set a time interval for how often to perform the calculation of the fragmentation percentage of the change log tables on the Oracle database. Use the `ttCacheConfig` built-in procedure providing the `AutorefreshLogMonitorInterval` as the value parameter. See "Defragmenting change log tables in the tablespace" on page 6-14 for details.

New features in Release 18.1.3.1.0

- Cache group autorefresh interval set to 0 milleseconds enables continuous autorefresh, where the next autorefresh cycle is scheduled immediately after the last autorefresh cycle has ended. See "AUTOREFRESH cache group attribute overview" on page 4-35 and "Minimizing delay for cached data with continuous autorefresh" on page 7-10 for details.
- Applications can have multiple dynamic load requests to the Oracle database, which could result in too many open connections to the back-end Oracle database. However, for client/server applications with multiple connections per server, you can configure TimesTen to use the cache connection pool for all connections to the Oracle database. The cache connection pool can only be utilized by an application using a client/server connection as the pooled connections are shared across all client/server connections. See "Managing a cache connection pool to the Oracle database for dynamic load requests" on page 7-1 for details.
- If you notice that your application is timing out because of a lock contention between autorefresh and dynamic load requests, you can set the `CacheCommitDurable` cache configuration parameter to 0 with the `ttCacheConfig` built-in procedure. This reduces the occurrence of lock contention between autorefresh and dynamic load requests in the same application. See "Reducing lock contention for read-only cache groups that use autorefresh and dynamic load" on page 7-11 for details.

New features in Release 18.1.2.2.0

You can reduce contention between autorefresh and dynamic load operations for dynamic read-only cache groups with incremental autorefresh by enabling the `DynamicLoadReduceContention` database system parameter. See "Reducing contention on TimesTen for dynamic read-only cache groups with incremental autorefresh" on page 7-10 for details.

New features in Release 18.1.2.1.0

Some applications choose incremental autorefresh instead of full autorefresh mode for performance reasons. However, a full autorefresh could still be requested in some situations. You can set the `DisableFullAutorefresh` cache configuration parameter to 1 to disallow any full autorefresh requests for all cache groups defined with incremental autorefresh. See "Disabling full autorefresh for cache groups" on page 4-38 for details.

New features in Release 18.1.1.1.0

- Oracle TimesTen In-Memory Database in classic mode or TimesTen Classic refers to single-instance environments and databases as in previous releases. TimesTen Cache is available with TimesTen Classic.
- TimesTen Cache is supported on multiple platforms. See "Platforms and configurations" in *Oracle TimesTen In-Memory Database Release Notes* (`README.html`) in your installation directory for specific platform versions supported by TimesTen.
- TimesTen Cache works with asynchronous Oracle Active Data Guard. You can cache tables from an Oracle Active Data Guard with the asynchronous redo transport mode into read-only cache groups. The read-only cache groups are replicated within an active standby pair replication scheme. The Active Data Guard configuration includes a primary Oracle database that communicates over an asynchronous transport to a single physical standby Oracle database. See "TimesTen Cache works with asynchronous Active Data Guard" on page 10-2 for full details.
- Cache grid and all its components are removed in this release.
- Cache Advisor is removed from TimesTen in this release.

1

TimesTen Application-Tier Database Cache Concepts

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

You can cache Oracle Database data in a TimesTen database within cache groups. A cache group in a TimesTen database can cache a single Oracle Database table or a group of related Oracle Database tables.

This chapter includes the following topics:

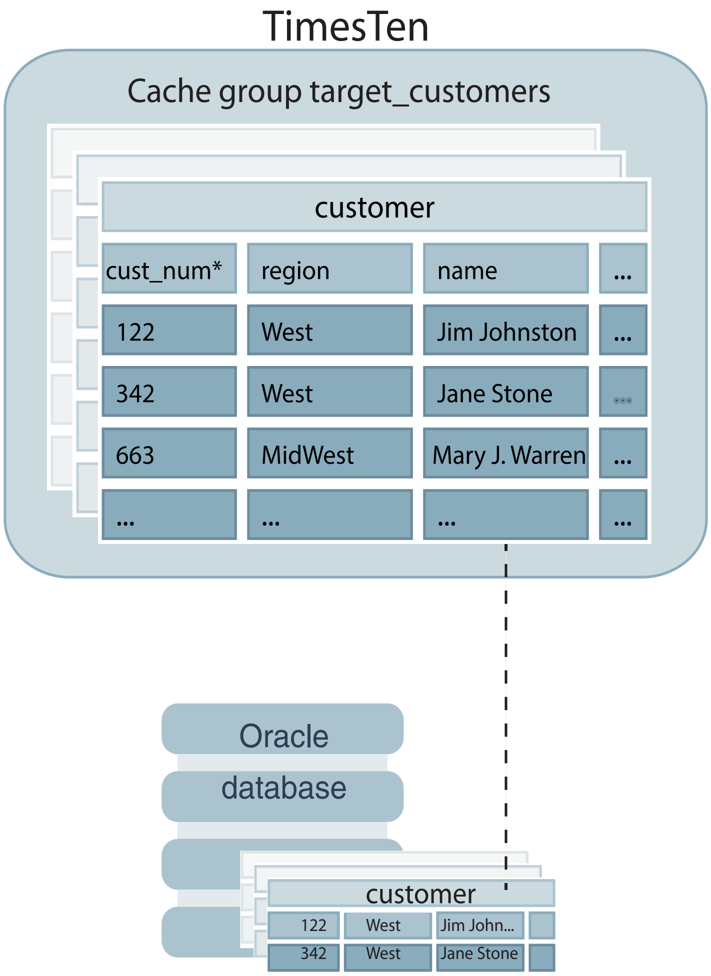
- Overview of cache groups
- High availability caching solution

Overview of cache groups

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

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

Figure 1–1 Single-table cache group



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

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

While you may have multiple TimesTen databases that interact with the same Oracle database, they will each operate independently. Thus, any data cached in separate TimesTen databases will each interact with the Oracle database independently.

An Oracle Database table cannot be cached in separate cache groups within the same TimesTen database. However, the table can be cached in separate cache groups within different TimesTen databases. If the table is cached in separate AWT cache groups and the same cache instance is updated simultaneously on multiple TimesTen databases, there is no guarantee as to the order in which the updates are propagated to the cached Oracle Database table. In addition, the contents of the updated cache tables can be inconsistent on the separate TimesTen databases.

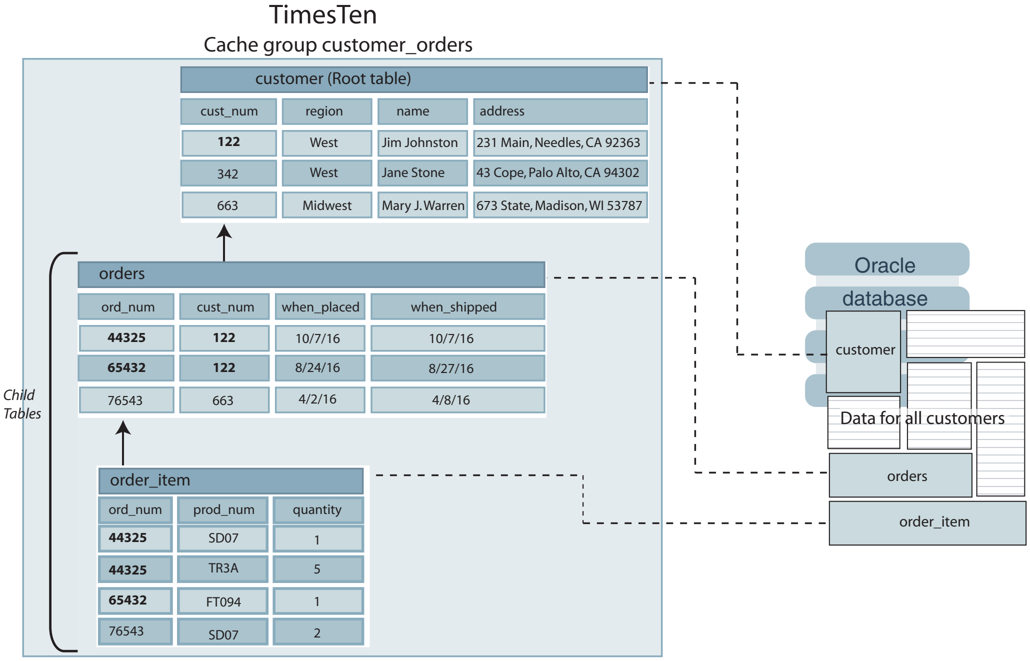
Cache instance

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

Figure 1–2 shows three tables in the `customer_orders` cache group. The root table is `customer`. `orders` and `order_item` are child tables. The cache instance identified by the row with the value `122` in the `cust_num` primary key column of the customer table includes:

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

Figure 1–2 Multiple-table cache group



customer (Root table)

cust_num	region	name	address
122	West	Jim Johnston	231 Main, Needles, CA 92363
342	West	Jane Stone	43 Cope, Palo Alto, CA 94302
663	Midwest	Mary J.Warren	673 State, Madison, WI 53787

orders

ord_num	cust_num	when_placed	when_shipped
44325	122	10/7/16	10/7/16
65432	122	8/24/16	8/27/16
76543	663	4/2/16	4/8/16

order_item

ord_num	prod_num	quantity
44325	SD07	1
44325	TR3A	5
65432	FT094	1
76543	SD07	2

Cache group types

The most commonly used types of cache groups are:

- Read-only cache group

  A read-only cache group enforces a caching behavior in which committed updates on cached tables in the Oracle database are automatically refreshed to the cache tables in the TimesTen database. Using a read-only cache group is suitable for reference data that is heavily accessed by applications.

  See "Read-only cache group" on page 4-8 for details about read-only cache groups.

- Asynchronous WriteThrough (AWT) cache group

  An AWT cache group enforces a caching behavior in which committed updates on cache tables in the TimesTen database are automatically propagated to the cached tables in the Oracle database in asynchronous fashion. Using an AWT cache group is suitable for high speed data capture and online transaction processing.

See "Asynchronous WriteThrough (AWT) cache group" on page 4-11 for details about AWT cache groups.

Other types of cache groups include:

- Synchronous writethrough (SWT) cache group

  An SWT cache group enforces a caching behavior in which committed updates on cache tables in the TimesTen database are automatically propagated to the cached tables in the Oracle database in synchronous fashion.

  See "Synchronous WriteThrough (SWT) cache group" on page 4-25 for details about SWT cache groups.

- User managed cache group

  A user managed cache group defines customized caching behavior.

  For example, you can define a cache group that does not use automatic refresh or automatic propagation where committed updates on the cache tables are manually propagated or flushed to the cached Oracle Database tables.

  You can also define a cache group that uses both automatic propagation in synchronous fashion on every table and automatic refresh.

  See "User-managed cache group" on page 4-27 for details about user managed cache groups.

Transmitting updates between the TimesTen and Oracle databases

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

As shown in Figure 1–3, propagate and flush are operations that transmit committed updates on cache tables in the TimesTen database to the cached tables in the Oracle database. Flush is a manual operation and propagate is an automatic operation.

Load, refresh, and autorefresh are operations that transmit committed updates on cached tables in the Oracle database to the cache tables in the TimesTen database. Load and refresh are manual operations; autorefresh is an automatic operation.

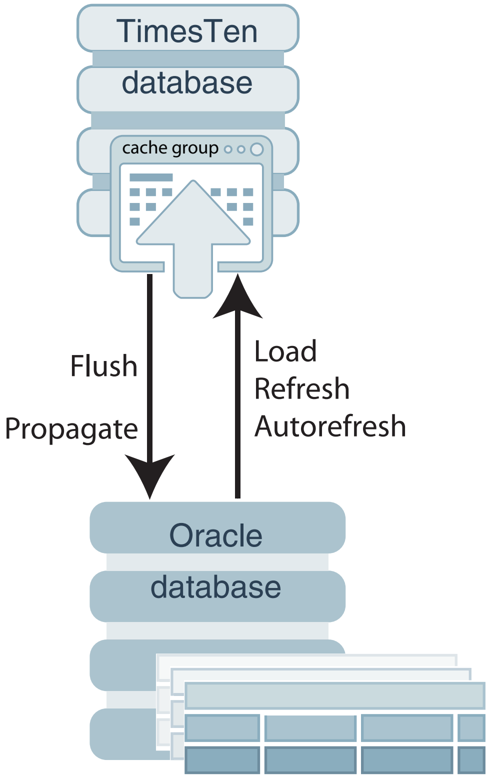
See "Flushing a user managed cache group" on page 5-16 for information about the `FLUSH CACHE GROUP` statement which can only be issued on a user managed cache group.

See "Asynchronous WriteThrough (AWT) cache group" on page 4-11 and "Synchronous WriteThrough (SWT) cache group" on page 4-25 for details about how a propagate operation is processed on AWT and SWT cache groups, respectively.

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

See "AUTOREFRESH cache group attribute" on page 4-34 for details about an autorefresh operation.

Figure 1–3 Transmitting committed updates between the TimesTen and Oracle databases



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

Cache groups are categorized as either explicitly loaded or dynamic.

- In an explicitly loaded cache group, cache instances are loaded manually into the TimesTen cache tables from an Oracle database by using a load or refresh operation or automatically by using an autorefresh operation. The cache tables are loaded before operations such as queries are performed on the tables. An explicitly loaded cache group is appropriate when the set of data to cache is static and can be predetermined before applications begin performing operations on the cache tables. By default, cache groups are explicitly loaded unless they are defined as dynamic.
- In a dynamic cache group, cache instances are loaded into the TimesTen cache tables on demand from an Oracle database using a dynamic load operation or manually using a load operation. A manual refresh or an autorefresh operation on a dynamic cache group can result in existing cache instances being updated or deleted, but committed updates on Oracle Database data that are not being cached do not result in new cache instances being loaded into its cache tables. A dynamic cache group is appropriate when the set of data to cache is small and should not be preloaded from Oracle Database before applications begin performing operations on the cache tables.

See "Transmitting updates between the TimesTen and Oracle databases" on page 1-4 for details about cache group load and refresh operations.

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

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

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

See "Dynamic cache groups" on page 4-51 for more information about dynamic cache groups.

High availability caching solution

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

An active standby pair provides for fault tolerance of a TimesTen database. Oracle Real Application Clusters (Oracle RAC) and Data Guard provides for high availability of an Oracle database.

See "Replicating cache tables" on page 4-53 for information on configuring replication of cache tables.

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

See "Using TimesTen Cache with Data Guard" on page 10-1 for more information on TimesTen Cache and Data Guard.

2

Getting Started

This chapter illustrates the creation and use of cache groups by demonstrating how to create an explicitly loaded read-only local cache group and a dynamic updatable cache group. In addition, this chapter describes how to populate cache tables, and how to observe the transfer of updates between the cache tables in the TimesTen database and the cached tables in the Oracle database.

- Setting up the Oracle Database and TimesTen Classic systems
- Creating cache groups
- Performing operations on the read-only cache group
- Performing operations on a dynamically updatable cache group
- Cleaning up the TimesTen Classic and Oracle Database systems

Setting up the Oracle Database and TimesTen Classic systems

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

Note: It is best to have the TimesTen and Oracle databases on different systems, to avoid resource contention between them. TimesTen, being an in-memory database, uses a significant amount of memory. It may also use a significant amount of CPU time and generate a significant amount of I/O, depending on the workload.

TimesTen Cache must know which Oracle database to connect to, which credentials to use when connecting to the Oracle database and which users own the tables in both TimesTen and Oracle databases.

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

Create users in the Oracle database

Before you can use TimesTen Cache, you must create the following users on the Oracle Database:

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

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

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

Use SQL*Plus to create a default tablespace to be used for storing TimesTen Cache management objects that should not be shared with other applications. While you may also store Oracle base tables that are cached in a TimesTen database, we strongly recommend that this tablespace be used solely by TimesTen Classic for cache management.

In the following example, the name of the default tablespace is `cachetblsp`:

```
SQL> CREATE TABLESPACE cachetblsp;
```

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

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

Then use SQL*Plus to perform the following operations:

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

Pass the cache administration user name as arguments to the `grantCacheAdminPrivileges.sql` script. In the following example, the cache administration user name is `cacheuser`:

Note: See the comments in the `grantCacheAdminPrivileges.sql` script for the required privileges by the user who executes this script and the privileges that this user grants to the cache administration user.

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

The privileges that the cache administration user requires depend on the types of cache groups you create and the operations that you perform on the cache groups.

See "Create the Oracle database users" on page 3-2 for more information about the `timesten` user, the schema users, and the cache administration user.

Create a DSN for the TimesTen database

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

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

On UNIX or Linux, in the `.odbc.ini` file that resides in your home directory or the *`timesten_home`*`/conf/sys.odbc.ini` file, create a TimesTen DSN `cache1` and set the following connection attributes:

```
[cache1]
DataStore=/users/OracleCache/ttcache
PermSize=64
OracleNetServiceName=oracledb
DatabaseCharacterSet=AL32UTF8
```

On Windows, create a TimesTen user DSN or system DSN `cache1` and set the following connection attributes:

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

Use the default settings for all the other connection attributes.

See "Define a DSN for the TimesTen database" on page 3-6 for more information about defining a DSN for a TimesTen database that caches data from an Oracle database.

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

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

Create users in the TimesTen database

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

- A *cache manager user* performs cache group operations. The TimesTen cache manager user must have the same name as a companion Oracle Database user that can access the cached Oracle Database tables. The companion Oracle Database user can be the cache administration user, a schema user, or some other existing user. For ease of use, making the cache administration user the companion Oracle Database user of the cache manager user is preferable. The password of the cache manager user can be different than the password of the companion Oracle Database user with the same name.

Note: See "Create the TimesTen users" on page 3-7 for more details on the cache manager user and its companion Oracle Database user.

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

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

Start the `ttIsql` utility on the TimesTen Classic system from an operating system shell or command prompt as the instance administrator, and connect to the `cache1` DSN to create the TimesTen database that is to be used to cache data from an Oracle database:

```
% ttIsql cache1
```

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

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

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

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

The privileges that the cache manager user requires depend on the types of cache groups you create and the operations that you perform on the cache groups. See "Create the TimesTen users" on page 3-7 for more information about the cache manager user and the cache table users.

See "Authentication in TimesTen" and "Authorization in TimesTen" in *Oracle TimesTen In-Memory Database Security Guide* for more information about TimesTen users and privileges.

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

Start the `ttIsql` utility and connect to the `cache1` DSN as the cache manager user. In the connection string, specify the cache manager user name in the `UID` connection attribute. Specify the cache manager user's password in the `PWD` connection attribute. Specify the password of its companion Oracle user (created with the same name to be the companion user to the cache manager) in the `OraclePWD` connection attribute within the connection string. In this example, the cache administration user is the companion user to the cache manager user and so its password is provided.

```
% ttIsql "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
```

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

```
Command> call ttCacheUidPwdSet('cacheuser','oracle');
```

The cache administration user name and password need to be set only once in a TimesTen database. See "Set the cache administration user name and password" on page 3-9 for information on how to use this setting by the TimesTen database.

Creating cache groups

This section creates a read-only cache group (as shown in Figure 2–1) and an Asynchronous WriteThrough (AWT) cache group (as shown in Figure 2–2).

Figure 2–1 Single-table read-only cache group

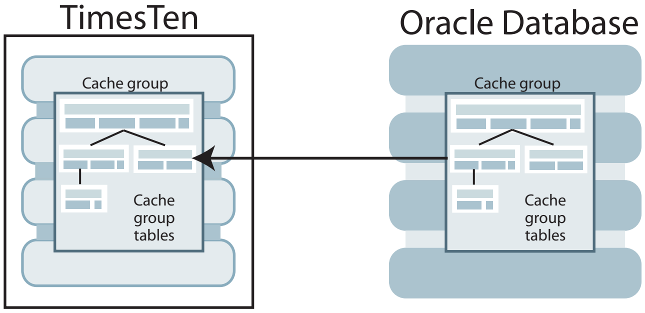
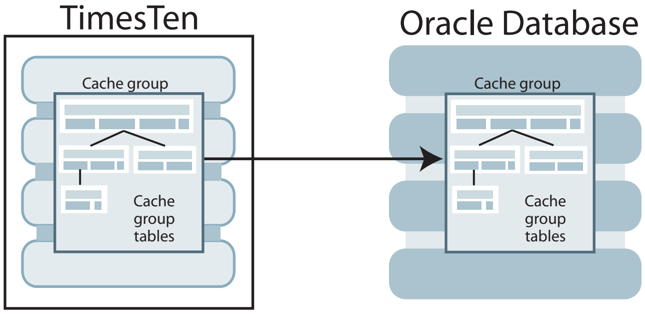


Figure 2–2 Single-table WriteThrough cache group



Complete the following tasks to create a read-only cache group and an AWT cache group:

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

Create the Oracle Database tables to be cached

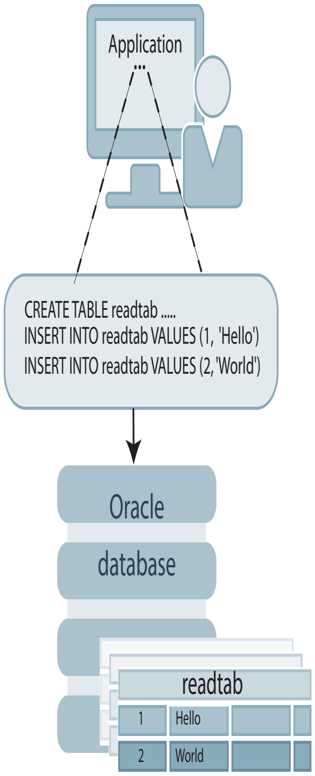
Start SQL*Plus and connect to the Oracle database as the schema user:

```
% sqlplus oratt/oracle
```

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

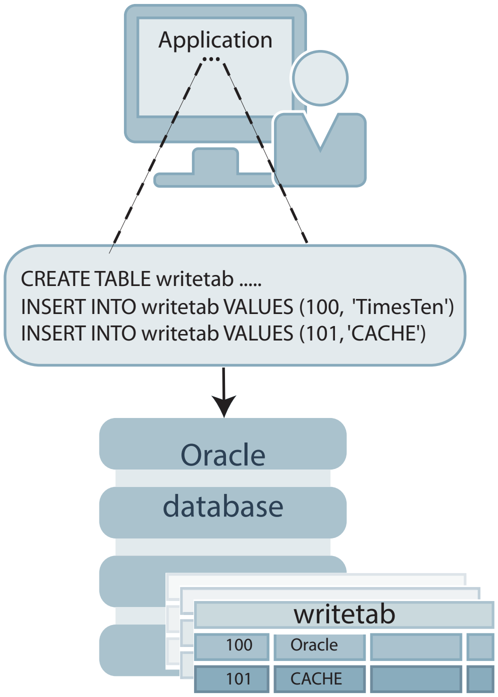
```
SQL> CREATE TABLE readtab (keyval NUMBER NOT NULL PRIMARY KEY, str VARCHAR2(32));
SQL> CREATE TABLE writetab (pk NUMBER NOT NULL PRIMARY KEY, attr VARCHAR2(40));
```

Figure 2–3 Creating an Oracle Database table to be cached in a read-only cache group



```
CREATE TABLE readtab .....
INSERT INTO readtab VALUES (1, 'Hello')
INSERT INTO readtab VALUES (2, 'World')
```

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



```
CREATE TABLE writetab .....
INSERT INTO writetab VALUES (100, 'TimesTen')
INSERT INTO writetab VALUES (101,'CACHE')
```

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

```
SQL> INSERT INTO readtab VALUES (1, 'Hello');
SQL> INSERT INTO readtab VALUES (2, 'World');

SQL> INSERT INTO writetab VALUES (100, 'TimesTen');
SQL> INSERT INTO writetab VALUES (101, 'CACHE');
SQL> COMMIT;
```

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

```
SQL> GRANT SELECT ON readtab TO cacheuser;

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

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

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

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

Start the cache agent

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

```
Command> call ttCacheStart;
```

See "Managing the cache agent" on page 3-11 for more information about starting the cache agent.

Create the cache groups

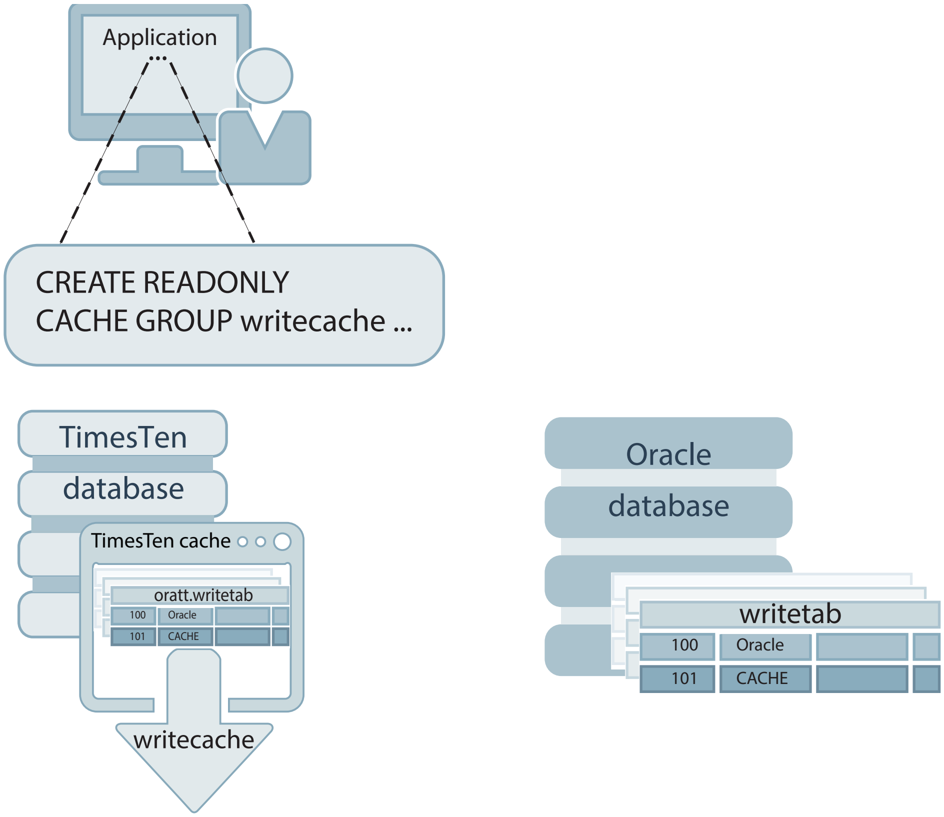
As the cache manager user, use the `ttIsql` utility to create a read-only cache group `readcache` that caches the Oracle Database `oratt.readtab` table and a dynamic AWT cache group `writecache` that caches the Oracle Database `oratt.writetab` table:

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

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

The cache groups `readcache` and `writecache`, and their respective cache tables `oratt.readtab` and `oratt.writetab`, whose owners and names are identical to the cached Oracle Database tables, are created in the TimesTen database. Figure 2–5 shows that the `writecache` cache group caches the `oratt.writetab` table.

Figure 2–5 Creating an Asynchronous WriteThrough cache group



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

```
Command> cachegroups;

Cache Group CACHEUSER.READCACHE:

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

  Root Table: ORATT.READTAB
  Table Type: Read Only

Cache Group CACHEUSER.WRITECACHE:

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

  Root Table: ORATT.WRITETAB
  Table Type: Propagate

2 cache groups found.
```

See "Read-only cache group" on page 4-8 for more information about read-only cache groups.

See "Asynchronous WriteThrough (AWT) cache group" on page 4-11 for more information about AWT cache groups.

See "Dynamic cache groups" on page 4-51 for more information about dynamic cache groups.

Start the replication agent for the AWT cache group

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

```
Command> call ttRepStart;
```

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

See "Managing the replication agent" on page 4-13 for more information about starting the replication agent.

Performing operations on the read-only cache group

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

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

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

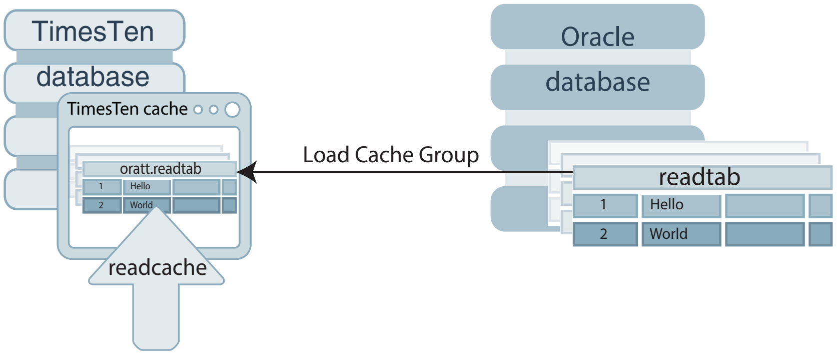
Manually load the cache group

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

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

Figure 2–6 shows that the Oracle Database data is loaded into the `oratt.readtab` cache table.

Figure 2–6 Loading a read-only cache group



Start the `ttIsql` utility and connect to the `cache1` DSN as the instance administrator. Use `ttIsql` to grant the `SELECT` privilege on the `oratt.readtab` cache table to the cache manager user so that this user can issue a `SELECT` query on this table.

```
% ttIsql cache1
Command> GRANT SELECT ON oratt.readtab TO cacheuser;
Command> exit
```

Start the `ttIsql` utility and connect to the `cache1` DSN as the cache manager user, including the cache manager user password and the password of its companion Oracle user. Use `ttIsql` to query the contents of `oratt.readtab` cache table.

```
% ttIsql "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> SELECT * FROM oratt.readtab;
< 1, Hello >
< 2, World >
2 rows found.
```

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

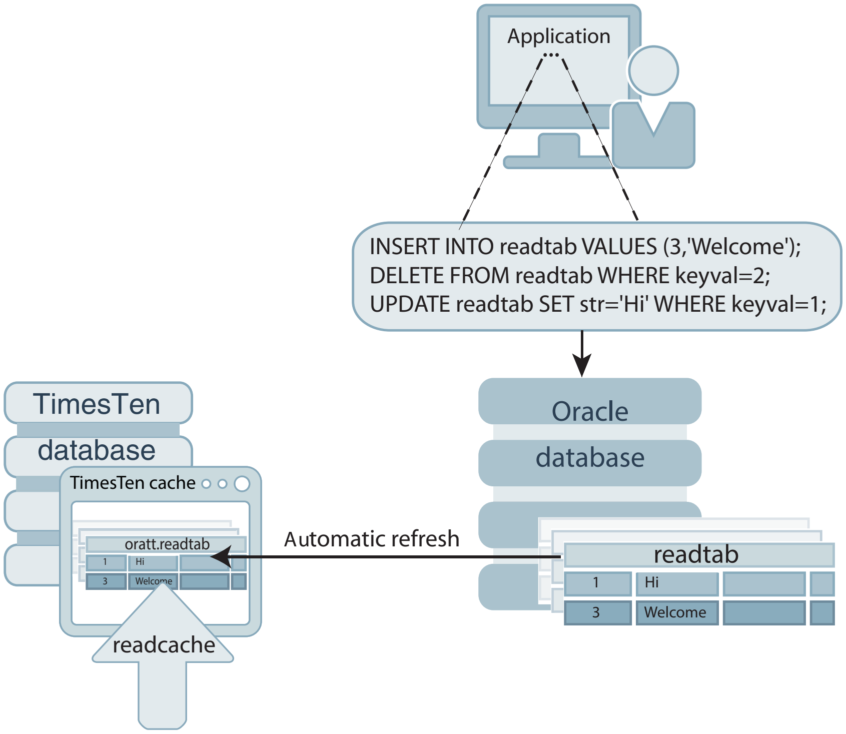
Update the cached Oracle Database table

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

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

Since the read-only cache group was created specifying autorefresh with an interval of 5 seconds, the `oratt.readtab` cache table in the `readcache` cache group is automatically refreshed after 5 seconds with the committed updates on the cached Oracle Database `oratt.readtab` table as shown in Figure 2–7.

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



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

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

See "AUTOREFRESH cache group attribute" on page 4-34 for more information about automatically refreshing cache groups.

Performing operations on a dynamically updatable cache group

This section shows how to dynamically load an AWT cache group. Then it shows committed updates on the TimesTen cache table being automatically propagated to the cached Oracle Database table.

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

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

Dynamically load the cache group

Start the `ttIsql` utility and connect to the `cache1` DSN as the instance administrator. Use `ttIsql` to grant the `SELECT` privilege on the `oratt.writetab` cache table to the cache manager user so that this user can issue a dynamic load `SELECT` statement on this table.

```
% ttIsql cache1
Command> GRANT SELECT ON oratt.writetab TO cacheuser;
Command> exit
```

Start the `ttIsql` utility and connect to the `cache1` DSN as the cache manager user, including the cache manager user password and the password of its companion Oracle user. Use `ttIsql` to load a cache instance on demand from the Oracle Database `oratt.writetab` table to the TimesTen `oratt.writetab` cache table in the `writecache` cache group.

```
% ttIsql "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> SELECT * FROM oratt.writetab WHERE pk=100;
< 100, TimesTen >
1 row found.
Command> exit
```

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

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

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

Update the TimesTen cache table

Start the `ttIsql` utility and connect to the `cache1` DSN as the instance administrator. Use `ttIsql` to grant the `INSERT`, `DELETE`, and `UPDATE` privileges on the `oratt.writetab` cache table to the cache manager user so that this user can perform updates on this table.

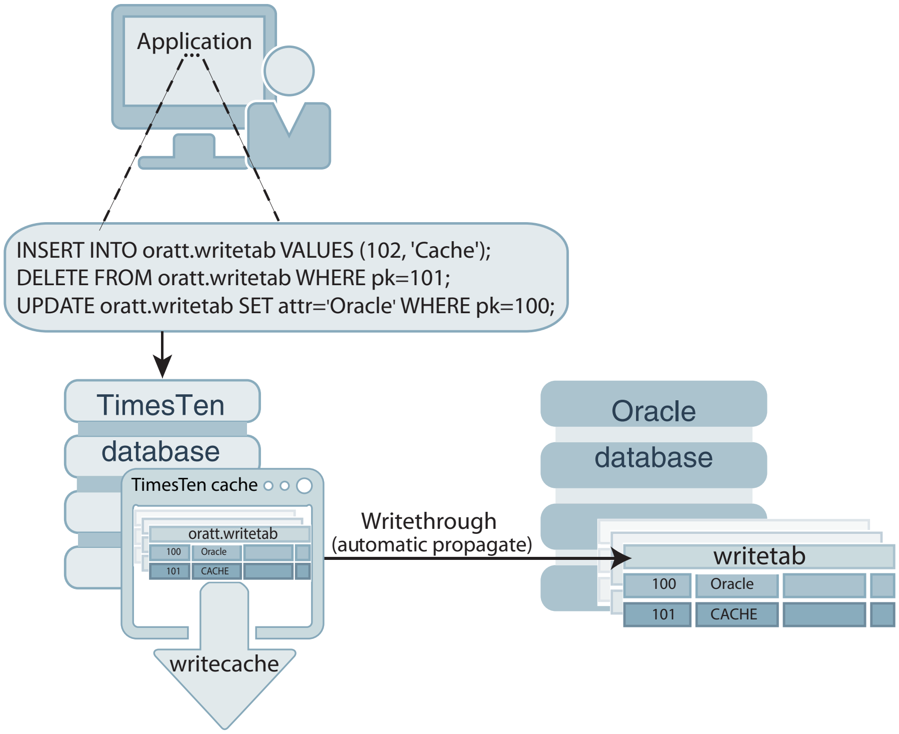
```
% ttIsql cache1
Command> GRANT INSERT ON oratt.writetab TO cacheuser;
Command> GRANT DELETE ON oratt.writetab TO cacheuser;
Command> GRANT UPDATE ON oratt.writetab TO cacheuser;
Command> exit
```

Start the `ttIsql` utility and connect to the `cache1` DSN as the cache manager user. Use `ttIsql` to insert a new row, delete an existing row, and update an existing row in the `oratt.writetab` cache table, and commit the changes.

```
% ttIsql "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> INSERT INTO oratt.writetab VALUES (102, 'Cache');
Command> DELETE FROM oratt.writetab WHERE pk=101;
Command> UPDATE oratt.writetab SET attr='Oracle' WHERE pk=100;
Command> COMMIT;
Command> exit
```

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

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



```
INSERT INTO oratt.writetab VALUES (102, 'Cache');
DELETE FROM oratt.writetab WHERE pk=101;
UPDATE oratt.writetab SET attr='Oracle' WHERE pk=100;
```

As the Oracle Database schema user, use SQL*Plus to query the contents of the `writetab` table:

```
SQL> SELECT * FROM writetab;

        PK ATTR
---------- ------------------------------
       100 Oracle
       102 Cache

SQL> exit
```

Cleaning up the TimesTen Classic and Oracle Database systems

Complete the following tasks to restore the TimesTen Classic and Oracle Database systems to their original state before creating cache groups:

1. Stop the replication agent.
2. Drop the cache groups.
3. Stop the cache agent and destroy the TimesTen database.
4. Drop the Oracle Database users and their objects.

Stop the replication agent

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

```
Command> call ttRepStop;
Command> exit
```

See "Managing the replication agent" on page 4-13 for more information about stopping the replication agent.

Drop the cache groups

Start the `ttIsql` utility and connect to the `cache1` DSN as the instance administrator. Use `ttIsql` to grant the `DROP ANY TABLE` privilege to the cache manager user so that this user can drop the underlying cache tables when dropping the cache groups.

```
% ttIsql cache1
Command> GRANT DROP ANY TABLE TO cacheuser;
Command> exit
```

Start the `ttIsql` utility and connect to the `cache1` DSN as the cache manager user. Use `ttIsql` to drop the `readcache` read-only cache group and the `writecache` AWT cache group.

```
% ttIsql "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> DROP CACHE GROUP readcache;
Command> DROP CACHE GROUP writecache;
```

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

See "Dropping a cache group" on page 8-1 for more information about dropping cache groups.

Stop the cache agent and destroy the TimesTen database

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

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

See "Managing the cache agent" on page 3-11 for more information about stopping the cache agent.

Then use the `ttDestroy` utility to connect to the `cache1` DSN and destroy the TimesTen database:

```
% ttDestroy cache1
```

Drop the Oracle Database users and their objects

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

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

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

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

```
SQL> DROP ROLE TT_CACHE_ADMIN_ROLE;
```

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

```
SQL> DROP TABLESPACE cachetblsp INCLUDING CONTENTS AND DATAFILES;
SQL> exit
```

3

Setting Up a Caching Infrastructure

Before you can start caching Oracle Database data in a TimesTen database, perform these tasks for setting up the TimesTen Classic and Oracle Database systems:

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

Configuring your system to cache Oracle Database data in TimesTen Classic

Note: See the Platforms section in the *Oracle TimesTen In-Memory Database Release Notes* (`README.htm`) in your installation directory to find out which Oracle Database Server releases are supported by the TimesTen In-Memory Database.

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

Then, install TimesTen. Instructions for installing TimesTen are described in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.

Note: From a product perspective, "TimesTen Cache" is used interchangeably with "TimesTen Classic" because the TimesTen Cache product option is included with TimesTen Classic.

TimesTen Classic does not support Oracle Name Server for Windows clients.

TimesTen Cache environment variables for UNIX or Linux

The shared library search path environment variable such as `LD_LIBRARY_PATH` or `SHLIB_PATH` must include the *`timesten_home`*`/install/lib` directories. For more information, see "Shared library path environment variable" in *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.

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

In the following example, the `timesten_home` directory is the `/timesten/myinstance` directory and TimesTen Classic is installed in the `/timesten/myinstance/install` directory:

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

TimesTen Cache environment variables for Microsoft Windows

The `PATH` system environment variable must include the following directories:

- *`Oracle_install_dir`*`\bin`
- *`timesten_home`*`\install\lib`
- *`timesten_home`*`\bin`

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

```
PATH=C:\oracle\ora112\bin;C:\timesten\myinstance\install\lib;C:\timesten\myinstanc
e\bin
```

Configuring the Oracle database to cache data in TimesTen Classic

The following sections describe the tasks that must be performed on the Oracle database by the `sys` user:

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

Create the Oracle database users

You must create a default tablespace and a user `timesten` that is to own the Oracle Database tables that store information about cache operations. This tablespace is used for storing TimesTen Cache management objects that should not be shared with other applications. While you may also store Oracle base tables that are cached in a TimesTen database, we strongly recommend that this tablespace be used solely by the TimesTen database for cache management.

See "Managing a caching environment with Oracle Database objects" on page 6-7 for a list of Oracle Database tables owned by the `timesten` user.

Example 3–1 Creating the timesten user and its tables

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

```
% cd timesten_home/install/oraclescripts
% sqlplus sys as sysdba
Enter password: password
SQL> CREATE TABLESPACE cachetblsp;
```

Create or designate one or more users to own Oracle Database tables that are to be cached in a TimesTen database. These users are the schema users. These may be existing users or new users. The tables to be cached may or may not already exist.

Example 3–2 Creating a schema user

As the `sys` user, the following SQL*Plus example creates a schema user `oratt`.

```
SQL> CREATE USER oratt IDENTIFIED BY oracle;
```

Next, you must create a user that creates, owns, and maintains Oracle Database objects that store information used to manage the cache environment for a TimesTen database and enforce predefined behaviors of particular cache group types. We refer to this user as the cache administration user.

Note: Each TimesTen database can be managed by only a single cache administration user on the Oracle database. However, a single cache administration user can manage multiple TimesTen databases. You can specify one or more cache administration users where each manages one or more TimesTen databases.

For more details, see "Caching the same Oracle table on two or more TimesTen databases" on page 7-21.

Designate the tablespace that was created for the `timesten` user as the default tablespace for the cache administration user. This user creates tables in this tablespace that are used to store information about the cache environment and its cache groups. Other Oracle Database objects (such change log tables, replication metadata tables, and triggers) are used to enforce the predefined behaviors of autorefresh cache groups and AWT cache groups are created in the same tablespace. To create and manage these objects, the cache administration user must have a high level of privileges.

Note: If you create multiple cache administration users, each may use the same or different tablespace as their default tablespace.

See "Managing a caching environment with Oracle Database objects" on page 6-7 for a list of Oracle Database tables and triggers owned by the cache administration user.

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

Example 3–3 Creating the cache administration user

As the `sys` user, create a cache administration user `cacheuser`. In the following example, the default tablespace for the `cacheuser` user is `cachetblsp`.

Use SQL*Plus to create the cache administration user:

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

Grant privileges to the Oracle database users

The cache administration user must be granted a high level of privileges depending on the cache group types created and the operations performed on these cache groups. You can run the SQL*Plus script *`timesten_home`*`/install/oraclescripts/grantCacheAdminPrivileges.sql` as the `sys` user to grant the cache administration user the minimum set of privileges required to perform cache operations. For more information on this SQL script, see "Automatically create Oracle Database objects used to manage data caching" on page 3-4.

The entire list of privileges required for this user for each cache operation are listed in "Required privileges for the cache administration user and the cache manager user" on page A-3.

Automatically create Oracle Database objects used to manage data caching

TimesTen Classic can automatically create Oracle Database objects owned by the cache administration user, such as cache and replication metadata tables, change log tables, and triggers when particular cache environment and cache group operations are performed. Some of these objects are used to enforce the predefined behaviors of autorefresh cache groups and AWT cache groups.

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

> **Note:** Alternatively, you can manually create these objects as described in "Manually create Oracle Database objects used to manage data caching" on page 3-5 before performing any cache group operations if, for security purposes, you do not want to grant the `RESOURCE` or `CREATE ANY TRIGGER` privileges to the cache administration user required to automatically create these tables and triggers.

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

Example 3–4 Granting privileges to automatically create Oracle Database objects

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

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

```
SQL> @grantCacheAdminPrivileges "cacheuser"
SQL> exit
```

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

- The cache group is created with its autorefresh state set to `PAUSED` or `ON`.

- The cache group is created with its autorefresh state set to `OFF` and then altered to either `PAUSED` or `ON`.

Manually create Oracle Database objects used to manage data caching

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

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

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

To manually create the Oracle Database tables and triggers used to enforce the predefined behaviors of particular cache group types, run the SQL*Plus script *`timesten_home`*`/install/oraclescripts/initCacheAdminSchema.sql` as the `sys` user. These objects must be created before you can create autorefresh cache groups and AWT cache groups. The `initCacheAdminSchema.sql` script requires the cache administration user name as input.

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

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

As the `sys` user, run the `initCacheAdminSchema.sql` script to manually create Oracle Database objects used to enforce the predefined behaviors of autorefresh cache groups and AWT cache groups, and grant a limited set of privileges to the cache administration user. In the following example, the cache administration user name is `cacheuser`.

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

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

Other Oracle Database objects associated with Oracle Database tables that are cached in an autorefresh cache group are needed to enforce the predefined behaviors of these cache group types. See "Manually creating Oracle Database objects for autorefresh cache groups" on page 4-37 for details about how to create these additional objects after you create the cache group.

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

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

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

Configuring a TimesTen database to cache Oracle Database data

The following sections describe the operations that must be performed on the TimesTen database by the instance administrator or the cache manager user:

- Define a DSN for the TimesTen database
- Create the TimesTen users
- Grant privileges to the TimesTen users
- Set the cache administration user name and password

Define a DSN for the TimesTen database

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

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

- `PermSize` specifies the allocated size of the database's permanent region in MB. Set this value to at least 32 MB.
- `OracleNetServiceName` must be set to the net service name of the Oracle database instance.

  For Microsoft Windows systems, the net service name of the Oracle database instance must be specified in the **Oracle Net Service Name** field of the TimesTen Cache tab within the TimesTen ODBC Setup dialog box.
- `DatabaseCharacterSet` must match the Oracle database character set.

  You can determine the Oracle database character set by executing the following query in SQL*Plus as any user:

  ```
  SQL> SELECT value FROM nls_database_parameters
         WHERE parameter='NLS_CHARACTERSET';
  ```
- `UID` specifies the name of a cache user, such as the cache manager user, that has the same name as a companion Oracle Database user who can access the cached Oracle Database tables. The `UID` connection attribute can be specified in a Direct DSN, a Client DSN, or a connection string.
- `PWD` specifies the password of the TimesTen user specified in the `UID` connection attribute. The `PWD` connection attribute can be specified in a Direct DSN, a Client DSN, or a connection string.

- `OraclePWD` specifies the password of the companion Oracle Database user that has the same name as the TimesTen user specified in the `UID` connection attribute and can access the cached Oracle Database tables.

> **Note:** See "Create the TimesTen users" on page 3-7 for more details on the cache manager user and its companion Oracle Database user.

- `PassThrough` can be set to control whether statements are to be executed in the TimesTen database or passed through to be executed in the Oracle database. See "Setting a passthrough level" on page 5-17.
- `LockLevel` must be set to its default of 0 (row-level locking) because TimesTen Cache does not support database-level locking.
- `ReplicationApplyOrdering` and `CacheAWTParallelism` control parallel propagation of changes to TimesTen cache tables in an AWT cache group to the corresponding Oracle Database tables. See "Configuring parallel propagation to Oracle Database tables" on page 4-15.

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

The following example is the definition of the `cache1` DSN:

```
[cache1]
DataStore=/users/OracleCache/ttcache
PermSize=64
OracleNetServiceName=orcl
DatabaseCharacterSet=WE8ISO8859P1
```

Create the TimesTen users

First, you must create a user who performs cache group operations. We refer to this user as the *cache manager user*. The TimesTen cache manager user must have the same name as a companion Oracle Database user that can access the cached Oracle Database tables. For example, the companion Oracle Database user must have privileges to select from and update the cached Oracle Database tables. The companion Oracle Database user can be the cache administration user, a schema user, or some other existing user. For ease of use, making the cache administration user be the companion Oracle Database user of the cache manager user is preferable; however, if you are concerned with the high level of privileges assigned to the cache administration user, then choose another Oracle Database user as the companion Oracle user. The password of the cache manager user can be different than the password of the companion Oracle Database user with the same name.

> **Note:** You can create multiple cache manager users on a TimesTen database, such as one for each TimesTen DBA. However, you can only define a single cache administration user on the Oracle database for this particular TimesTen database. (You can use the same cache administration user for all TimesTen databases that connect to the Oracle database or define a separate cache administration user for each TimesTen database.) If you create multiple cache manager users, one or more of these users can use the cache administration user as its companion Oracle user.

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

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

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

Example 3–7 Creating the TimesTen users

In the following `ttIsql` utility example, create the TimesTen database by connecting to the `cache1` DSN as the instance administrator. Then create the cache manager user `cacheuser` whose name, in this example, is the same as the cache administration user, who will also act as the cache manager's companion Oracle user. Then, create a cache table user `oratt` whose name is the same as the Oracle Database schema user who is to own the Oracle Database tables to be cached in the TimesTen database.

```
% ttIsql cache1
Command> CREATE USER cacheuser IDENTIFIED BY timesten;
Command> CREATE USER oratt IDENTIFIED BY timesten;
```

Grant privileges to the TimesTen users

The privileges that the TimesTen users require depend on the types of cache groups you create and the operations that you perform on the cache groups. All of the privileges required for the TimesTen cache manager user for each cache operation are listed in "Required privileges for the cache administration user and the cache manager user" on page A-3.

Example 3–8 Granting privileges to the cache manager user

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

- Set the cache manager user and password (`CACHE_MANAGER`).
- Start or stop the cache agent and replication agent processes on the TimesTen database (`CACHE_MANAGER`).
- Set a cache agent start policy (`CACHE_MANAGER`).

- Set a replication agent start policy (`ADMIN`)
- Create cache groups to be owned by the cache manager user (`CREATE [ANY] CACHE GROUP`, inherited by the `CACHE_MANAGER` privilege; `CREATE [ANY] TABLE` to create the underlying cache tables which are to be owned by the `oratt` cache table user).
- Alter, load, refresh, flush, unload or drop a cache group requires the appropriate privilege:
    - `ALTER ANY CACHE GROUP`
    - `LOAD {ANY CACHE GROUP | ON` *cache_group_name*
    - `REFRESH {ANY CACHE GROUP | ON` *cache_group_name*
    - `FLUSH {ANY CACHE GROUP | ON` *cache_group_name*
    - `UNLOAD {ANY CACHE GROUP | ON` *cache_group_name*
    - `DROP ANY CACHE GROUP` and `DROP ANY TABLE`
- Required privileges for other cache operations, such as dynamic load, full autorefresh and asynchronous writethrough, are listed in "Required privileges for the cache administration user and the cache manager user" on page A-3.

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

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

Set the cache administration user name and password

You must set the cache administration user name and password in the TimesTen database before any cache group operation can be issued with the `ttCacheUidPwdSet` built-in procedure. The cache agent connects to the Oracle database as this user to create and maintain Oracle Database objects that store information used to enforce predefined behaviors of particular cache group types. In addition, both the cache and replication agents connect to the Oracle database with the credentials set with the `ttCacheUidPwdSet` built-in procedure to manage Oracle database operations.

Note: When you connect to the TimesTen database to work with AWT or read-only cache groups, TimesTen Classic uses the credentials set with the `ttCacheUidPwdSet` built-in procedure when connecting to the Oracle database on behalf of these cache groups.

When you connect to the TimesTen database to work with SWT or user managed cache groups or passthrough operations, TimesTen Classic connects to the Oracle database using the current user’s credentials as the user name and the `OraclePwd` connection attribute as the Oracle password. Thus, the correct user name and Oracle database password that should be used for connecting to the Oracle database must be set correctly in the connection string or with the connection attributes.

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

The cache administration user name cannot be changed if there are cache groups in the database. The cache groups must be dropped before you can drop and recreate the cache administration user. See "Changing cache user names and passwords" on page 6-21 for more details.

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

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

```
% ttIsql "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> call ttCacheUidPwdSet('cacheuser','oracle');
```

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

```
% ttAdmin -cacheUidPwdSet -cacheUid cacheuser -cachePwd oracle cache1
```

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

For more information about the utility, see "ttAdmin" in *Oracle TimesTen In-Memory Database Reference*.

Testing the connectivity between the TimesTen and Oracle databases

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

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

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

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

Example 3–10 Determining the cache administration user name setting

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

```
Command> call ttCacheUidGet;
```

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

```
% ttAdmin -cacheUidGet cache1
```

Managing the cache agent

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

Example 3–11 Starting the cache agent

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

```
Command> call ttCacheStart;
```

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

```
% ttAdmin -cacheStart cache1
```

Example 3–12 Stopping the cache agent

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

```
Command> call ttCacheStop;
```

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

```
% ttAdmin -cacheStop cache1
```

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

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

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

Set a cache agent start policy

A cache agent start policy determines how and when the cache agent process starts on a TimesTen database. The cache agent start policy can be set to:

- `manual`
- `always`
- `norestart`

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

When the start policy is set to `always`, the cache agent starts automatically when the TimesTen main daemon process starts. With the `always` start policy, the cache agent cannot be stopped when the main daemon is running unless the start policy is first changed to either `manual` or `norestart`. Then issue a manual stop by calling the `ttCacheStop` built-in procedure or running a `ttAdmin -cacheStop` utility command.

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

Setting the cache agent start policy to `norestart` means the cache agent must be started manually by calling the `ttCacheStart` built-in procedure or running a `ttAdmin -cacheStart` utility command, and stopped manually by calling the `ttCacheStop` built-in procedure or running a `ttAdmin -cacheStop` utility command.

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

Note: For more details, see "ttAdmin," "ttCachePolicySet," "ttCacheStart" and "ttCacheStop" in the *Oracle TimesTen In-Memory Database Reference*.

Example 3–13 Setting a cache agent start policy

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

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

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

```
% ttAdmin -cachePolicy norestart cache1
```

4

Defining Cache Groups

The following sections describe the different types of cache groups and how to define them:

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

Cache groups and cache tables

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

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

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

Before you define the cache group table, create the Oracle Database tables that are to be cached. Each table should be either:

- An Oracle Database table with a primary key on non-nullable columns. The TimesTen cache table primary key must be defined on the full Oracle Database table primary key. For example, if the cached Oracle Database table has a composite primary key on columns `c1`, `c2` and `c3`, the TimesTen cache table must also have a composite primary key on columns `c1`, `c2` and `c3`.

The following example shows how to create a cache group from an Oracle Database table with a composite primary key. Create the `job_history` table with a composite key on the Oracle database:

```
SQL> CREATE TABLE job_history
    (employee_id NUMBER(6) NOT NULL,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    job_id VARCHAR2(10) NOT NULL,
    department_id NUMBER(4),
    PRIMARY KEY(employee_id, start_date));
Table created.
```

Create the cache group on the TimesTen database with all columns of the composite primary key:

```
Command> CREATE WRITETHROUGH CACHE GROUP job_hist_cg
        FROM oratt.job_history
        (employee_id NUMBER(6) NOT NULL,
        start_date DATE NOT NULL,
        end_date DATE NOT NULL,
        job_id VARCHAR2(10) NOT NULL,
        department_id NUMBER(4),
        PRIMARY KEY(employee_id, start_date));
```

- An Oracle Database table with non-nullable columns upon which a unique index is defined on one or more of the non-nullable columns in the table. The TimesTen cache table primary key must be defined on all of the columns in the unique index. For example, if the unique index for the Oracle Database table is made up of multiple columns `c1`, `c2`, and `c3`, the TimesTen cache table must have a composite primary key on columns `c1`, `c2`, and `c3`.

  The following examples create Oracle Database unique indexes defined on tables with non-nullable columns.

```
SQL> CREATE TABLE regions(
      region_id NUMBER NOT NULL,
      region_name VARCHAR2(25));
Table created.
SQL> CREATE UNIQUE INDEX region_idx
      ON regions(region_id);
Index created.

SQL> CREATE TABLE sales(
      prod_id INT NOT NULL,
      cust_id INT NOT NULL,
      quantity_sold INT NOT NULL,
      time_id DATE NOT NULL);
Table created.
SQL> CREATE UNIQUE INDEX sales_index ON sales(prod_id, cust_id);
Index created.
```

  After creation of the Oracle Database table and unique index, you can create cache groups on a TimesTen database for these tables using the unique index columns as the primary key definition as shown below:

```
Command> CREATE WRITETHROUGH CACHE GROUP region_cg
  FROM oratt.regions
  (region_id NUMBER NOT NULL PRIMARY KEY,
   region_name VARCHAR2(25));
```

```
Command> CREATE WRITETHROUGH CACHE GROUP sales_cg
  FROM oratt.sales
  (prod_id INT NOT NULL, cust_id INT NOT NULL,
   quantity_sold INT NOT NULL, time_id DATE NOT NULL,
   PRIMARY KEY(prod_id, cust_id));
```

A TimesTen database can contain multiple cache groups. A cache group can contain one or more cache tables.

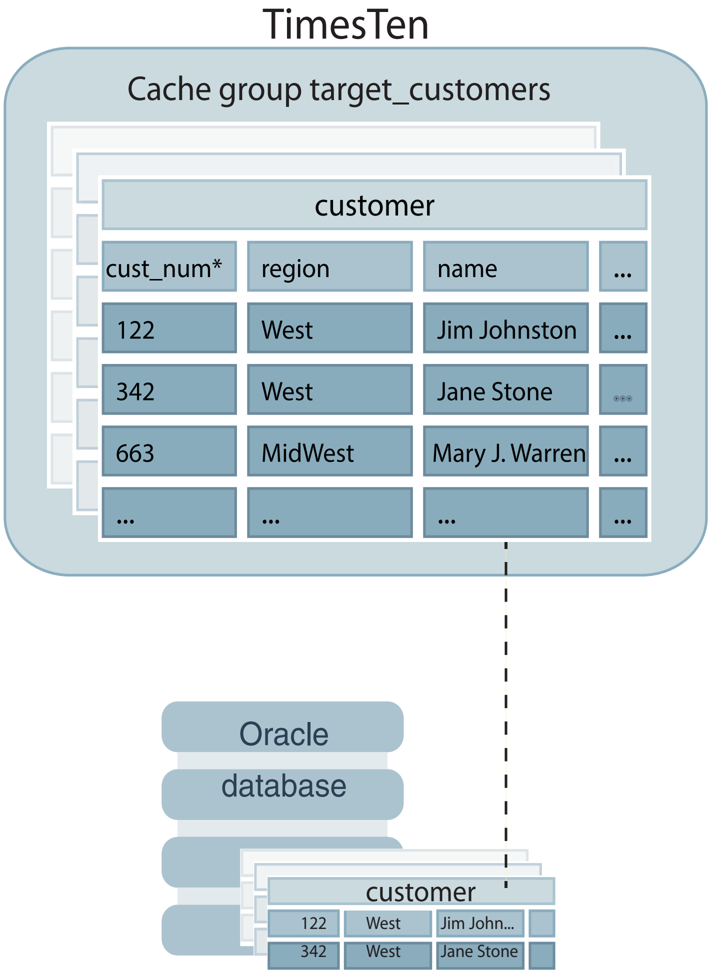
Creating indexes on a cache table in TimesTen can help speed up particular queries issued on the table in the same fashion as on a TimesTen regular table. You can create non-unique indexes on a TimesTen cache table. Do not create unique indexes on a cache table that do not match any unique index on the cached Oracle Database table. Otherwise, it can cause unique constraint failures in the cache table that do not occur in the cached Oracle Database table, and result in these tables in the two databases being no longer synchronized with each other when autorefresh operations are performed.

Single-table cache group

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

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

Figure 4–1 Cache group with a single table

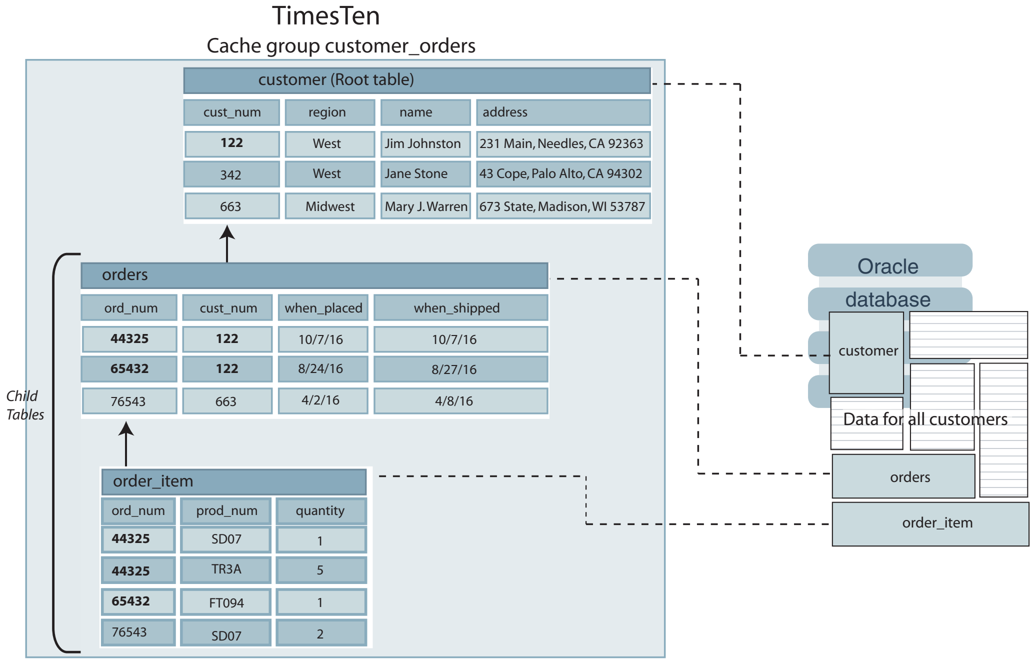


Multiple-table cache group

A multiple-table cache group is one that defines a root table and one or more child tables. A cache group can only contain one root table. Each child table must reference the primary key or a unique index of the root table or of another child table in the cache group using a foreign key constraint. Although tables in a multiple-table cache group must be related to each other in the TimesTen database through foreign key constraints, it is not required that the tables be related to each other in the Oracle database. The root table does not reference any table in the cache group with a foreign key constraint.

Figure 4–2 shows a multiple-table cache group `customer_orders` that caches the `customer`, `orders` and `order_item` tables. Each parent table in the `customer_orders` cache group has a primary key that is referenced by a child table through a foreign key constraint. The `customer` table is the root table of the cache group because it does not reference any table in the cache group with a foreign key constraint. The primary key of the root table is considered the primary key of the cache group. The `orders` table is a child table of the customer root table. The `order_item` table is a child table of the `orders` child table.

Figure 4–2 Cache group with multiple tables



customer (Root table)

cust_num	region	name	address
122	West	Jim Johnston	231 Main, Needles, CA 92363
342	West	Jane Stone	43 Cope, Palo Alto, CA 94302
663	Midwest	Mary J. Warren	673 State, Madison, WI 53787

orders

ord_num	cust_num	when_placed	when_shipped
44325	**122**	10/7/16	10/7/16
65432	**122**	8/24/16	8/27/16
76543	663	4/2/16	4/8/16

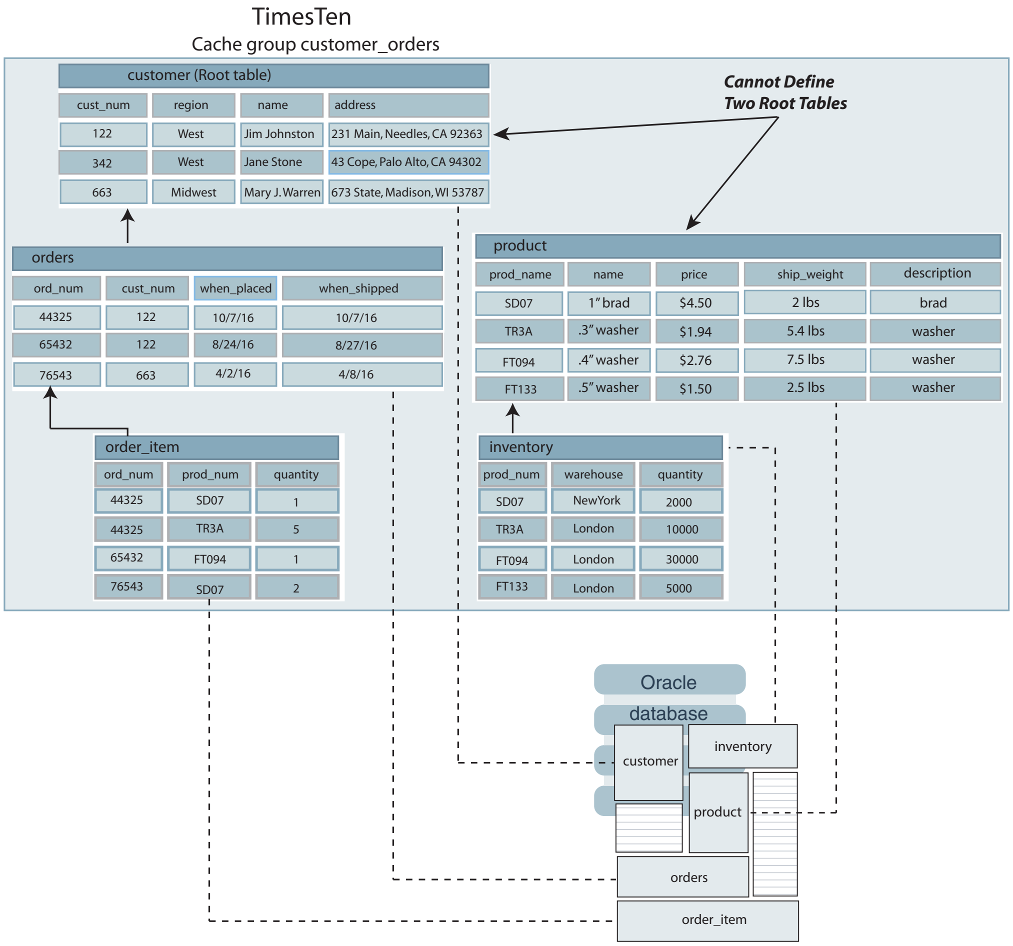
order_item

ord_num	prod_num	quantity
44325	SD07	1
44325	TR3A	5
65432	FT094	1
76543	SD07	2

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

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

Figure 4–3 Problem: Cache group contains two root tables



customer (Root table)

cust_num	region	name	address
122	West	Jim Johnston	231 Main, Needles, CA 92363
342	West	Jane Stone	43 Cope, Palo Alto, CA 94302
663	Midwest	Mary J. Warren	673 State, Madison, WI 53787

orders

ord_num	cust_num	when_placed	when_shipped
44325	122	10/7/16	10/7/16
65432	122	8/24/16	8/27/16
76543	663	4/2/16	4/8/16

order_item

ord_num	prod_num	quantity
44325	SD07	1
44325	TR3A	5
65432	FT094	1
76543	SD07	2

product

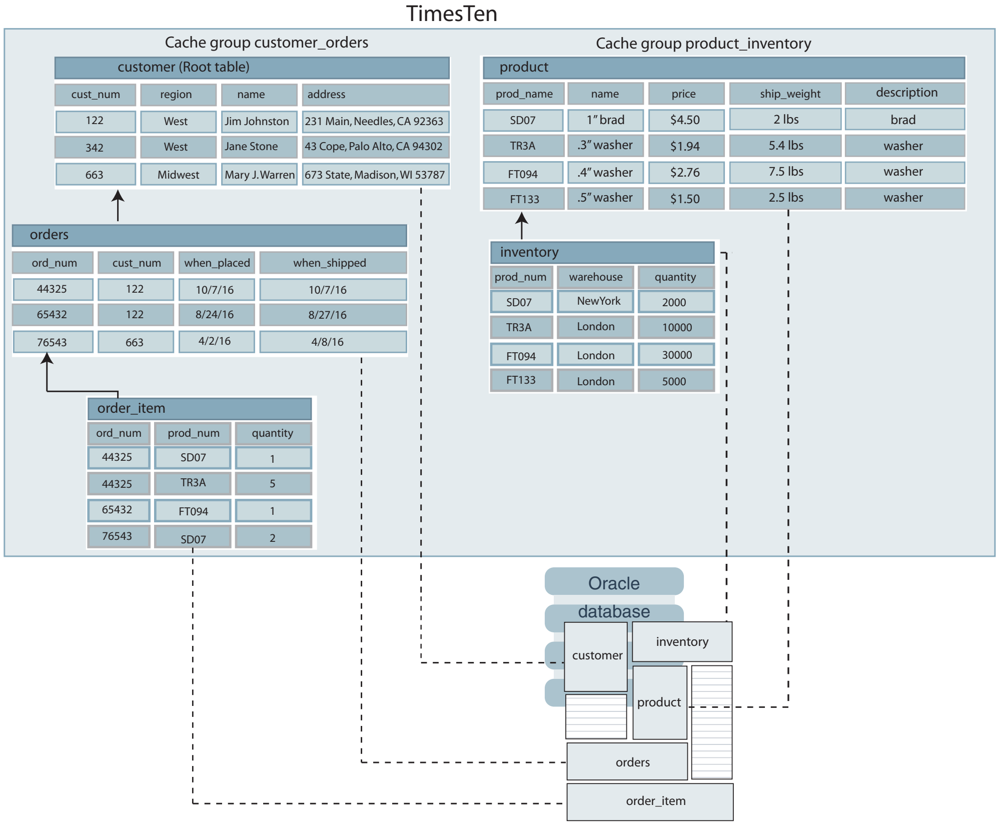
prod_name	name	price	ship_weight	description
SD07	1" brad	$4.50	2 lbs	brad
TR3A	.3" washer	$1.94	5.4 lbs	washer
FT094	.4" washer	$2.76	7.5 lbs	washer
FT133	.5" washer	$1.50	2.5 lbs	washer

inventory

prod_num	warehouse	quantity
SD07	NewYork	2000
TR3A	London	10000
FT094	London	30000
FT133	London	5000

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

Figure 4–4 Solution: Create two cache groups



customer (Root table)

cust_num	region	name	address
122	West	Jim Johnston	231 Main, Needles, CA 92363
342	West	Jane Stone	43 Cope, Palo Alto, CA 94302
663	Midwest	Mary J.Warren	673 State, Madison, WI 53787

orders

ord_num	cust_num	when_placed	when_shipped
44325	122	10/7/16	10/7/16
65432	122	8/24/16	8/27/16
76543	663	4/2/16	4/8/16

order_item

ord_num	prod_num	quantity
44325	SD07	1
44325	TR3A	5
65432	FT094	1
76543	SD07	2

product

prod_name	name	price	ship_weight	description
SD07	1" brad	$4.50	2 lbs	brad
TR3A	.3" washer	$1.94	5.4 lbs	washer
FT094	.4" washer	$2.76	7.5 lbs	washer
FT133	.5" washer	$1.50	2.5 lbs	washer

inventory

prod_num	warehouse	quantity
SD07	NewYork	2000
TR3A	London	10000
FT094	London	30000
FT133	London	5000

Creating a cache group

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

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

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

See "User-managed cache group" on page 4-27 for information about user-managed cache groups.

The following topics also apply to creating a cache group:

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

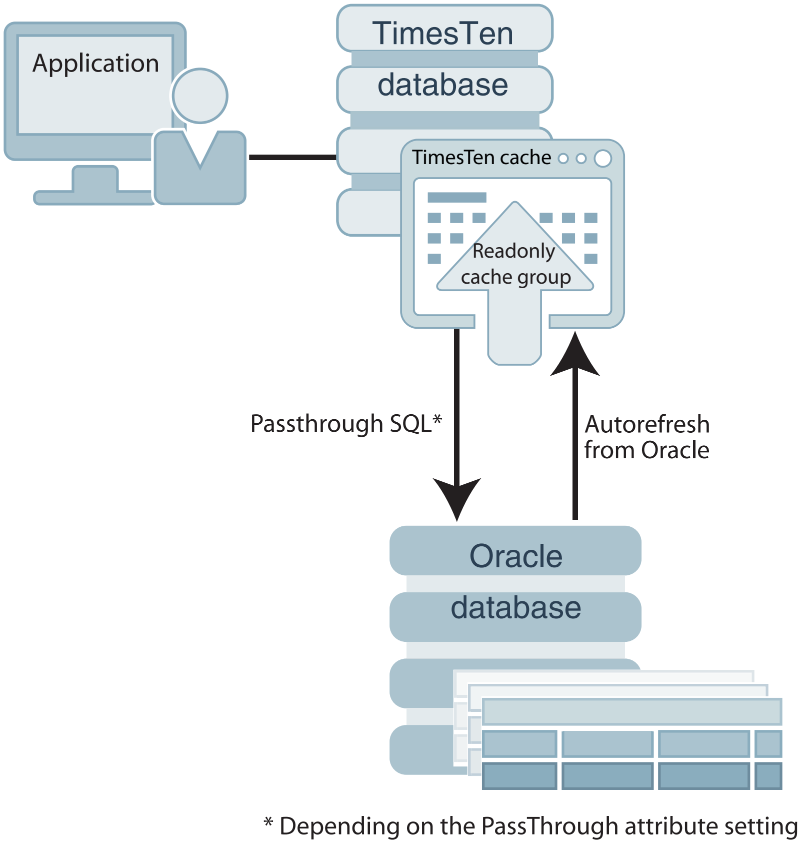
Cache groups must be created by and are owned by the cache manager user.

You cannot cache Oracle Database data in a temporary database.

Read-only cache group

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

Figure 4–5 Read-only cache group



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

Note: When TimesTen manages operations for read only cache groups, it connects to the Oracle database using the cache administration user name and password set with the `ttCacheUidPwdSet` built-in procedure. For more details on `ttCacheUidPwdSet`, see "Set the cache administration user name and password" on page 3-9.

The following are the definitions of the Oracle Database tables that are to be cached in the read-only cache groups that are defined in Example 4–1, Example 4–12, Example 4–15, Example 4–23 and Example 4–24. The Oracle Database tables are owned by the schema user `oratt`. The `oratt` user must be granted the `CREATE SESSION` and `RESOURCE` privileges before it can create tables.

```
CREATE TABLE customer
(cust_num NUMBER(6) NOT NULL PRIMARY KEY,
 region   VARCHAR2(10),
 name     VARCHAR2(50),
 address  VARCHAR2(100));

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

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

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

Example 4–1 Creating a read-only cache group

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

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

The cache tables in a read-only cache group cannot be updated directly. However, you can set the passthrough level to 2 to allow committed update operations issued on a TimesTen cache table to be passed through and processed on the cached Oracle Database table, and then have the updates be automatically refreshed into the cache table. See "Setting a passthrough level" on page 5-17.

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

If you manually created the Oracle Database objects used to enforce the predefined behaviors of an autorefresh cache group as described in "Manually create Oracle Database objects used to manage data caching" on page 3-5, you need to set the autorefresh state to `OFF` when creating the cache group.

Then you need to run the `ttIsql` utility's `cachesqlget` command to generate a SQL*Plus script used to create a log table and a trigger in the Oracle database for each Oracle Database table that is cached in the read-only cache group. See "Manually creating Oracle Database objects for autorefresh cache groups" on page 4-37 for information about how to create these objects.

Restrictions with read-only cache groups

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

- The cache tables on TimesTen cannot be updated directly.
- Only the `ON DELETE CASCADE` and `UNIQUE HASH ON` cache table attributes can be used in the cache table definitions.

  See "ON DELETE CASCADE cache table attribute" on page 4-43 for more information about the `ON DELETE CASCADE` cache table attribute.

  See "UNIQUE HASH ON cache table attribute" on page 4-44 for more information about the `UNIQUE HASH ON` cache table attribute.
- A `FLUSH CACHE GROUP` statement cannot be issued on the cache group.

  See "Flushing a user managed cache group" on page 5-16 for more information about the `FLUSH CACHE GROUP` statement.
- A `TRUNCATE TABLE` statement issued on a cached Oracle Database table is not automatically refreshed to the TimesTen cache table.
- A `LOAD CACHE GROUP` statement can only be issued on the cache group if the cache tables are empty, unless the cache group is dynamic.

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

  See "Dynamic cache groups" on page 4-51 for more information about dynamic cache groups.
- The autorefresh state must be `PAUSED` before you can issue a `LOAD CACHE GROUP` statement on the cache group, unless the cache group is dynamic, in which case the autorefresh state must be `PAUSED` or `ON`. The `LOAD CACHE GROUP` statement cannot contain a `WHERE` clause, unless the cache group is dynamic, in which case the `WHERE` clause must be followed by a `COMMIT EVERY` *n* `ROWS` clause.

  See "AUTOREFRESH cache group attribute" on page 4-34 for more information about autorefresh states.

  See "Using a WHERE clause" on page 4-40 for more information about `WHERE` clauses in cache group definitions and operations.
- The autorefresh state must be `PAUSED` before you can issue a `REFRESH CACHE GROUP` statement on the cache group. The `REFRESH CACHE GROUP` statement cannot contain a `WHERE` clause.

  See "Loading and refreshing a cache group" on page 5-2 for more information about the `REFRESH CACHE GROUP` statement.
- All tables and columns referenced in `WHERE` clauses when creating, loading or unloading the cache group must be fully qualified. For example:

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

- Least recently used (LRU) aging cannot be specified on the cache group, unless the cache group is dynamic where LRU aging is defined by default.

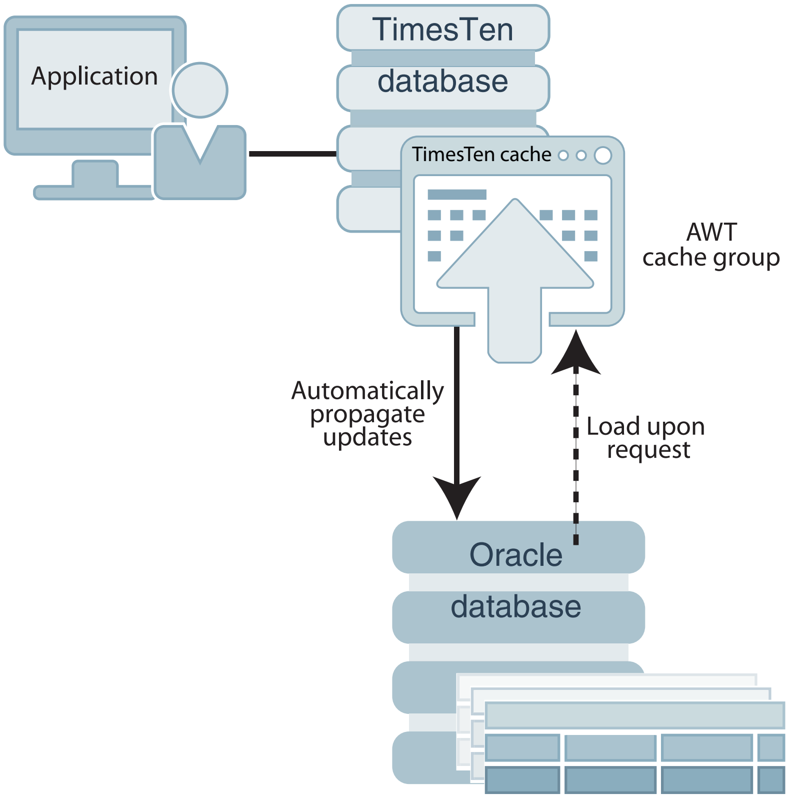
  See "LRU aging" on page 4-46 for more information about LRU aging.
- Read-only cache groups cannot cache Oracle Database views or materialized views.

Asynchronous WriteThrough (AWT) cache group

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

Note: You should avoid executing DML statements on Oracle Database tables cached in an AWT cache group. This can result in an error condition. For more information, see "Restrictions with AWT cache groups" on page 4-22.

Figure 4–6 AWT cache group



Since an AWT cache group propagates data from the TimesTen database to the Oracle database, any data modified by the user in the cached tables on the Oracle database is not automatically uploaded from the Oracle database to the TimesTen database. In this case, you must explicitly unload and then reload the AWT cache groups on TimesTen.

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

Execution of the `UNLOAD CACHE GROUP` statement for an AWT cache group waits until updates on the rows have been propagated to the Oracle database.

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

If there are updates from DML statements that you do not want propagated to the Oracle database, then you can disable propagation of committed updates (as a result of executing DML statements) within the current transaction to the Oracle database by setting the flag in the `ttCachePropagateFlagSet` built-in procedure to zero. After the flag is set to zero, the effects of executing any DML statements are never propagated to the back-end Oracle database. Thus, these updates exist only on the TimesTen database. You can then re-enable propagation by resetting the flag to one with the `ttCachePropagateFlagSet` built-in procedure. After the flag is set back to one, propagation of all committed updates to the Oracle database resumes. The propagation flag automatically resets to one after the transaction is committed or rolled back. See "ttCachePropagateFlagSet" in the *Oracle TimesTen In-Memory Database Reference* for more details.

Note: When TimesTen manages operations for AWT cache groups, it connects to the Oracle database using the cache administration user name and password set with the `ttCacheUidPwdSet` built-in procedure. For more details on `ttCacheUidPwdSet`, see "Set the cache administration user name and password" on page 3-9.

The following is the definition of the Oracle Database table that is to be cached in the AWT cache groups that are defined in Example 4–2, Example 4–16 and Example 4–18. The Oracle Database table is owned by the schema user `oratt`. The `oratt` user must be granted the `CREATE SESSION` and `RESOURCE` privileges before it can create tables.

```
CREATE TABLE customer
(cust_num NUMBER(6) NOT NULL PRIMARY KEY,
 region   VARCHAR2(10),
 name     VARCHAR2(50),
 address  VARCHAR2(100));
```

The companion Oracle Database user of the TimesTen cache manager user must be granted the `SELECT` privilege on the `oratt.customer` table in order for the cache manager user to create an AWT cache group that caches this table. The cache administration user must be granted the `INSERT`, `UPDATE` and `DELETE` Oracle Database privileges on the `oratt.customer` table for asynchronous writethrough operations to be applied to the Oracle Database.

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

Example 4–2 Creating an AWT cache group

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

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

The following sections describe configuration, behavior, and management for AWT cache groups:

- Managing the replication agent
- Configuring parallel propagation to Oracle Database tables
- What an AWT cache group does and does not guarantee
- Restrictions with AWT cache groups
- Reporting Oracle Database permanent errors for AWT cache groups

Managing the replication agent

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

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

Example 4–3 Starting the replication agent

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

```
% ttIsql "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> call ttRepStart;
```

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

```
% ttAdmin -repStart cache1
```

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

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

Example 4–4 Stopping the replication agent

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

```
Command> call ttRepStop;
```

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

```
% ttAdmin -repStop cache1
```

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

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

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

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

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

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

Example 4–5 Setting a replication agent start policy

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

```
% ttIsql cache1
Command> GRANT ADMIN TO cacheuser;
Command> exit
```

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

```
% ttIsql "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> call ttRepPolicySet('manual');
Command> exit
```

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

```
% ttAdmin -repPolicy always cache1
```

Since the AWT cache group uses the replication agent to asynchronously propagate transactions to the Oracle database, these transactions remain in the transaction log buffer and transaction log files until the replication agent confirms they have been fully processed by the Oracle database. You can monitor the propagation for these transactions with the `ttLogholds` built-in procedure. When you call the `ttLogHolds` built-in procedure, the description field contains "`_ORACLE`" to identify the transaction log hold for the AWT cache group propagation.

```
Command> call ttLogHolds();
```

```
< 0, 18958336, Checkpoint                    , cache1.ds0 >
< 0, 19048448, Checkpoint                    , cache1.ds1 >
< 0, 19050904, Replication                   , ADC6160529:_ORACLE >
3 rows found.
```

For more details on the `ttLogHolds` built-in procedure and how to monitor replication through bookmarks and log sequence numbers, see the "Show replicated log records" section in the *Oracle TimesTen In-Memory Database Replication Guide*.

Configuring parallel propagation to Oracle Database tables

To improve throughput for an AWT cache group, you can configure multiple threads that act in parallel to propagate and apply transactional changes to the Oracle database. Parallel propagation enforces transactional dependencies and applies changes in AWT cache tables to Oracle Database tables in commit order.

Parallel propagation is supported for AWT cache groups with the following configurations:

- AWT cache groups involved in an active standby pair replication scheme
- AWT cache groups in a single TimesTen database (without a replication scheme configuration)
- AWT cache groups configured with any aging policy

The following data store attributes enable parallel propagation and control the number of threads that operate in parallel to propagate changes from AWT cache tables to the corresponding Oracle Database tables:

- `ReplicationApplyOrdering` enables parallel propagation by default.
- `ReplicationParallelism` defines the number of transmitter threads on the source database and the number of receiver threads on the target database for parallel replication in a replication scheme. This value can be between 2 and 32 when used solely for parallel replication. The default is 1. In addition, the value of `ReplicationParellelism` cannot exceed half the value of `LogBufParallelism`.
- `CacheAWTParallelism`, when set, determines the number of threads used in parallel propagation of changes from AWT cache tables to the Oracle Database tables. Set this attribute to a number from 2 to 31. The default is 1.

Parallel propagation for an AWT cache group is configured with one of the following scenarios:

- `ReplicationApplyOrdering` is set to 0 and `ReplicationParallelism` is greater than 1.

  If you do not set `CacheAWTParallelism`, the number of threads that apply changes to Oracle Database is 2 times the setting for `ReplicationParallelism`. For example, if `ReplicationParallelism=3`, the number of threads that apply changes to Oracle Database tables is 6. In this case, `ReplicationParallelism` can only be set from 2 to 16; otherwise, twice the value would exceed the maximum number of 31 threads for parallel propagation. If the value is set to 16, the maximum number of threads defaults to 31.

- `ReplicationApplyOrdering` is set to 0, `ReplicationParallelism` is equal to or greater than 1, and `CacheAWTParallelism` is greater than 1. The value for `CacheAWTParallelism` must be greater than or equal to the value set for `ReplicationParallelism` and less than or equal to 31.

  If `CacheAWTParallelism` is not specified, then `ReplicationParallelism` is used to determine the number of threads that are used for parallel propagation to Oracle

Database. However, since this value is doubled for parallel propagation threads, you can only set `ReplicationParallelism` to a number from 2 to 16. If the value is set to 16, the maximum number of threads defaults to 31.

If both `ReplicationParallelism` and `CacheAWTParallelism` attributes are set, the value set in `CacheAWTParallelism` configures the number of threads used for parallel propagation. The setting for `CacheAWTParallelism` determines the number of apply threads for parallel propagation and the setting for `ReplicationParallelism` determines the number of threads for parallel replication. Thus, if `ReplicationParallelism` is set to 4 and `CacheAWTParallelism` is set to 6, then the number of threads that apply changes to Oracle Database tables is 6. This enables the number of threads used to be different for parallel replication and parallel propagation to Oracle Database tables.

> **Note:** For more information about parallel replication, see "Configuring parallel replication" in the *Oracle TimesTen In-Memory Database Replication Guide*.
>
> For more details on these data store attributes, see "ReplicationApplyOrdering," "ReplicationParallelism," and "CacheAWTParallelism" in the *Oracle TimesTen In-Memory Database Reference*.

These data store attributes are interrelated. Table 4–1 shows the result with the combination of the various possible attribute values.

Table 4–1 Results of Parallel Propagation Data Store Attribute Relationships

ReplicationApply Ordering	ReplicationParallelism	CacheAWTParallelism	Number of parallel propagation threads
Set to 0, which enables parallel propagation	Set to > 1 for multiple tracks and <= 16.	Not specified.	Set to twice the value of `ReplicationParallelism`.
Set to 0, which enables parallel propagation	Set to > 16 and <= 32 for multiple tracks.	Not specified.	Error is thrown. If `CacheAWTParallelism` is not set, then 2 times the value set in `ReplicationParallelism` specifies the number of threads. Thus, in this case, `ReplicationParallelism` cannot be greater than 16.
Set to 0, which enables parallel propagation	Set to > 1 and <= 32 for multiple tracks.	Set to >= to `ReplicationParallelism`.	Set to number specified by `CacheAWTParallelism`.
Set to 0, which enables parallel propagation	Set to > 1 and <= 32 for multiple tracks.	Set to < `ReplicationParallelism`.	Error is thrown at database creation. The `CacheAWTParallelism` must be set to a value greater than or equal to `ReplicationParallelism`.
Set to 0, which enables parallel propagation	Set to 1 or not specified. Single track.	Set to > 1	Set to number specified by `CacheAWTParallelism`.
Set to 1, which disables parallel propagation.	N/A	Set to > 1	Error is thrown at database creation, since parallelism is turned off, but `CacheAWTParallelism` is set to a value, expecting parallel propagation to be enabled.

Foreign keys in Oracle Database tables that are to be cached must have indexes created on the foreign keys. Consider these Oracle Database tables:

```
CREATE TABLE parent (c1 NUMBER PRIMARY KEY NOT NULL);
CREATE TABLE child (c1 NUMBER PRIMARY KEY NOT NULL,
                    c2 NUMBER REFERENCES parent(c1));
CREATE TABLE grchild (c1 NUMBER PRIMARY KEY NOT NULL,
                      c2 NUMBER REFERENCES parent(c1),
                      c3 NUMBER REFERENCES parent(c1));
```

These indexes must be created:

```
CREATE INDEX idx_1 ON child(c2);
CREATE INDEX idx_2 ON grchild(c2);
CREATE INDEX idx_3 ON grchild(c3);
```

Table constraint restrictions when using parallel propagation for AWT cache groups When you use parallel propagation for AWT cache groups, you must manually enforce data consistency. Any unique index, unique constraint, or foreign key constraint that exists on columns in the Oracle Database tables that are to be cached should also be created on the AWT cache tables within TimesTen. If you cannot create these constraints on the AWT cache tables and you have configured for parallel propagation, then TimesTen serializes any transactions with DML operations to any table with missing constraints. For example, if a unique index created on a table in the Oracle database cannot be created on the corresponding cached table in TimesTen, all transactions for this table are serialized.

TimesTen automatically checks for missing constraints on the Oracle database that are not cached on TimesTen when you issue any of the following SQL statements:

- When you create an AWT cache group with the `CREATE ASYNCHRONOUS CACHE GROUP` statement
- When you create a unique index on an AWT cache table with the `CREATE UNIQUE INDEX` statement
- When you drop a unique index on an AWT cache table with the `DROP INDEX` statement

---

Note: You can manually initiate a check for missing constraints with the `ttCacheCheck` built-in procedure. For example, TimesTen does not automatically check for missing constraints after a schema change on cached Oracle Database tables. After any schema change on the Oracle database, you should perform an manual check for missing constraints by executing `ttCacheCheck` on the TimesTen database.

See "Manually initiate check for missing constraints" on page 4-19 for other conditions where you should manually check for missing constraints.

If the check notes missing constraints on the cached tables, TimesTen issues warnings about each missing constraint.

For the following scenarios, the cached table is marked so that transactions that include DML operations are serialized when propagated to the Oracle database.

- Transactions that apply DML operations to AWT cache tables that are missing unique indexes or unique constraints.
- Missing foreign key constraints for tables within a single AWT cache group.

  - If both the referencing table and the referenced table for the foreign key relationship are in the same AWT cache group and the foreign key relationship is not defined, both tables are marked for transaction serialization.
  - If the referencing table is in an AWT cache group and the referenced table is not in an AWT cache group, the table inside the cache group is not marked for transaction serialization. Only a warning is issued to notify the user of the missing constraint.
  - If the referenced table is in an AWT cache group and the referencing table is not in an AWT cache group, the table inside the cache group is not marked for transaction serialization. Only a warning is issued to notify the user of the missing constraint.
- Missing foreign key constraints between cache groups. When you have tables defined in separate AWT cache groups that are missing a foreign key constraint, both tables are marked for serialized transactions.
- If a missing foreign key constraint causes a chain of foreign key constraints to be broken between two AWT cache groups, transactions for all tables within both AWT cache groups are serialized.

Note: An Oracle Database trigger may introduce an operational dependency of which TimesTen may not be aware. In this case, you should either disable parallel propagation for the AWT cache group or do not cache the table in an AWT cache group on which the trigger is created.

Example 4–6 Examples of missing constraints when creating an AWT cache group

The following example creates two tables in the `oratt` schema in the Oracle database. There is a foreign key relationship between `active_customer` and the `ordertab` tables. Because the examples use these tables for parallel propagation, an index is created on the foreign key in the `ordertab` table.

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

SQL> CREATE TABLE ordertab
       (orderid NUMBER(10) NOT NULL PRIMARY KEY,
        custid NUMBER(6) NOT NULL);
Table created.

SQL> ALTER TABLE ordertab
      ADD CONSTRAINT cust_fk
       FOREIGN KEY (custid) REFERENCES active_customer(custid);
Table altered.

SQL> CREATE INDEX order_idx on ordertab (custid);
```

TimesTen automatically checks for missing constraints when each `CREATE CACHE GROUP` is issued. In the following example, a single cache group is created that includes the `active_customer` table. Only a warning is issued since the `active_customer` is the referenced table and the referencing table, `ordertab`, is not in any AWT cache group. The `active_customer` table is not marked for serialized transactions.

```
CREATE WRITETHROUGH CACHE GROUP update_cust
 FROM oratt.active_customer
 (custid NUMBER(6) NOT NULL PRIMARY KEY,
 name VARCHAR2(50),
 addr VARCHAR2(100),
 zip VARCHAR2(12));
Warning  5297: The following Oracle foreign key constraints on AWT cache table
ORATT.ACTIVE_CUSTOMER contain cached columns that do not have corresponding
foreign key constraints on TimesTen: ORATT.CUST_FK [Outside of CG].
```

The following example creates two AWT cache groups on TimesTen, one that includes the `active_customer` table and the other includes the `ordertab` table. There is a missing foreign key constraint between the cache groups. Thus, a warning is issued for both tables, but only the `ordertab` table is marked for serial transactions since it is the referencing table that should contain the foreign key.

```
CREATE WRITETHROUGH CACHE GROUP update_cust
 FROM oratt.active_customer
 (custid NUMBER(6) NOT NULL PRIMARY KEY,
 name VARCHAR2(50),
 addr VARCHAR2(100),
 zip VARCHAR2(12);
Warning  5297: The following Oracle foreign key constraints on AWT cache table
oratt.update_customer contain cached columns that do not have corresponding
foreign key constraints on TimesTen: ordertab.cust_fk [Outside of CG].

CREATE WRITETHROUGH CACHE GROUP update_orders
 FROM oratt.ordertab
 (orderid NUMBER(10) NOT NULL PRIMARY KEY,
  custid NUMBER(6) NOT NULL);
Warning  5295: Propagation will be serialized on AWT cache table
ORATT.ORDERTAB because the following Oracle foreign key constraints on this
table contain cached columns that do not have corresponding foreign key
constraints on TimesTen: ORDERTAB.CUST_FK [Across AWT cache groups].
```

Manually initiate check for missing constraints The `ttCacheCheck` built-in procedure performs the same check for missing constraints for cached tables on the Oracle database as performed automatically by TimesTen. The `ttCacheCheck` provides appropriate messages about missing constraints and the tables marked for serialized propagation. With the `ttCacheCheck` built-in procedure, you can check for missing constraints for a given cache group or for all cache groups in TimesTen to ensure that all cache groups are not missing constraints.

Note: Since `ttCacheCheck` updates system tables to indicate if DML executed against a table should or should not be serialized, you must commit or roll back after the `ttCacheCheck` built-in completes.

For more details of the `ttCacheCheck` built-in procedure, see "ttCacheCheck" in the *Oracle TimesTen In-Memory Database Reference*.

You may need to manually call the `ttCacheCheck` built-in procedure to update the known dependencies after any of the following scenarios:

- After dropping a series of AWT cache groups on TimesTen with the `DROP CACHE GROUP` statement.

- After adding or dropping a unique index, unique constraint, or foreign key on an Oracle Database table that is cached in an AWT cache group. If you do not call the `ttCacheCheck` built-in procedure after adding a constraint, you may receive a run time error on the AWT cache group. After dropping a constraint, TimesTen may serialize transactions even if it is not necessary. Calling the `ttCacheCheck` built-in procedure verifies whether serialization is necessary.
- You can use this built-in procedure to determine why some transactions are being serialized.

Note: The `ttCacheCheck` built-in procedure cannot be called while the replication agent is running.

If a DDL statement is being executed on an AWT cache group when `ttCacheCheck` is called, then `ttCacheCheck` waits for the statement to complete or until the timeout period is reached.

If you have not defined the `CacheAwtParallelism` data store attribute to greater than one or the specified cache group is not an AWT cache group, then the `ttCacheCheck` built-in procedure returns an empty result set.

Example 4–7 Manually executing ttCacheCheck update missing dependencies

The following example shows the user manually executing the `ttCacheCheck` built-in procedure to determine if there are any missing constraints for an AWT cache group `update_orders` that is owned by `cacheuser`. A result set is returned that includes the error message. The `ordertab` table in the `update_orders` cache group is marked for serially propagated transactions.

```
Command> call ttCacheCheck(NULL, 'cacheuser', 'update_orders');

< CACHEUSER, UPDATE_ORDERS, CACHEUSER, ORDERTAB, Foreign Key, CACHEUSER,
CUST_FK, 1, Transactions updating this table will be serialized to Oracle
because: The missing foreign key connects two AWT cache groups.,
table CACHEUSER.ORDERTAB constraint CACHEUSER.CUST_FK foreign key(CUSTID)
references CACHEUSER.ACTIVE_CUSTOMER(CUSTID) >
1 row found.
```

Whenever the cache group schema changes in either the TimesTen or Oracle databases, you can call `ttCacheCheck` against all AWT cache groups to verify all constraints. The following example shows the user manually executing the `ttCacheCheck` built-in procedure to determine if there are any missing constraints for any AWT cache group in the entire TimesTen database by providing a `NULL` value for all input parameters. A result set is returned that includes any error messages.

```
Command> call ttCacheCheck(NULL, NULL, NULL);

< CACHEUSER, UPDATE_ORDERS, CACHEUSER, ORDERTAB, Foreign Key, CACHEUSER,
CUST_FK, 1, Transactions updating this table will be serialized to Oracle
because: The missing foreign key connects two AWT cache groups.,
table CACHEUSER.ORDERTAB constraint CACHEUSER.CUST_FK foreign key(CUSTID)
references CACHEUSER.ACTIVE_CUSTOMER(CUSTID) >
1 row found.
```

Configuring batch size for parallel propagation for AWT cache groups When using AWT cache groups, TimesTen batches together one or more transactions that are to be applied in parallel to the back-end Oracle database. The `CacheParAwtBatchSize` parameter configures a threshold value for the number of rows included in a single batch. Once the maximum number of rows is reached, TimesTen includes the rest of the rows in the transaction (TimesTen does not break up any transactions), but does not add any more transactions to the batch.

For example, a user sets the `CacheParAwtBatchSize` to 200. For the next AWT propagation, there are three transactions, each with 120 rows, that need to be propagated and applied to the Oracle database. TimesTen includes the first two transactions in the first batch for a total of 240 rows. The third transaction is included in a second batch.

The default value for the `CacheParAwtBatchSize` parameter is 125 rows. The minimum value is 1. For more details on the `CacheParAwtBatchSize` parameter in the `ttDBConfig` built-in procedure, see "ttDBConfig" in the *Oracle TimesTen In-Memory Database Reference*.

You can retrieve the current value of `CacheParAwtBatchSize` as follows:

```
call ttDBConfig('CacheParAwtBatchSize');
< CACHEPARAWTBATCHSIZE, 125 >
1 row found.
```

You can set the `CacheParAwtBatchSize` parameter to 200 as follows:

```
call ttDBConfig('CacheParAwtBatchSize','200');
< CACHEPARAWTBATCHSIZE, 200 >
1 row found
```

Set the `CacheParAwtBatchSize` parameter only when advised by Oracle Support, who analyzes the workload and any dependencies in the workload to determine if a different value for `CacheParAwtBatchSize` could improve performance. Dependencies exist when transactions concurrently change the same data. Oracle Support may advise you to reduce this value if there are too many dependencies in the workload.

What an AWT cache group does and does not guarantee

An AWT cache group *can* guarantee that:

- No transactions are lost because of communication failures between the TimesTen and Oracle databases.
- If the replication agent is not running or loses its connection to the Oracle database, automatic propagation of committed updates on the TimesTen cache tables to the cached Oracle Database tables resumes after the agent restarts or reconnects to the Oracle database.
- Transactions are committed in the Oracle database in the same order they were committed in the TimesTen database.

An AWT cache group *cannot* guarantee that:

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

  Execution errors are considered permanent errors and are reported to the *`TimesTenDatabaseFileName`*`.awterrs` file that resides in the same directory as the TimesTen database's checkpoint files. See "Reporting Oracle Database permanent errors for AWT cache groups" on page 4-23 for more information.

- The absolute order of Oracle Database updates is preserved because TimesTen does not resolve update conflicts. The following are some examples:
  - In two separate TimesTen databases (`DB1` and `DB2`), different AWT cache groups cache the same Oracle Database table. An update is committed on the cache table in `DB1`. An update is then committed on the cache table in `DB2`. The two cache tables reside in different TimesTen databases and cache the same Oracle Database table. Because the writethrough operations are asynchronous, the update from `DB2` may get propagated to the Oracle database before the update from `DB1`, resulting in the update from `DB1` overwriting the update from `DB2`.
  - An update is committed on a cache table in an AWT cache group. The same update is committed on the cached Oracle Database table using a passthrough operation. The cache table update, which is automatically and asynchronously propagated to the Oracle database, may overwrite the passed through update that was processed directly on the cached Oracle Database table depending on when the propagated update and the passed through update is processed on the Oracle database. For this and other potential error conditions, TimesTen recommends that you do not execute DML statements directly against Oracle Database tables cached in an AWT cache group. For more information, see "Restrictions with AWT cache groups" on page 4-22.

Restrictions with AWT cache groups

The following restrictions apply when using an AWT cache group:

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

  See "ON DELETE CASCADE cache table attribute" on page 4-43 for more information about the `ON DELETE CASCADE` cache table attribute.

  See "UNIQUE HASH ON cache table attribute" on page 4-44 for more information about the `UNIQUE HASH ON` cache table attribute.
- A `FLUSH CACHE GROUP` statement cannot be issued on the cache group.

  See "Flushing a user managed cache group" on page 5-16 for more information about the `FLUSH CACHE GROUP` statement.
- The cache table definitions cannot contain a `WHERE` clause.

  See "Using a WHERE clause" on page 4-40 for more information about `WHERE` clauses in cache group definitions and operations.
- A `TRUNCATE TABLE` statement cannot be issued on the cache tables.
- AWT cache groups cannot cache Oracle Database views or materialized views.
- The replication agent must be stopped before creating or dropping an AWT cache group.

  See "Managing the replication agent" on page 4-13 for information about how to stop and start the replication agent.
- Committed updates on the TimesTen cache tables are not propagated to the cached Oracle Database tables unless the replication agent is running.
- To create an AWT cache group, the length of the absolute path name of the TimesTen database cannot exceed 248 characters.
- You should avoid executing DML statements on Oracle Database tables cached in an AWT cache group. This could result in an error condition. Any insert, update,

or delete operation on the cached Oracle Database table can negatively affect the operations performed on TimesTen for the affected rows. TimesTen does not detect or resolve update conflicts that occur on the Oracle database. Committed updates made directly on a cached Oracle Database table may be overwritten by a committed update made on the TimesTen cache table when the cache table update is propagated to the Oracle database. In addition, deleting rows on the cached Oracle Database table could cause an empty update if TimesTen tries to update a row that no longer exists.

To ensure that not all data is restricted from DML statements on Oracle Database, you can partition the data on Oracle Database to separate the data that is to be included in the AWT cache group from the data to be excluded from the AWT cache group.

- TimesTen performs deferred checking when determining whether a single SQL statement causes a constraint violation with a unique index.

  For example, suppose there is a unique index on a cached Oracle Database table's `NUMBER` column, and a unique index on the same `NUMBER` column on the TimesTen cache table. There are five rows in the cached Oracle Database table and the same five rows in the cache table. The values in the `NUMBER` column range from 1 to 5.

  An `UPDATE` statement is issued on the cache table to increment the value in the `NUMBER` column by 1 for all rows. The operation succeeds on the cache table but fails when it is propagated to the cached Oracle Database table.

  This occurs because TimesTen performs the unique index constraint check at the end of the statement's execution after all the rows have been updated. The Oracle database, however, performs the constraint check each time after a row has been updated.

  Therefore, when the row in the cache table with value 1 in the `NUMBER` column is changed to 2 and the update is propagated to the Oracle database, it causes a unique constraint violation with the row that has the value 2 in the `NUMBER` column of the cached Oracle Database table.

Reporting Oracle Database permanent errors for AWT cache groups

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

Permanent errors are always reported to the *`TimesTenDatabaseFileName`*`.awterrs` text file that resides in the same directory as the TimesTen database checkpoint files. See "Oracle Database errors reported by TimesTen for AWT" in the *Oracle TimesTen In-Memory Database Troubleshooting Guide* for information about the contents of this file.

You can configure TimesTen to report these errors in both ASCII and XML formats with the `ttCacheConfig` built-in procedure.

Note: Do not pass in any values to the *`tblOwner`* and *`tblName`* parameters for `ttCacheConfig` as they are not applicable to setting the format for the errors file.

- To configure TimesTen to report permanent errors to only the *`TimesTenDatabaseFileName`*`.awterrs` text file, call the `ttCacheConfig` built-in procedure with the `ASCII` parameter. This is the default.

```
Command> call ttCacheConfig('AwtErrorXmlOutput',,,'ASCII');
```

- To configure TimesTen to report permanent errors to both the *`TimesTenDatabaseFileName`*`.awterrs` text file as well as to an XML file named *`TimesTenDatabaseFileName`*`.awterrs.xml`, call the `ttCacheConfig` built-in procedure with the `XML` parameter.

```
Command> call ttCacheConfig('AwtErrorXmlOutput',,,'XML');
```

> **Note:** Before calling `ttCacheConfig` to direct permanent errors to the XML file, you must first stop the replication agent. Then, restart the replication agent after the built-in procedure completes.
>
> For full details on this built-in procedure, see "ttCacheConfig" in the *Oracle TimesTen In-Memory Database Reference*.

When you configure error reporting to be reported in XML format, the following two files are generated when Oracle Database permanent errors occur:

- *`TimesTenDatabaseFileName`*`.awterrs.xml` contains the Oracle Database permanent error messages in XML format.
- *`TimesTenDatabaseFileName`*`.awterrs.dtd` is the file that contains the XML Document Type Definition (DTD), which is used when parsing the *`TimesTenDatabaseFileName`*`.awterrs.xml` file.

  The XML DTD, which is based on the XML 1.0 specification, is a set of markup declarations that describes the elements and structure of a valid XML file containing a log of errors. The XML file is encoded using UTF-8. The following are the elements for the XML format.

> **Note:** For more information on reading and understanding XML Document Type Definitions, see `http://www.w3.org/TR/REC-xml/`.

```
<!ELEMENT ttawterrorreport (awterrentry*) >
<!ELEMENT awterrentry(header, (failedop)?, failedtxn) >
<!ELEMENT header (time, datastore, oracleid, transmittingagent, errorstr,
 (ctn)?, (batchid)?, (depbatchid)?) >
<!ELEMENT failedop (sql) >
<!ELEMENT failedtxn ((sql)+) >
<!ELEMENT time (hour, min, sec, year, month, day) >
<!ELEMENT hour (#PCDATA) >
<!ELEMENT min (#PCDATA) >
<!ELEMENT sec (#PCDATA) >
<!ELEMENT year (#PCDATA) >
<!ELEMENT month (#PCDATA) >
<!ELEMENT day (#PCDATA) >
<!ELEMENT datastore (#PCDATA) >
<!ELEMENT oracleid (#PCDATA) >
<!ELEMENT transmittingagent (transmitingname, pid, threadid) >
<!ELEMENT pid (#PCDATA) >
<!ELEMENT threadid (#PCDATA) >
<!ELEMENT transmittingname (#PCDATA) >
<!ELEMENT errorstr (#PCDATA) >
<!ELEMENT ctn (timestamp, seqnum) >
<!ELEMENT timestamp(#PCDATA) >
<!ELEMENT seqnum(#PCDATA) >
```

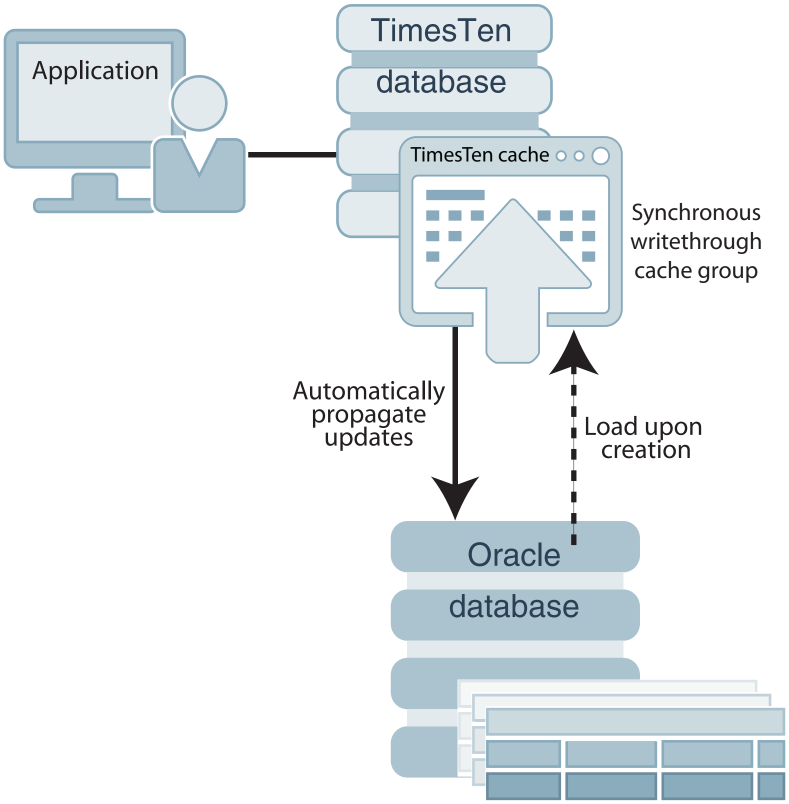
```
<!ELEMENT batchid(#PCDATA) >
<!ELEMENT depbatchid(#PCDATA) >
<!ELEMENT sql(#PCDATA) >
```

Synchronous WriteThrough (SWT) cache group

A synchronous writethrough (SWT) cache group enforces a caching behavior where committed updates on the TimesTen cache tables are automatically and synchronously propagated to the cached Oracle Database tables as shown in Figure 4–7.

Note: You should avoid executing DML statements on Oracle Database tables cached in an SWT cache group. This can result in an error condition. For more information, see "Restrictions with SWT cache groups" on page 4-27.

Figure 4–7 Synchronous writethrough cache group



The transaction commit on the TimesTen database occurs synchronously with the commit on the Oracle database. When an application commits a transaction in the TimesTen database, the transaction is processed in the Oracle database before it is processed in TimesTen. The application is blocked until the transaction has completed in both the Oracle and TimesTen databases.

If the transaction fails to commit in the Oracle database, the application must roll back the transaction in TimesTen. If the Oracle Database transaction commits successfully but the TimesTen transaction fails to commit, the cache tables in the SWT cache group are no longer synchronized with the cached Oracle Database tables.

Note: The behavior and error conditions for how commit occurs on both the TimesTen and Oracle databases when committing propagated updates is the same commit process on a user-managed cache group with the `PROPAGATE` cache attribute that is described in "PROPAGATE cache table attribute" on page 4-29.

To manually resynchronize the cache tables with the cached Oracle Database tables, call the `ttCachePropagateFlagSet` built-in procedure to disable update propagation, and then reissue the transaction in the TimesTen database after correcting the problem that caused the transaction commit to fail in TimesTen. Then, call the `ttCachePropagateFlagSet` built-in procedure to re-enable update propagation. You can also resynchronize the cache tables with the cached Oracle Database tables by reloading the accompanying cache groups.

The following is the definition of the Oracle Database table that is to be cached in the SWT cache group that is defined in Example 4–8. The Oracle Database table is owned by the schema user `oratt`. The `oratt` user must be granted the `CREATE SESSION` and `RESOURCE` privileges before it can create tables.

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

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

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

Example 4–8 Creating a SWT cache group

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

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

When TimesTen manages operations for SWT cache groups, it connects to the Oracle database using the current user's credentials as the user name and the `OraclePwd` connection attribute as the Oracle password. TimesTen does not connect to the Oracle database with the cache administration user name and password set with the `ttCacheUidPwdSet` built-in procedure when managing SWT cache group operations. For more details, see "Set the cache administration user name and password" on page 3-9.

Restrictions with SWT cache groups

The following restrictions apply when using an SWT cache group:

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

  See "ON DELETE CASCADE cache table attribute" on page 4-43 for more information about the `ON DELETE CASCADE` cache table attribute.

  See "UNIQUE HASH ON cache table attribute" on page 4-44 for more information about the `UNIQUE HASH ON` cache table attribute.

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

  See "Flushing a user managed cache group" on page 5-16 for more information about the `FLUSH CACHE GROUP` statement

- The cache table definitions cannot contain a `WHERE` clause.

  See "Using a WHERE clause" on page 4-40 for more information about `WHERE` clauses in cache group definitions and operations.

- A `TRUNCATE TABLE` statement cannot be issued on the cache tables.
- SWT cache groups cannot cache Oracle Database views or materialized views.
- You should avoid executing DML statements directly on Oracle Database tables cached in an SWT cache group. This could result in an error condition. Any insert, update, or delete operation on the cached Oracle Database table can negatively affect the operations performed on TimesTen for the affected rows. TimesTen does not detect or resolve update conflicts that occur on the Oracle database. Committed updates made directly on a cached Oracle Database table may be overwritten by a committed update made on the TimesTen cache table when the cache table update is propagated to the Oracle database. In addition, deleting rows on the cached Oracle Database table could cause an empty update if TimesTen tries to update a row that no longer exists.

  To ensure that not all data is restricted from DML statements on Oracle Database, you can partition the data on Oracle Database to separate the data that is to be included in the SWT cache group from the data to be excluded from the SWT cache group.

User-managed cache group

If the system managed cache groups (read-only, AWT, SWT) do not satisfy your application's requirements, you can create a user-managed cache group that defines customized caching behavior with one or more of the following cache table attributes:

Note: When TimesTen manages operations for user-managed cache groups, it connects to the Oracle database using the current user's credentials as the user name and the `OraclePwd` connection attribute as the Oracle password. TimesTen does not connect to the Oracle database with the cache administration user name and password set with the `ttCacheUidPwdSet` built-in procedure for user-managed cache group operations. For more details, see "Set the cache administration user name and password" on page 3-9.

- You can specify the READONLY cache table attribute on individual cache tables in a user-managed cache group to define read-only behavior where the data is refreshed on TimesTen from the Oracle database at the table level.
- You can specify the `PROPAGATE` cache table attribute on individual cache tables in a user-managed cache group to define synchronous writethrough behavior at the table level. The PROPAGATE cache table attribute specifies that committed updates on the cache table are automatically and synchronously propagated to the cached Oracle Database table.
- You can define a user-managed cache group to automatically refresh and propagate committed updates between the Oracle and TimesTen databases by using the `AUTOREFRESH` cache group attribute and the `PROPAGATE` cache table attribute. Using both attributes enables bidirectional transmit, so that committed updates on the TimesTen cache tables or the cached Oracle Database tables are propagated or refreshed to each other.

  See "AUTOREFRESH cache group attribute" on page 4-34 for more information about defining an autorefresh mode, interval, and state.
- You can use the `LOAD CACHE GROUP`, `REFRESH CACHE GROUP`, and `FLUSH CACHE GROUP` statements to manually control the transmit of committed updates between the Oracle and TimesTen databases.

  See "Loading and refreshing a cache group" on page 5-2 for more information about the `LOAD CACHE GROUP` and `REFRESH CACHE GROUP` statements. See "Flushing a user managed cache group" on page 5-16 for more information about the `FLUSH CACHE GROUP` statement.
- You can cache Oracle Database materialized views in a user-managed cache group that does not use either the `PROPAGATE` or `AUTOREFRESH` cache group attributes. The cache group must be manually loaded and flushed. You cannot cache Oracle Database views.

The following sections provide more information about user-managed cache groups:

- READONLY cache table attribute
- PROPAGATE cache table attribute
- Examples of user-managed cache groups

READONLY cache table attribute

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

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

Example 4–10 demonstrates the `READONLY` cache table attribute in the `oratt.cust_interests` cache table.

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

- If the cache group uses the `AUTOREFRESH` cache group attribute, the `READONLY` cache table attribute must be specified on all or none of its cache tables.

  See "AUTOREFRESH cache group attribute" on page 4-34 for more information about using the `AUTOREFRESH` cache group attribute.

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

  See "PROPAGATE cache table attribute" on page 4-29 for more information about using the `PROPAGATE` cache table attribute.

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

  See "Flushing a user managed cache group" on page 5-16 for more information about the `FLUSH CACHE GROUP` statement.

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

PROPAGATE cache table attribute

The `PROPAGATE` cache table attribute can be specified only for cache tables in a user-managed cache group. `PROPAGATE` specifies that committed updates on the TimesTen cache table as part of a TimesTen transaction are automatically and synchronously propagated to the cached Oracle Database table. If the `PROPAGATE` cache table attribute is not specified, then the default setting for a cache table in a user-managed cache group is the `NOT PROPAGATE` cache table attribute (which does not propagate committed updates on the cache table to the cached Oracle table).

All SQL statements executed by an application on cached tables are applied to the cached tables immediately. All of these operations are buffered until the transaction commits or reaches a memory upper limit. At this time, all operations are propagated to the tables in the Oracle database.

Note: If the TimesTen database or its daemon fails unexpectedly, the results of the transaction on either the TimesTen or Oracle databases are not guaranteed.

Since the operations in the transaction are applied to tables in both the TimesTen and Oracle databases, the process for committing is as follows:

1. After the operations are propagated to the Oracle database, the commit is first attempted in the Oracle database.
   - If an error occurs when applying the operations on the tables in the Oracle database, then all operations are rolled back on the tables on the Oracle database. If the commit fails in the Oracle database, the commit is not attempted in the TimesTen database and the application must roll back the TimesTen transaction. If the user tries to execute another statement, an error displays informing them of the need for a roll back. As a result, the Oracle database never misses updates committed in TimesTen.
2. If the commit succeeds in the Oracle database, the commit is attempted in the TimesTen database.
   - If the transaction successfully commits on the Oracle database, the user's transaction is committed on TimesTen (indicated by the commit log record in the transaction log) and notifies the application. If the application ends abruptly before TimesTen informs it of the success of the local commit, TimesTen is still able to finalize the transaction commit on TimesTen based on what is saved in the transaction log.

- If the transaction successfully commits on the Oracle database and a failure occurs before returning the status of the commit on TimesTen, then no record of the successful commit is written into the transaction log and the transaction is rolled back.
- If the commit fails in TimesTen, an error message is returned from TimesTen indicating the cause of the failure. You then need to manually resynchronize the cache tables with the Oracle Database tables.

Note: See "Synchronous WriteThrough (SWT) cache group" on page 4-25 for information on how to resynchronize the cache tables with the Oracle Database tables.

You can disable propagation of committed updates on the TimesTen cached tables to the Oracle database with the `ttCachePropagateFlagSet` built-in procedure. This built-in procedure can enable or disable automatic propagation so that committed updates on a cache table on TimesTen for the current transaction are never propagated to the cached Oracle Database table. You can then re-enable propagation for DML statements by resetting the flag to one with the `ttCachePropagateFlagSet` built-in procedure. After the flag is set back to one, propagation of committed updates to the Oracle database resumes. The propagation flag automatically resets to one after the transaction is committed or rolled back. See "ttCachePropagateFlagSet" in the *Oracle TimesTen In-Memory Database Reference* for more details.

Example 4–9 demonstrates the use of the `PROPAGATE` cache table attribute in the `oratt.active_customer` cache table.

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

- If the cache group uses the `AUTOREFRESH` cache group attribute, the `PROPAGATE` cache table attribute must be specified on all or none of its cache tables.

  See "AUTOREFRESH cache group attribute" on page 4-34 for more information about using the `AUTOREFRESH` cache group attribute.
- If the cache group uses the `AUTOREFRESH` cache group attribute, the `NOT PROPAGATE` cache table attribute cannot be explicitly specified on any of its cache tables.
- You cannot use both the `PROPAGATE` and `READONLY` cache table attributes on the same cache table.

  See "READONLY cache table attribute" on page 4-28 for more information about using the `READONLY` cache table attribute.
- A `FLUSH CACHE GROUP` statement cannot be issued on the cache group unless one or more of its cache tables use neither the `PROPAGATE` nor the `READONLY` cache table attribute.

  See "Flushing a user managed cache group" on page 5-16 for more information about the `FLUSH CACHE GROUP` statement.
- After the `PROPAGATE` cache table attribute has been specified on a cache table, you cannot change this attribute unless you drop the cache group and re-create it.
- The `PROPAGATE` cache table attribute cannot be used when caching Oracle Database materialized views.
- TimesTen does not perform a conflict check to prevent a propagate operation from overwriting data that was updated directly on a cached Oracle Database table.

Therefore, updates should only be performed directly on the TimesTen cache tables or the cached Oracle Database tables, but not both.

Examples of user-managed cache groups

The following are the definitions of the Oracle Database tables that are to be cached in the user-managed cache groups that are defined in Example 4–9 and Example 4–10. The Oracle Database tables are owned by the schema user `oratt`. The `oratt` user must be granted the `CREATE SESSION` and `RESOURCE` privileges before it can create tables.

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

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

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

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

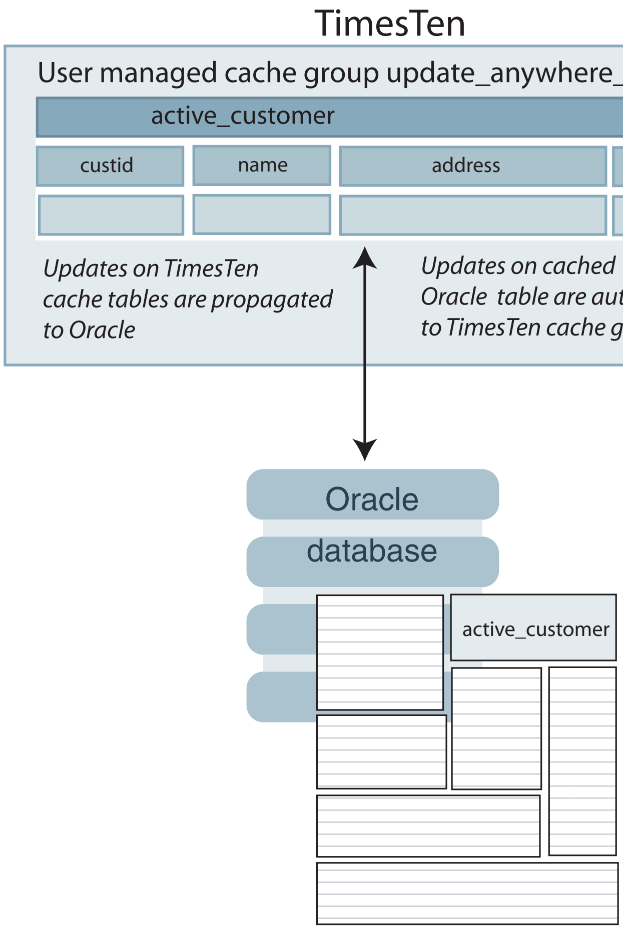
Use the `CREATE USERMANAGED CACHE GROUP` statement to create a user-managed cache group.

Example 4–9   Creating a single-table user-managed cache group

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

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

Figure 4–8 Single-table user-managed cache group


TimesTen
User managed cache group update_anywhere_customers
active_customer
custid
name
address
zip
Updates on TimesTen cache tables are propagated to Oracle
Updates on cached Oracle table are autorefreshed to TimesTen cache group
Oracle database
active_customer

In this example, all columns except `region` from the `oratt.active_customer` table are cached in TimesTen. Since this is defined with the `PROPAGATE` cache table attribute, updates committed on the `oratt.active_customer` cache table on TimesTen are transmitted to the `oratt.active_customer` cached Oracle Database table. Since the user-managed cache table is also defined with the `AUTOREFRESH` cache attribute, any committed updates on the `oratt.active_customer` Oracle Database table are transmitted to the `update_anywhere_customers` cached table.

The companion Oracle Database user of the TimesTen cache manager user must be granted the `SELECT` privilege on the `oratt.active_customer` table in order for the cache manager user to create a user-managed cache group that caches this table, and for autorefresh operations to occur from the cached Oracle Database table to the TimesTen cache table. The companion Oracle Database user must also be granted the `INSERT`, `UPDATE` and `DELETE` privileges on the `oratt.active_customer` table for synchronous writethrough operations to occur from the TimesTen cache table to the cached Oracle Database table.

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

If you manually created the Oracle Database objects used to enforce the predefined behaviors of a user-managed cache group that uses the `AUTOREFRESH MODE INCREMENTAL` cache group attribute as described in "Manually create Oracle Database objects used to manage data caching" on page 3-5, you need to set the autorefresh state to `OFF` when creating the cache group.

Then you need to run the `ttIsql` utility's `cachesqlget` command to generate a SQL*Plus script used to create a log table and a trigger in the Oracle database for each Oracle Database table that is cached in the user-managed cache group.

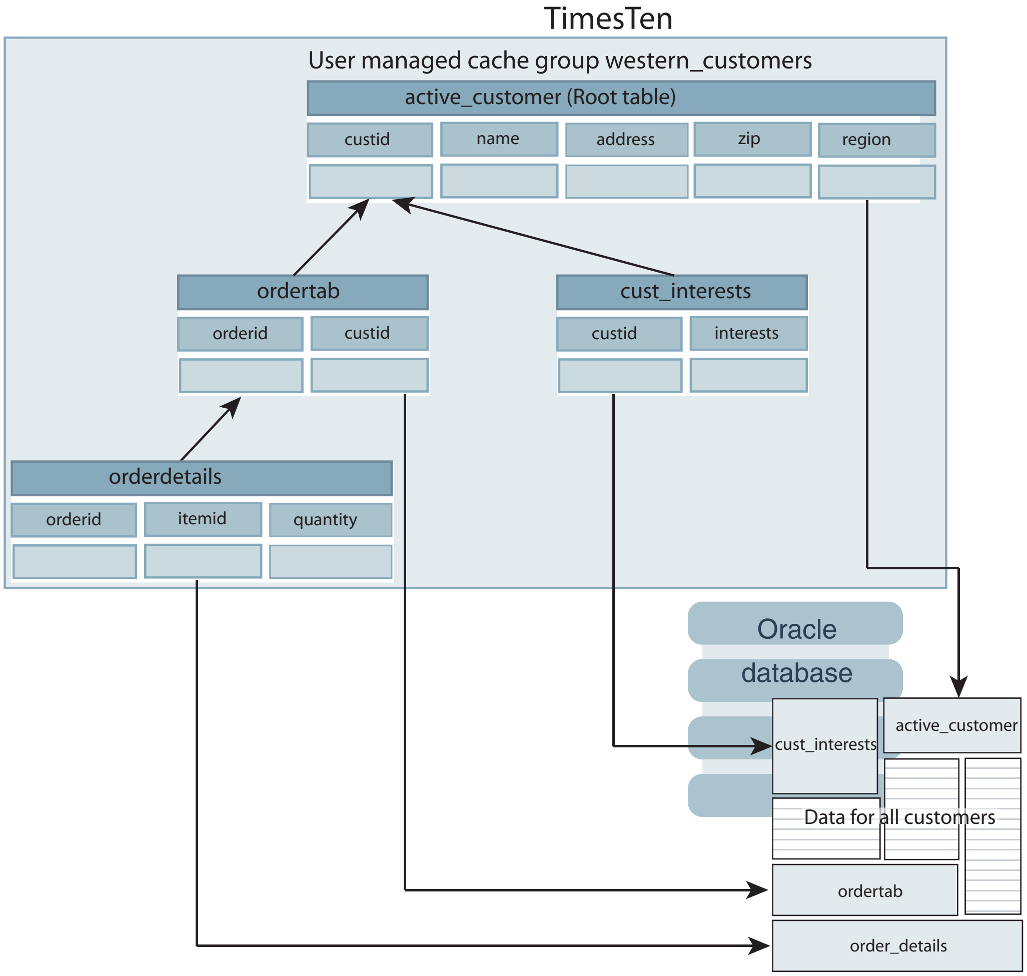
See "Manually creating Oracle Database objects for autorefresh cache groups" on page 4-37 for more information.

Example 4–10 Creating a multiple-table user-managed cache group

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

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

Figure 4–9 Multiple-table user-managed cache group



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

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

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

AUTOREFRESH cache group attribute

The following describes how to use the `AUTOREFRESH` cache group attribute

- AUTOREFRESH cache group attribute overview
- Altering a cache group to change the AUTOREFRESH mode, interval or state
- Manually creating Oracle Database objects for autorefresh cache groups
- Disabling full autorefresh for cache groups

AUTOREFRESH cache group attribute overview

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

The following are the default settings of the autorefresh attributes:

- The autorefresh mode is incremental.
- The autorefresh interval is 5 minutes.
- The autorefresh state is `PAUSED`.

TimesTen supports two autorefresh modes:

- `INCREMENTAL`: Committed updates on cached Oracle Database tables are automatically refreshed to the TimesTen cache tables based on the cache group's autorefresh interval. Incremental autorefresh mode uses Oracle Database objects to track committed updates on cached Oracle Database tables. See "Managing a caching environment with Oracle Database objects" on page 6-7 for information on these objects.
- `FULL`: All cache tables are automatically refreshed, based on the cache group's autorefresh interval, by unloading all their rows and then reloading from the cached Oracle Database tables.

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

When using incremental autorefresh mode, committed updates on cached Oracle Database tables are tracked in change log tables in the Oracle database. Under certain circumstances, it is possible for some change log records to be deleted from the change log table before they are automatically refreshed to the TimesTen cache tables. If this occurs, TimesTen initiates a full automatic refresh on the cache group.

- See "Disabling full autorefresh for cache groups" on page 4-38 for information on how to disable any full autorefresh request when configured to use incremental autorefresh.
- See "Monitoring the cache administration user's tablespace" on page 6-14 for information on how to configure an action to take when the tablespace that the change log tables reside in becomes full.
- If you have a dynamic read-only cache group with incremental autorefresh, you can reduce contention and improve performance by enabling the `DynamicLoadReduceContention` database system parameter. See "Reducing contention on TimesTen for dynamic read-only cache groups with incremental autorefresh" on page 7-10 for more details.

The change log table on the Oracle database does not have column-level resolution because of performance reasons. Thus the autorefresh operation updates all of the columns in a row. XLA reports that all of the columns in the row have changed even if the data did not actually change in each column.

The autorefresh interval determines how often autorefresh operations occur in minutes, seconds or milliseconds. Cache groups with the same autorefresh interval are refreshed within the same transaction. You can specify continuous autorefresh with an autorefresh interval of 0 milliseconds. With continuous autorefresh, the next autorefresh cycle is scheduled as soon as possible after the last autorefresh cycle has ended.

You can manually initiate an immediate autorefresh operation with the `ttCacheAutorefresh` built-in procedure. For more information, see "ttCacheAutorefresh" in *Oracle TimesTen In-Memory Database Reference*.

The autorefresh state can be set to `ON`, `PAUSED` or `OFF`. Autorefresh operations are scheduled by TimesTen when the cache group's autorefresh state is `ON`.

When the cache group's autorefresh state is `OFF`, committed updates on the cached Oracle Database tables are not tracked.

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

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

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

  See "Flushing a user managed cache group" on page 5-16 for more information about the `FLUSH CACHE GROUP` statement.

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

  See "Altering a cache group to change the AUTOREFRESH mode, interval or state" on page 4-37 for more information about the `ALTER CACHE GROUP` statement.

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

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

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

  See "Dynamic cache groups" on page 4-51 for more information about dynamic cache groups.

- The autorefresh state must be `PAUSED` before you can issue a `LOAD CACHE GROUP` statement on the cache group, unless the cache group is dynamic, in which case the autorefresh state must be `PAUSED` or `ON`. The `LOAD CACHE GROUP` statement cannot contain a `WHERE` clause, unless the cache group is dynamic, in which case the `WHERE` clause must be followed by a `COMMIT EVERY` *n* `ROWS` clause.

  See "Using a WHERE clause" on page 4-40 for more information about `WHERE` clauses in cache group definitions and operations.

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

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

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

- To use the `AUTOREFRESH` cache group attribute in a user-managed cache group, all of the cache tables must be specified with the `PROPAGATE` cache table attribute or all of the cache tables must be specified the `READONLY` cache table attribute.
- You cannot specify the `AUTOREFRESH` cache group attribute in a user-managed cache group that contains cache tables that explicitly use the `NOT PROPAGATE` cache table attribute.
- The `AUTOREFRESH` cache table attribute cannot be used when caching Oracle Database materialized views in a user-managed cache group.
- LRU aging cannot be specified on the cache group, unless the cache group is dynamic where LRU aging is defined by default.

  See "LRU aging" on page 4-46 for more information about LRU aging.

If you create a unique index on a cache group with the `AUTOREFRESH` cache group attribute, the index is changed to a non-unique index to avoid a constraint violation. A constraint violation could occur with a unique index because conflicting updates could occur in the same statement execution on the Oracle Database table, while each row update is executed separately in TimesTen. If the unique index exists on the Oracle Database table that is being cached, then uniqueness is enforced on the Oracle Database table and does not need to be verified again in TimesTen.

In Example 4–9, the `update_anywhere_customers` cache group uses the `AUTOREFRESH` cache group attribute.

Altering a cache group to change the AUTOREFRESH mode, interval or state

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

If you change a cache group's autorefresh state to `OFF` or drop a cache group that has an autorefresh operation in progress:

- The autorefresh operation stops if the setting of the `LockWait` connection attribute is greater than 0. The `ALTER CACHE GROUP` or `DROP CACHE GROUP` statement preempts the autorefresh operation.
- The autorefresh operation continues if the `LockWait` connection attribute is set to 0. The `ALTER CACHE GROUP` or `DROP CACHE GROUP` statement is blocked until the autorefresh operation completes or the statement fails with a lock timeout error.

Example 4–11 Altering the autorefresh attributes of a cache group

The following statements change the autorefresh mode, interval and state of the `customer_orders` cache group:

```
ALTER CACHE GROUP customer_orders SET AUTOREFRESH MODE FULL;
ALTER CACHE GROUP customer_orders SET AUTOREFRESH INTERVAL 30 SECONDS;
ALTER CACHE GROUP customer_orders SET AUTOREFRESH STATE ON;
```

Manually creating Oracle Database objects for autorefresh cache groups

If you manually created the Oracle Database objects used to enforce the predefined behaviors of an autorefresh cache group as described in "Manually create Oracle

Database objects used to manage data caching" on page 3-5, you need to set the autorefresh state to `OFF` when creating the cache group.

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

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

Example 4–12 Creating a read-only cache group when Oracle Database objects were manually created

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

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

% ttIsql "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> cachesqlget INCREMENTAL_AUTOREFRESH customer_orders INSTALL /tmp/obj.sql;
Command> exit

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

ALTER CACHE GROUP customer_orders SET AUTOREFRESH STATE PAUSED;
```

Disabling full autorefresh for cache groups

When using incremental autorefresh mode on your cache group, committed updates on cached Oracle Database tables are tracked in change log tables in the Oracle database. Under certain error scenarios, it is possible for some change log records to be deleted (truncated) from the change log table before they are automatically refreshed to the TimesTen cache tables. If this occurs, TimesTen initiates a full autorefresh on the cache group.

Some applications choose incremental autorefresh instead of full autorefresh mode for performance reasons. A full autorefresh can affect performance because:

- More rows are refreshed with a full autorefresh.
- A full autorefresh executes within a single transaction with no parallelism.

If performance is a concern, you can set the `DisableFullAutorefresh` cache configuration parameter to 1 to disallow full autorefresh requests for all cache groups defined with incremental autorefresh. In this case, the initial load for each cache group requires a manual load.

Note: The default value is `0` for the `DisableFullAutorefresh` cache configuration parameter, which specifies the normal full autorefresh behavior.

```
call ttCacheConfig('DisableFullAutorefresh',,,'1');
```

You can query the current value of the `DisableFullAutorefresh` parameter.

```
call ttCacheConfig('DisableFullAutorefresh');
```

If a full autorefresh is triggered for a cache group, TimesTen changes the cache group status to `disabled`. After which, all autorefresh operations cease on the cache group. You are notified of this action with a daemon log message.

The TimesTen database status is set to `recovering` when at least one of its autorefresh cache groups have an autorefresh status of `disabled` or `recovering`. You can check the state of a database and cache group with the `ttCacheDbCgStatus` built-in procedure. The following example shows that:

- `Recovering`: Some or all the cache groups with the `AUTOREFRESH` attribute in the database are being resynchronized with the Oracle database server. The status of at least one cache group is `recovering`.
- `Disabled`: The `cg1` cache group is `disabled`.

```
Command> call ttCacheDbCgStatus('ttuser','cg1');
< recovering, disabled >
1 row found.
```

When you set the `DisableFullAutorefresh` cache configuration parameter to 1, then the `DeadDbRecovery` cache configuration parameter automatically changes to `Manual`. TimesTen restores the original setting for the `DeadDbRecovery` cache configuration parameter if you change the `DisableFullAutorefresh` cache configuration parameter to 0.

If the autorefresh status of a cache group is either disabled or dead, its cache tables are no longer being automatically refreshed when updates are committed on the cached Oracle Database tables. The cache group must be recovered in order to resynchronize the cache tables with the cached Oracle Database tables.

- For each cache group whose autorefresh status is disabled, a `REFRESH CACHE GROUP` statement must be issued in order to resume autorefresh operations for these cache groups.
- For each dynamic cache group whose autorefresh status is disabled, an `UNLOAD CACHE GROUP` statement must be issued in order to resume autorefresh operations for these cache groups.

Example 4–13 Manual refresh of a disabled cache group

Pause autorefresh for the cache group and return the cache group status to `OK` with the `ALTER CACHE GROUP SET AUTOREFRESH STATE PAUSED` statement. Then, manually request a full refresh with the `REFRESH CACHE GROUP` statement (optionally, with parallelism).

```
ALTER CACHE GROUP cg_static SET AUTOREFRESH STATE PAUSED;
REFRESH CACHE GROUP cg_static COMMIT EVERY 500 ROWS PARALLEL 3;
```

Example 4–14 Reload dynamic cache group

1. To return the cache group status to `OK`, pause autorefresh for the cache group with the `ALTER CACHE GROUP SET AUTOREFRESH STATE PAUSED` statement.
2. Unload the disabled dynamic cache group with the `UNLOAD CACHE GROUP` statement.
3. Optionally, you can load the cache group with the `LOAD CACHE GROUP` statement (optionally, with parallelism) or initiate a dynamic load. See "Dynamically loading a cache instance" on page 5-10 for details on dynamic load requests.

```
ALTER CACHE GROUP cg_dynamic SET AUTOREFRESH STATE PAUSED;
UNLOAD CACHE GROUP cg_dynamic COMMIT EVERY 500 ROWS;
LOAD CACHE GROUP cg_dynamic COMMIT EVERY 500 ROWS PARALLEL 3;
```

Using a WHERE clause

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

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

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

- `WHERE` clauses can only be specified in the cache table definitions of a `CREATE CACHE GROUP` statement for read-only and user-managed cache groups.
- A `WHERE` clause can be specified in a `LOAD CACHE GROUP` statement except on an explicitly loaded autorefresh cache group.

  See "Loading and refreshing a cache group" on page 5-2 for more information about the `LOAD CACHE GROUP` statement.
- A `WHERE` clause can be specified in a `REFRESH CACHE GROUP` statement except on an autorefresh cache group.

  See "Loading and refreshing a cache group" on page 5-2 for more information about the `REFRESH CACHE GROUP` statement.
- A `WHERE` clause can be specified in a `FLUSH CACHE GROUP` statement on a user-managed cache group that allows committed updates on the TimesTen cache tables to be flushed to the cached Oracle Database tables.

  See "Flushing a user managed cache group" on page 5-16 for more information about the `FLUSH CACHE GROUP` statement.

- `WHERE` clauses in a `CREATE CACHE GROUP` statement cannot contain a subquery. Therefore, each `WHERE` clause cannot reference any table other than the one in its cache table definition. However, a `WHERE` clause in a `LOAD CACHE GROUP`, `UNLOAD CACHE GROUP`, `REFRESH CACHE GROUP` or `FLUSH CACHE GROUP` statement may contain a subquery.
- A `WHERE` clause in a `LOAD CACHE GROUP`, `REFRESH CACHE GROUP` or `FLUSH CACHE GROUP` statement can reference only the root table of the cache group, unless the `WHERE` clause contains a subquery.
- `WHERE` clauses in the cache table definitions are only enforced when the cache group is manually loaded or refreshed, or the cache tables are dynamically loaded. If a cache table is updatable, you can insert or update a row such that the `WHERE` clause in the cache table definition for that row is not satisfied.
- All tables and columns referenced in `WHERE` clauses when creating, loading, refreshing, unloading or flushing the cache group must be fully qualified. For example:

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

In Example 4–10, both the `oratt.active_customer` and `oratt.orderdetails` tables contain a `WHERE` clause.

Proper placement of WHERE clause in a CREATE CACHE GROUP statement

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

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

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

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

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

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

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

Referencing Oracle Database PL/SQL functions in a WHERE clause

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

For example, in the Oracle database:

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

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

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

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

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

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

ON DELETE CASCADE cache table attribute

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

Example 4–15 Using the ON DELETE CASCADE cache table attribute

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

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

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

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

- For AWT and SWT cache groups, and for TimesTen cache tables in user-managed cache groups that use the `PROPAGATE` cache table attribute, foreign keys in cache tables that use the `ON DELETE CASCADE` cache table attribute must be a proper subset of the foreign keys in the cached Oracle Database tables that use the `ON DELETE CASCADE` attribute. `ON DELETE CASCADE` actions on the cached Oracle

Database tables are applied to the TimesTen cache tables as individual deletes. `ON DELETE CASCADE` actions on the cache tables are applied to the cached Oracle Database tables as a cascaded operation.

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

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

UNIQUE HASH ON cache table attribute

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

Example 4–16 Using the UNIQUE HASH ON cache table attribute

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

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

See the `CREATE CACHE GROUP` statement in *Oracle TimesTen In-Memory Database SQL Reference* for more information about the `UNIQUE HASH ON` cache table attribute.

Caching Oracle Database synonyms

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

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

Caching Oracle Database LOB data

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

- Oracle Database `CLOB` data is cached as TimesTen `VARCHAR2` data.
- Oracle Database `BLOB` data is cached as TimesTen `VARBINARY` data.

- Oracle Database `NCLOB` data is cached as TimesTen `NVARCHAR2` data.

Example 4–17 Caching Oracle Database LOB data

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

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

Insert values into the Oracle Database table. The values are implicitly converted to TimesTen `VARCHAR2`, `VARBINARY`, OR `NVARCHAR2` data types.

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

1 row inserted.
```

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

```
CREATE DYNAMIC ASYNCHRONOUS WRITETHROUGH CACHE GROUP cg1
  FROM t
 (i INT NOT NULL PRIMARY KEY
  , c VARCHAR2(4194303)
  , b VARBINARY(4194303)
  , nc NVARCHAR2(2097152));

CALL ttrepstart;
```

Load the data dynamically into the TimesTen cache group.

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

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

1 row found.
```

Restrictions on caching Oracle Database LOB data

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

- Column size is enforced when a cache group is created. `VARBINARY`, `VARCHAR2` and `NVARCHAR2` data types have a size limit of 4 megabytes. Values that exceed the user-defined column size are truncated at run time without notification.
- Empty values in fields with `CLOB` and `BLOB` data types are initialized but not populated with data. Empty `CLOB` and `BLOB` fields are treated as follows:
  - Empty `LOB` fields in the Oracle database are returned as `NULL` values.
  - Empty `VARCHAR2` and `VARBINARY` fields in the TimesTen cache are propagated as `NULL` values.

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

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

Implementing aging in a cache group

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

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

An aging policy is specified in the cache table definition of the root table in a `CREATE CACHE GROUP` statement and applies to all cache tables in the cache group because aging is performed at the cache instance level. When rows are deleted from the cache tables by aging out, the rows in the cached Oracle Database table are not deleted.

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

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

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

LRU aging

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

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

Example 4–18 Defining an LRU aging policy on a cache group

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

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

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

The following are the LRU aging attributes:

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

Example 4–19 Setting the LRU aging attributes

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

```
Command> CALL ttAgingLRUConfig(.75, .95, 5);
```

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

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

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

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

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

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

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

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

Time-based aging

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

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

The definitions of the Oracle Database tables that are to be cached in the AWT cache group defined in Example 4–21 are defined in Example 4–20. The Oracle Database tables are owned by the schema user `oratt`. The `oratt` user must be granted the `CREATE SESSION` and `RESOURCE` privileges before it can create tables.

Example 4–20 Oracle Database table definitions

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

CREATE TABLE order_item
(orditem_id NUMBER(12) NOT NULL PRIMARY KEY,
 ord_num    NUMBER(10),
 prod_num   VARCHAR2(6),
 quantity   NUMBER(3));
```

The companion Oracle Database user of the TimesTen cache manager user must be granted the `SELECT` privilege on the `oratt.orders` and `oratt.order_item` tables in order for the cache manager user to create an AWT cache group that caches these tables. The cache administration user must be granted the `INSERT`, `UPDATE` and `DELETE` Oracle Database privileges for the `oratt.orders` and `oratt.order_item` tables for asynchronous writethrough operations to be applied on the Oracle Database.

Example 4–21 Defining a time-based aging policy on a cache group

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

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

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

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

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

You can create an index on the timestamp column to optimize performance of the aging process.

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

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

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

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

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

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

- Drop the time-based aging policy on a cache group by specifying the root table and using the `DROP AGING` clause.

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

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

Manually scheduling an aging process

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

Example 4–22 Starting a one-time aging process

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

```
Command> CALL ttAgingScheduleNow('oratt.orders');
```

Rows in the `oratt.orders` root table that are candidates for aging are deleted as well as related rows in the `oratt.order_item` child table.

When you call the `ttAgingScheduleNow` built-in procedure, the aging process starts regardless of whether the table's aging state is `ON` or `OFF`. If you want to start an aging process on a particular cache group, specify the name of the cache group's root table when you call the built-in procedure. If the `ttAgingScheduleNow` built-in procedure is called with no parameters, it starts an aging process and then resets the start of the next aging cycle on all tables in the TimesTen database that have an aging policy defined.

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

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

Configuring a sliding window

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

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

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

Example 4–23 Defining a cache group with sliding window properties

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

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

New data in the `oratt.orders` cached Oracle Database table are automatically refreshed into the `oratt.orders` TimesTen cache table every 1440 minutes. Cache instances that are greater than 30 days old based on the difference between the current system timestamp and the timestamp in the `when_shipped` column are candidates for aging. The aging process checks every 24 hours to see if there are cache instances that can be aged out of the cache tables. Therefore, this cache group stores orders that have been shipped within the last 30 days.

The autorefresh interval and the lifetime used for aging determine the duration that particular rows remain in the cache tables. It is possible for data to be aged out of the cache tables before it has been in the cache tables for its lifetime. For example, for a read-only cache group if the autorefresh interval is 3 days and the lifetime is 30 days, data that is already 3 days old when it is refreshed into the cache tables is deleted after 27 days because aging is based on the timestamp stored in the rows of the cached Oracle Database tables that gets loaded into the TimesTen cache tables, not when the data is refreshed into the cache tables.

Dynamic cache groups

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

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

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

Example 4–24 Dynamic read-only cache group

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

```
CREATE DYNAMIC READONLY CACHE GROUP online_customers
```

```
FROM oratt.customer
 (cust_num NUMBER(6) NOT NULL,
  region   VARCHAR2(10),
  name     VARCHAR2(50),
  address  VARCHAR2(100),
  PRIMARY KEY(cust_num));
```

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

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

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

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

Note: If you have a dynamic read-only cache group with incremental autorefresh, you can reduce contention and improve performance by enabling the `DynamicLoadReduceContention` database system parameter. See "Reducing contention on TimesTen for dynamic read-only cache groups with incremental autorefresh" on page 7-10 for more details.

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

Replicating cache tables

To achieve high availability, configure an active standby pair replication scheme for cache tables in a read-only cache group or an AWT cache group.

An active standby pair that replicates cache tables from one of these cache group types can automatically change the role of a TimesTen database as part of failover and recovery with minimal chance of data loss. Cache groups themselves provide resilience from Oracle database outages, further strengthening system availability. An active standby pair replication scheme provides for high availability of a TimesTen database.

Note: This section describes one scenario in including cache groups within an active standby pair replication scheme. See "Administering an Active Standby Pair with Cache Groups" in *Oracle TimesTen In-Memory Database Replication Guide* for more scenarios for including AWT and read-only cache groups in an active standby pair replication scheme.

Oracle Real Application Clusters (Oracle RAC) provides for high availability of an Oracle database. For more information about using TimesTen Cache in an Oracle RAC environment, see "Using TimesTen Cache in an Oracle RAC Environment" on page 9-1.

Perform the following tasks to configure an active standby pair for TimesTen databases that cache Oracle Database tables:

- Create and configure the active database
- Create and configure the standby database
- Create and configure the read-only subscriber database

Create and configure the active database

The following is the definition of the `cacheactive` DSN for the active database of the active standby pair:

```
[cacheactive]
DataStore=/users/OracleCache/cacheact
PermSize=64
OracleNetServiceName=orcl
DatabaseCharacterSet=WE8ISO8859P1
```

Start the `ttIsql` utility and connect to the `cacheactive` DSN as the instance administrator to create the database. Then create the cache manager user `cacheuser` whose name is the same as a companion Oracle Database user. In this example, the cache administration user is acting as the companion Oracle Database user.

Then create a cache table user `oratt` whose name is the same as the Oracle Database schema user who owns the Oracle Database tables to be cached in the TimesTen database.

```
% ttIsql cacheactive
Command> CREATE USER cacheuser IDENTIFIED BY timesten;
Command> CREATE USER oratt IDENTIFIED BY timesten;
```

As the instance administrator, use the `ttIsql` utility to grant the cache manager user `cacheuser` the privileges required to perform the operations listed in Example 3–8 as well as create an active standby pair replication scheme which requires the `ADMIN` privilege:

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

Start the `ttIsql` utility and connect to the `cacheactive` DSN as the cache manager user. Set the cache administration user name and password by calling the `ttCacheUidPwdSet` built-in procedure.

```
% ttIsql "DSN=cacheactive;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> CALL ttCacheUidPwdSet('cacheuser','oracle');
```

If desired, you can test the connectivity between the active database and the Oracle database using the instructions stated in "Testing the connectivity between the TimesTen and Oracle databases" on page 3-10.

Start the cache agent on the active database by calling the `ttCacheStart` built-in procedure as the cache manager user:

```
Command> CALL ttCacheStart;
```

The following statement is the definition of the Oracle Database table that is to be cached in a dynamic AWT cache group. The Oracle Database table is owned by the schema user `oratt`. The `oratt` user must be granted the `CREATE SESSION` and `RESOURCE` privileges before it can create tables.

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

The Oracle Database user with the same name as the TimesTen cache manager user must be granted the `SELECT` privilege on the `oratt.subscriber` table so that the cache manager user can create an AWT cache group that caches this table. The cache administration user must be granted the `INSERT`, `UPDATE` and `DELETE` Oracle Database privileges for the `oratt.subscriber` table for asynchronous writethrough operations to be applied to the Oracle Database.

Then, create cache groups in the TimesTen database with the `CREATE DYNAMIC ASYNCHRONOUS WRITETHROUGH CACHE GROUP` statement as the cache manager user. For example, the following statement creates a dynamic AWT cache group `subscriber_accounts` that caches the `oratt.subscriber` table:

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

As the cache manager user, create an active standby pair replication scheme in the active database using a `CREATE ACTIVE STANDBY PAIR` statement.

In the following example, `cacheact`, `cachestand` and `subscr` are the file name prefixes of the checkpoint and transaction log files of the active database, standby database and read-only subscriber database. `sys3`, `sys4` and `sys5` are the host names of the TimesTen systems where the active database, standby database and read-only subscriber database reside, respectively.

```
Command> CREATE ACTIVE STANDBY PAIR cacheact ON "sys3", cachestand ON "sys4"
        SUBSCRIBER subscr ON "sys5";
```

As the cache manager user, start the replication agent on the active database by calling the `ttRepStart` built-in procedure. Then declare the database as the active by calling the `ttRepStateSet` built-in procedure.

```
Command> CALL ttRepStart;
Command> CALL ttRepStateSet('active');
```

Create and configure the standby database

The following is the definition of the `cachestandby` DSN for the standby database of the active standby pair:

```
[cachestandby]
DataStore=/users/OracleCache/cachestand
PermSize=64
OracleNetServiceName=orcl
DatabaseCharacterSet=WE8ISO8859P1
```

As the instance administrator, create the standby database as a duplicate of the active database by running a `ttRepAdmin -duplicate` utility command from the standby database system. The instance administrator user name of the active database's and standby database's instances must be identical.

Use the `-keepCG` option so that cache tables in the active database are duplicated as cache tables in the standby database, because the standby database is connected with the Oracle database.

In the following example:

- The `-from` option specifies the file name prefix of the active database's checkpoint and transaction log files.
- The `-host` option specifies the host name of the TimesTen system where the active database resides.
- The `-uid` and `-pwd` options specify a user name and password of a TimesTen internal user defined in the active database that has been granted the `ADMIN` privilege.
- The `-cacheuid` and `-cachepwd` options specify the cache administration user name and password.
- `cachestandby` is the DSN of the standby database.
- The `-keepCG` option specifies that the standby database keeps the cache groups defined on the active database.

```
% ttRepAdmin -duplicate -from cacheact -host "sys3" -uid cacheuser -pwd timesten
    -cacheuid cacheuser -cachepwd oracle -keepCG cachestandby
```

Start the `ttIsql` utility and connect to the `cachestandby` DSN as the cache manager user. Set the cache administration user name and password by calling the `ttCacheUidPwdSet` built-in procedure.

```
% ttIsql "DSN=cachestandby;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> CALL ttCacheUidPwdSet('cacheuser','oracle');
```

If desired, you can test the connectivity between the standby database and the Oracle database using the instructions stated in "Testing the connectivity between the TimesTen and Oracle databases" on page 3-10.

Start the cache agent on the standby database by calling the `ttCacheStart` built-in procedure as the cache manager user:

```
Command> CALL ttCacheStart;
```

As the cache manager user, start the replication agent on the standby database by calling the `ttRepStart` built-in procedure.

```
Command> CALL ttRepStart;
```

Create and configure the read-only subscriber database

The following is the definition of the `rosubscriber` DSN for the read-only subscriber database of the active standby pair:

```
[rosubscriber]
DataStore=/users/OracleCache/subscr
PermSize=64
DatabaseCharacterSet=WE8ISO8859P1
```

As the instance administrator, create the read-only subscriber database as a duplicate of the standby database by running a `ttRepAdmin -duplicate` utility command from the read-only subscriber database system. The instance administrator user name of the standby database and read-only subscriber database must be identical.

Use the `-noKeepCG` option so that cache tables in the standby database are duplicated as regular tables in the read-only subscriber database because the read-only subscriber database is not connected with the Oracle database.

In the following example:

- The `-from` option specifies the file name prefix of the standby database's checkpoint and transaction log files.
- The `-host` option specifies the host name of the TimesTen system where the standby database resides.
- The `-uid` and `-pwd` options specify a user name and password of a TimesTen internal user defined in the standby database that has been granted the `ADMIN` privilege.
- `rosubscriber` is the DSN of the read-only subscriber database.

```
% ttRepAdmin -duplicate -from cachestand -host "sys4" -uid cacheuser -pwd timesten
    -noKeepCG rosubscriber
```

As the cache manager user, start the replication agent on the read-only subscriber database by calling the `ttRepStart` built-in procedure.

```
% ttIsql "DSN=rosubscriber;UID=cacheuser;PWD=timesten"
Command> CALL ttRepStart;
Command> exit
```

5

Cache Group Operations

The following sections describe operations that can be performed on cache groups:

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

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

Transmitting updates between the TimesTen and Oracle databases

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

SQL statement	Description
`LOAD CACHE GROUP`	Load cache instances that are not in the TimesTen cache tables from the cached Oracle Database tables.
`REFRESH CACHE GROUP`	Replace cache instances in the TimesTen cache tables with current data from the cached Oracle Database tables.
`FLUSH CACHE GROUP`	Propagate committed updates on the TimesTen cache tables to the cached Oracle Database tables. Only applicable for user managed cache groups.

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

See "Asynchronous WriteThrough (AWT) cache group" on page 4-11 for more information about AWT cache groups.

See "Synchronous WriteThrough (SWT) cache group" on page 4-25 for more information about SWT cache groups.

See "PROPAGATE cache table attribute" on page 4-29 for more information about using the `PROPAGATE` cache table attribute on cache tables in a user managed cache group.

The `AUTOREFRESH` cache group attribute can be used in a read-only or a user managed cache group to automatically refresh committed updates on cached Oracle Database tables into the TimesTen cache tables. Automatic refresh can be defined on explicitly loaded or dynamic cache groups.

See "AUTOREFRESH cache group attribute" on page 4-34 for more information about automatically refreshing a cache group.

Data is manually preloaded into the cache tables of explicitly loaded cache groups. For dynamic cache groups, data is loaded on demand into the cache tables. A cache instance is automatically loaded from the cached Oracle Database tables when a particular statement does not find the data in the cache tables.

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

Dynamic cache groups are typically configured to automatically age out from the cache tables data that is no longer being used.

Loading and refreshing a cache group

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

- `LOAD CACHE GROUP` only loads committed inserts on the cached Oracle Database tables into the TimesTen cache tables. New cache instances are loaded into the cache tables, but cache instances that already exist in the cache tables are not updated or deleted even if the corresponding rows in the cached Oracle Database tables have been updated or deleted. A load operation is primarily used to initially populate a cache group.
- `REFRESH CACHE GROUP` replaces cache instances in the TimesTen cache tables with the most current data from the cached Oracle Database tables including cache instances that are already exist in the cache tables. A refresh operation is primarily used to update the contents of a cache group with committed updates on the cached Oracle Database tables after the cache group has been initially populated.

  For an explicitly loaded cache group, a refresh operation is equivalent to issuing an `UNLOAD CACHE GROUP` statement followed by a `LOAD CACHE GROUP` statement on the cache group. In effect, all committed inserts, updates and deletes on the cached Oracle Database tables are refreshed into the cache tables. New cache instances may be loaded into the cache tables. Cache instances that already exist in the cache tables are updated or deleted if the corresponding rows in the cached Oracle Database tables have been updated or deleted. See "Unloading a cache group" on page 5-16 for more information about the `UNLOAD CACHE GROUP` statement.

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

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

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

If the cache group has a time-based aging policy defined, only cache instances where the timestamp in the root table's row is within the aging policy's lifetime are loaded or refreshed into the cache tables.

To prevent a load or refresh operation from processing a large number of cache instances within a single transaction, which can greatly reduce concurrency and throughput, use the `COMMIT EVERY` *n* `ROWS` clause to specify a commit frequency unless you are using the `WITH ID` clause. If you specify `COMMIT EVERY 0 ROWS`, the load or refresh operation is processed in a single transaction.

A `LOAD CACHE GROUP` or `REFRESH CACHE GROUP` statement that uses the `COMMIT EVERY` *n* `ROWS` clause must be performed in its own transaction without any other operations within the same transaction.

Example 5–1   Loading a cache group

The following statement loads new cache instances into the TimesTen cache tables in the `customer_orders` cache group from the cached Oracle Database tables:

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

Example 5–2   Loading a cache group using a WHERE clause

The following statement loads into the TimesTen cache tables in the `new_customers` cache group from the cached Oracle Database tables, new cache instances for customers whose customer number is greater than or equal to 5000:

```
LOAD CACHE GROUP new_customers WHERE (oratt.customer.cust_num >= 5000)
  COMMIT EVERY 256 ROWS;
```

Example 5–3   Refreshing a cache group

The following statement refreshes cache instances in the TimesTen cache tables within the `top_products` cache group from the cached Oracle Database tables:

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

Example 5–4   Refreshing a cache group using a WHERE clause

The following statement refreshes in the TimesTen cache tables within the `update_anywhere_customers` cache group from the cached Oracle Database tables, cache instances of customers located in zip code 60610:

```
REFRESH CACHE GROUP update_anywhere_customers
  WHERE (oratt.customer.zip = '60610') COMMIT EVERY 256 ROWS;
```

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

The rest of this section includes these topics:

- Loading and refreshing an explicitly loaded cache group with autorefresh

- Loading and refreshing a dynamic cache group with autorefresh
- Loading and refreshing a cache group using a WITH ID clause
- Initiating an immediate autorefresh
- Loading and refreshing a multiple-table cache group
- Improving the performance of loading or refreshing a large number of cache instances
- Example of manually loading and refreshing an explicitly loaded cache group
- Example of manually loading and refreshing a dynamic cache group

Loading and refreshing an explicitly loaded cache group with autorefresh

If the autorefresh state of an explicitly loaded cache group is `PAUSED`, the autorefresh state is changed to `ON` after a `LOAD CACHE GROUP` or `REFRESH CACHE GROUP` statement issued on the cache group completes.

The following restrictions apply when manually loading or refreshing an explicitly loaded cache group with autorefresh:

- A `LOAD CACHE GROUP` statement can only be issued if the cache tables are empty.
- The autorefresh state must be `PAUSED` before you can issue a `LOAD CACHE GROUP` statement.
- The autorefresh state must be `PAUSED` before you can issue a `REFRESH CACHE GROUP` statement.
- A `LOAD CACHE GROUP` statement cannot contain a `WHERE` clause.
- A `LOAD CACHE GROUP` or `REFRESH CACHE GROUP` statement cannot contain a `WITH ID` clause.
- A `REFRESH CACHE GROUP` statement cannot contain a `WHERE` clause.
- All tables and columns referenced in a `WHERE` clause when loading the cache group must be fully qualified. For example:

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

When an autorefresh operation occurs on an explicitly loaded cache group, all committed inserts, updates and deletes on the cached Oracle Database tables since the last autorefresh cycle are refreshed into the cache tables. New cache instances may be loaded into the cache tables. Cache instances that already exist in the cache tables are updated or deleted if the corresponding rows in the cached Oracle Database tables have been updated or deleted.

Loading and refreshing a dynamic cache group with autorefresh

If the autorefresh state of a dynamic cache group is `PAUSED`, the autorefresh state is changed to `ON` after any of the following events occur:

- Its cache tables are initially empty, and then a dynamic load, a `LOAD CACHE GROUP` or an unconditional `REFRESH CACHE GROUP` statement issued on the cache group completes.
- Its cache tables are not empty, and then an unconditional `REFRESH CACHE GROUP` statement issued on the cache group completes.

If the autorefresh state of a dynamic cache group is `PAUSED`, the autorefresh state remains at `PAUSED` after any of the following events occur:

- Its cache tables are initially empty, and then a `REFRESH CACHE GROUP ... WITH ID` statement issued on the cache group completes.
- Its cache tables are not empty, and then a dynamic load, a `REFRESH CACHE GROUP ... WITH ID`, or a `LOAD CACHE GROUP` statement issued on the cache group completes.

For a dynamic cache group, an autorefresh operation only refreshes committed updates and deletes on the cached Oracle Database tables since the last autorefresh cycle into the cache tables because only existing cache instances in the cache tables are refreshed. New cache instances are not loaded into the cache tables. To load new cache instances into the cache tables of a dynamic cache group, use a `LOAD CACHE GROUP` statement or perform a dynamic load operation. See "Dynamically loading a cache instance" on page 5-10 for more information about a dynamic load operation.

The following restrictions apply when manually loading or refreshing a dynamic cache group with automatic refresh:

- The autorefresh state must be `PAUSED` or `ON` before you can issue a `LOAD CACHE GROUP` statement.
- The autorefresh state must be `PAUSED` before you can issue a `REFRESH CACHE GROUP` statement.
- A `LOAD CACHE GROUP` statement that contains a `WHERE` clause must include a `COMMIT EVERY` *n* `ROWS` clause after the `WHERE` clause.
- A `REFRESH CACHE GROUP` statement cannot contain a `WHERE` clause.
- All tables and columns referenced in a `WHERE` clause when loading the cache group must be fully qualified. For example:

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

Loading and refreshing a cache group using a WITH ID clause

The `WITH ID` clause of the `LOAD CACHE GROUP` or `REFRESH CACHE GROUP` statement enables you to load or refresh a cache group based on values of the primary key columns without having to use a `WHERE` clause. The `WITH ID` clause is more convenient than the equivalent `WHERE` clause if the primary key contains more than one column. Using the `WITH ID` clause allows you to load one cache instance at a time. It also enables you to roll back the transaction containing the load or refresh operation, if necessary, unlike the equivalent statement that uses a `WHERE` clause because using a `WHERE` clause also requires specifying a `COMMIT EVERY` *n* `ROWS` clause.

Example 5–5 Loading a cache group using a WITH ID clause

A cache group recent_orders contains a single cache table `oratt.orderdetails` with a primary key of `(orderid, itemid)`. If a customer calls about an item within a particular order, the information can be obtained by loading the cache instance for the specified order number and item number.

Load the `oratt.orderdetails` cache table in the `recent_orders` cache group with the row whose value in the `orderid` column of the `oratt.orderdetails` cached Oracle Database table is 1756 and its value in the `itemid` column is 573:

```
LOAD CACHE GROUP recent_orders WITH ID (1756,573);
```

The following is an equivalent `LOAD CACHE GROUP` statement that uses a `WHERE` clause:

```
LOAD CACHE GROUP recent_orders WHERE orderid = 1756 and itemid = 573
  COMMIT EVERY 256 ROWS;
```

A `LOAD CACHE GROUP` or `REFRESH CACHE GROUP` statement issued on an autorefresh cache group cannot contain a `WITH ID` clause unless the cache group is dynamic.

You cannot use the `COMMIT EVERY` *n* `ROWS` clause with the `WITH ID` clause.

Initiating an immediate autorefresh

If the Oracle Database tables have been updated with data that needs to be applied to cache tables without waiting for the next autorefresh operation, you can call the `ttCacheAutorefresh` built-in procedure. The `ttCacheAutorefresh` built-in procedure initiates an immediate refresh operation and resets the autorefresh cycle to start at the moment you invoke `ttCacheAutorefresh`. The refresh operation is full or incremental depending on how the cache group is configured. The autorefresh state must be `ON` when `ttCacheAutorefresh` is called.

The autorefresh operation normally refreshes all cache groups sharing the same refresh interval in one transaction in order to preserve transactional consistency across these cache groups. Therefore, although you specify a specific cache group when you call `ttCacheAutorefresh`, the autorefresh operation occurs in one transaction for all cache groups that share the autorefresh interval with the specified cache group. If there is an existing transaction with table locks on objects that belong to the affected cache groups, `ttCacheAutofresh` returns an error without taking any action.

You can choose to run `ttCacheAutorefresh` asynchronously (the default) or synchronously. In synchronous mode, `ttCacheAutorefresh` returns an error if the refresh operation fails.

After calling `ttCacheAutorefresh`, you must commit or roll back the transaction before subsequent work can be performed.

Example 5–6 Calling ttCacheAutorefresh

This example calls `ttCacheAutorefresh` for the `ttuser.western_customers` cache group, using asynchronous mode.

```
Command> call ttCacheAutorefresh('ttuser', 'western_customers');
```

Loading and refreshing a multiple-table cache group

If you are loading or refreshing a multiple-table cache group while the cached Oracle Database tables are concurrently being updated, set the isolation level in the TimesTen database to serializable before issuing the `LOAD CACHE GROUP` or `REFRESH CACHE GROUP` statement. This causes TimesTen to query the cached Oracle Database tables in a serializable fashion during the load or refresh operation so that the loaded or refreshed cache instances in the cache tables are guaranteed to be transactionally consistent with the corresponding rows in the cached Oracle Database tables. After you have loaded or refreshed the cache group, set the isolation level back to read committed for better concurrency when accessing elements in the TimesTen database.

Improving the performance of loading or refreshing a large number of cache instances

You can improve the performance of loading or refreshing a large number of cache instances into a cache group by using the `PARALLEL` clause of the `LOAD CACHE GROUP` or `REFRESH CACHE GROUP` statement. Specify the number of threads to use when processing the load or refresh operation. You can specify 1 to 10 threads. One thread fetches rows from the cached Oracle Database tables, while the other threads insert the rows into the TimesTen cache tables. Do not specify more threads than the number of

CPUs available on your system or you may encounter decreased performance than if you had not used the `PARALLEL` clause.

Note: You cannot use the `WITH ID` clause with the `PARALLEL` clause. You can use the `COMMIT EVERY n ROWS` clause with the `PARALLEL` clause as long as `n` is greater than 0. In addition, you cannot use the `PARALLEL` clause for read-only dynamic cache groups or when database level locking is enabled. For more details, see "REFRESH CACHE GROUP" in the *Oracle TimesTen In-Memory Database SQL Reference*.

Example 5–7 Refreshing a cache group using a PARALLEL clause

The following statement refreshes cache instances in the TimesTen cache tables within the `western_customers` cache group from the cached Oracle Database tables using one thread to fetch rows from the cached Oracle Database tables and three threads to insert the rows into the cache tables:

```
REFRESH CACHE GROUP western_customers COMMIT EVERY 256 ROWS PARALLEL 4;
```

Example of manually loading and refreshing an explicitly loaded cache group

The following is the definition of the Oracle Database table that is to be cached in an explicitly loaded AWT cache group. The Oracle Database table is owned by the schema user `oratt`.

```
CREATE TABLE customer
(cust_num NUMBER(6) NOT NULL PRIMARY KEY,
 region   VARCHAR2(10),
 name     VARCHAR2(50),
 address  VARCHAR2(100));
```

The following is the data in the `oratt.customer` cached Oracle Database table.

```
CUST_NUM   REGION    NAME              ADDRESS
--------   -------   ---------------   ---------------------------
       1   West      Frank Edwards     100 Pine St. Portland OR
       2   East      Angela Wilkins    356 Olive St. Boston MA
       3   Midwest   Stephen Johnson   7638 Walker Dr. Chicago IL
```

The following statement creates an explicitly loaded AWT cache group `new_customers` that caches the `oratt.customer` table:

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

The `oratt.customer` TimesTen cache table is initially empty.

```
Command> SELECT * FROM oratt.customer;
0 rows found.
```

The following `LOAD CACHE GROUP` statement loads the three cache instances from the cached Oracle Database table into the TimesTen cache table:

```
Command> LOAD CACHE GROUP new_customers COMMIT EVERY 256 ROWS;
```

```
3 cache instances affected.
Command> SELECT * FROM oratt.customer;
< 1, West, Frank Edwards, 100 Pine St. Portland OR >
< 2, East, Angela Wilkins, 356 Olive St. Boston MA >
< 3, Midwest, Stephen Johnson, 7638 Walker Dr. Chicago IL >
```

Update the cached Oracle Database table by inserting a new row, updating an existing row, and deleting an existing row:

```
SQL> INSERT INTO customer
  2  VALUES (4, 'East', 'Roberta Simon', '3667 Park Ave. New York NY');
SQL> UPDATE customer SET name = 'Angela Peterson' WHERE cust_num = 2;
SQL> DELETE FROM customer WHERE cust_num = 3;
SQL> COMMIT;
SQL> SELECT * FROM customer;
CUST_NUM   REGION    NAME              ADDRESS
--------   -------   ---------------   ---------------------------
       1   West      Frank Edwards     100 Pine St. Portland OR
       2   East      Angela Peterson   356 Olive St. Boston MA
       4   East      Roberta Simon     3667 Park Ave. New York NY
```

A `REFRESH CACHE GROUP` statement issued on an explicitly loaded cache group is processed by unloading and then reloading the cache group. As a result, the cache instances in the cache table matches the rows in the cached Oracle Database table.

```
Command> REFRESH CACHE GROUP new_customers COMMIT EVERY 256 ROWS;
3 cache instance affected.
Command> SELECT * FROM oratt.customer;
< 1, West, Frank Edwards, 100 Pine St. Portland OR >
< 2, East, Angela Peterson, 356 Olive St. Boston MA >
< 4, East, Roberta Simon, 3667 Park Ave. New York NY >
```

Example of manually loading and refreshing a dynamic cache group

The following is the definition of the Oracle Database table that is to be cached in a dynamic AWT cache group. The Oracle Database table is owned by the schema user `oratt`.

```
CREATE TABLE customer
(cust_num NUMBER(6) NOT NULL PRIMARY KEY,
 region   VARCHAR2(10),
 name     VARCHAR2(50),
 address  VARCHAR2(100));
```

The following is the data in the `oratt.customer` cached Oracle Database table.

```
CUST_NUM   REGION    NAME              ADDRESS
--------   -------   ---------------   ---------------------------
       1   West      Frank Edwards     100 Pine St. Portland OR
       2   East      Angela Wilkins    356 Olive St. Boston MA
       3   Midwest   Stephen Johnson   7638 Walker Dr. Chicago IL
```

The following statement creates a dynamic AWT cache group `new_customers` that caches the `oratt.customer` table:

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

The `oratt.customer` TimesTen cache table is initially empty:

```
Command> SELECT * FROM oratt.customer;
0 rows found.
```

The following `LOAD CACHE GROUP` statement loads the three cache instances from the cached Oracle Database table into the TimesTen cache table:

```
Command> LOAD CACHE GROUP new_customers COMMIT EVERY 256 ROWS;
3 cache instances affected.
Command> SELECT * FROM oratt.customer;
< 1, West, Frank Edwards, 100 Pine St. Portland OR >
< 2, East, Angela Wilkins, 356 Olive St. Boston MA >
< 3, Midwest, Stephen Johnson, 7638 Walker Dr. Chicago IL >
```

Update the cached Oracle Database table by inserting a new row, updating an existing row, and deleting an existing row:

```
SQL> INSERT INTO customer
  2  VALUES (4, 'East', 'Roberta Simon', '3667 Park Ave. New York NY');
SQL> UPDATE customer SET name = 'Angela Peterson' WHERE cust_num = 2;
SQL> DELETE FROM customer WHERE cust_num = 3;
SQL> COMMIT;
SQL> SELECT * FROM customer;
CUST_NUM   REGION    NAME              ADDRESS
--------   -------   ---------------   ---------------------------
       1   West      Frank Edwards     100 Pine St. Portland OR
       2   East      Angela Peterson   356 Olive St. Boston MA
       4   East      Roberta Simon     3667 Park Ave. New York NY
```

A `REFRESH CACHE GROUP` statement issued on a dynamic cache group only refreshes committed updates and deletes on the cached Oracle Database tables into the cache tables. New cache instances are not loaded into the cache tables. Therefore, only existing cache instances are refreshed. As a result, the number of cache instances in the cache tables are either fewer than or the same as the number of rows in the cached Oracle Database tables.

```
Command> REFRESH CACHE GROUP new_customers COMMIT EVERY 256 ROWS;
2 cache instances affected.
Command> SELECT * FROM oratt.customer;
< 1, West, Frank Edwards, 100 Pine St. Portland OR >
< 2, East, Angela Peterson, 356 Olive St. Boston MA >
```

A subsequent `LOAD CACHE GROUP` statement loads one cache instance from the cached Oracle Database table into the TimesTen cache table because only committed inserts are loaded into the cache table. Therefore, only new cache instances are loaded. Cache instances that already exist in the cache tables are not changed because of a `LOAD CACHE GROUP` statement, even if the corresponding rows in the cached Oracle Database tables were updated or deleted.

```
Command> LOAD CACHE GROUP new_customers COMMIT EVERY 256 ROWS;
1 cache instance affected.
Command> SELECT * FROM oratt.customer;
< 1, West, Frank Edwards, 100 Pine St. Portland OR >
< 2, East, Angela Peterson, 356 Olive St. Boston MA >
< 4, East, Roberta Simon, 3667 Park Ave. New York NY >
```

Dynamically loading a cache instance

In a dynamic cache group, data is automatically loaded into the TimesTen cache tables from the cached Oracle Database tables when a qualifying `SELECT`, `INSERT`, `UPDATE`, or `DELETE` statement is issued on one of the cache tables and the data does not exist in the cache table but does exist in the cached Oracle Database table.

Note: If the Oracle database is down, the following error is returned:

```
5219: Temporary Oracle connection failure error in
OCISessionBegin():
ORA-01034: ORACLE not available
```

A dynamic load retrieves a single cache instance that is automatically loaded from the Oracle database to the TimesTen database. A cache instance consists of row from the root table of any cache group (that is uniquely identified by either a primary key or a unique index on the root table) and all related rows in the child tables associated by foreign key relationships.

If a row in the cached Oracle Database table satisfies the `WHERE` clause, the entire associated cache instance is loaded in order to maintain the defined relationships between primary keys and foreign keys of the parent and child tables. A dynamic load operation cannot load more than one row into the root table of any cache group. Only cache instances whose rows satisfy the `WHERE` clause of the cache table definitions are loaded.

The `WHERE` clause must specify one of the following for a dynamic load to occur:

- An equality condition with constants and/or parameters on all columns of a primary key or a foreign key of any table of the cache group. If more than one table of a cache group is referenced, each must be connected by an equality condition on the primary or foreign key relationship.
- A mixture of equality or `IS NULL` conditions on all columns of a unique index, provided that you use at least one equality condition. That is, you can perform a dynamic load where some columns of the unique index are `NULL`. The unique index must be created on the root table of the cache group.

Note: Dynamic loading based on a primary key search of the root table performs faster than primary key searches on a child table or foreign key searches on a child table.

The dynamic load is executed in a different transaction than the user transaction that triggers the dynamic load. The dynamic load transaction is committed before the SQL statement that triggers the dynamic load has finished execution. Thus, if the user transaction is rolled back, the dynamically loaded data remains in the cache group.

The following sections describes dynamic load for cache groups:

- Dynamic load configuration
- Dynamic load guidelines
- Examples of dynamically loading a cache instance
- Returning dynamic load errors

Dynamic load configuration

Dynamic load can be configured with the `DynamicLoadEnable` connection attribute as follows:

- 0 - Disables dynamic load of Oracle Database data to TimesTen dynamic cache groups for the current connection.
- 1 (default) - Enables dynamic load of Oracle Database data to a single TimesTen dynamic cache group per statement for the current connection. The statement must reference tables of only one dynamic cache group and only in the main query. The statement can also reference non-cache tables. Only one cache instance can be loaded.

Set the appropriate value in the `DynamicLoadEnable` connection attribute to configure the type of dynamic loading for all cache tables in dynamic cache groups that are accessed within a particular connection.

You can set the `DynamicLoadEnable` optimizer hint to temporarily enable or disable dynamic loading for a particular transaction. However, the `DynamicLoadEnable` connection attribute is the only method for configuring what type of dynamic load is enabled.

You can set the `DynamicLoadEnable` optimizer hint with one of the following methods:

- Use the `ttIsql` utility `set dynamicloadenable` command.
- Call the `ttOptSetFlag` built-in procedure with the `DynamicLoadEnable` flag set to the desired value. The following example sets dynamic loading to 1.

```
call ttOptSetFlag('DynamicLoadEnable', 1)
```

Note: For more details, see "DynamicLoadEnable", "ttIsql" or "ttOptSetFlag" in the *Oracle TimesTen In-Memory Database Reference*.

You can also set connection attributes with the `SQLSetConnectOption` ODBC function (ODBC 2.5) or the `SQLSetConnectAttr` function (ODBC 3.5). See the "Option support for ODBC 2.5 SQLSetConnectOption and SQLGetConnectOption" and "Attribute support for ODBC 3.5 SQLSetConnectAttr and SQLGetConnectAttr" sections in the *Oracle TimesTen In-Memory Database C Developer's Guide* for more details.

Dynamic load guidelines

Dynamic load retrieves at most one cache instance for each cache group referenced in the main query. This section details the guidelines under which dynamic load occurs.

Note: Examples for these guidelines are provided in "Examples of dynamically loading a cache instance" on page 5-12.

Dynamic load is available only for the following types of statements issued on a cache table in a dynamic cache group:

- When an `INSERT` statement inserts values into any of the child tables of a cache instance that does not currently exist in the TimesTen tables, the cache instance to which the new row belongs dynamically loads. The insert operation for the new child row is propagated to the cached Oracle Database table.

- `SELECT`, `UPDATE`, or `DELETE` statements require that the `WHERE` clause have the conditions as stated in "Dynamically loading a cache instance" on page 5-10.

The `SELECT`, `UPDATE`, or `DELETE` statements for which dynamic load is available must satisfy the following conditions:

- If the statement contains a subquery, only the cache group with tables referenced in the main query are considered for a dynamic load.
- If the statement references multiple tables of the cache group, the statement must include an equality join condition between the primary keys and foreign keys for all parent and child relationships.
- The statement cannot contain the `UNION`, `INTERSECT`, or `MINUS` set operators.
- The statement can reference non-cache tables.
- By default (`DynamicLoadEnable` = 1), the statement can reference cache tables from only one dynamic cache group. See "Dynamic load configuration" on page 5-11 for more information.

Dynamic load behavior depends on the setting of `DynamicLoadEnable`. The following describes the rules that are evaluated to determine if a dynamic load occurs when `DynamicLoadEnable` = 1.

- Dynamic load does not occur for a cache group if any table of the cache group is specified more than once in any `FROM` clause.
- Only the conditions explicitly specified in the query are considered for dynamic load, which excludes any derived conditions.
- If any cache group is referenced only in a subquery, it is not considered for a dynamic load.
- If the cache group has a time-based aging policy defined, the timestamp in the root table's row must be within the aging policy's lifetime in order for the cache instance to be loaded. See "Implementing aging in a cache group" on page 4-46 for information about defining an aging policy on a cache group.
- When using an active standby pair replication scheme, dynamic load cannot occur in any subscriber.

The following considerations can affect dynamic load:

- If tables within multiple cache groups or non-cache group tables are specified in the main query, the join order influences if the cache instance is loaded. If during the execution of the query, a dynamic load is possible and necessary to produce the query results, the dynamic load occurs. However, if no rows are returned, then some or all of the cache instances are not dynamically loaded.
- If a statement specifies more than the dynamic load condition on tables of a cache group, the cache instance may be dynamically loaded even though the additional conditions are not qualified for the statement.

Examples of dynamically loading a cache instance

The following is the definition of the Oracle Database tables that are to be cached in a dynamic AWT cache group. The Oracle Database table is owned by the schema user `oratt`.

```
CREATE TABLE customer
(cust_num NUMBER(6) NOT NULL PRIMARY KEY,
 region   VARCHAR2(10),
 name     VARCHAR2(50),
 address  VARCHAR2(100));

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

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

For example, the following data is in the `oratt.customer` cached Oracle Database table.

```
CUST_NUM   REGION    NAME              ADDRESS
--------   -------   ---------------   ---------------------------
       1   West      Frank Edwards     100 Pine St., Portland OR
       2   East      Angela Wilkins    356 Olive St., Boston MA
       3   Midwest   Stephen Johnson   7638 Walker Dr., Chicago IL
```

The following statement creates a dynamic AWT cache group `new_customers` that caches the `oratt.customer`, `oratt.orders`, and `oratt.orderdetails` tables:

```
CREATE DYNAMIC ASYNCHRONOUS WRITETHROUGH CACHE GROUP new_customers
FROM oratt.customer
 (cust_num NUMBER(6) NOT NULL,
  region   VARCHAR2(10),
  name     VARCHAR2(50),
  address  VARCHAR2(100),
  PRIMARY KEY(cust_num)),
oratt.orders
 (ord_num      NUMBER(10) NOT NULL,
  cust_num     NUMBER(6) NOT NULL,
  when_placed  DATE NOT NULL,
  when_shipped DATE NOT NULL,
  PRIMARY KEY(ord_num),
  FOREIGN KEY(cust_num) REFERENCES oratt.customer(cust_num)),
oratt.orderdetails
 (orderid  NUMBER(10) NOT NULL,
  itemid   NUMBER(8) NOT NULL,
  quantity NUMBER(4) NOT NULL,
  PRIMARY KEY(orderid, itemid),
  FOREIGN KEY(orderid) REFERENCES oratt.orders(order_num));
```

The following examples can be used when `DynamicLoadEnable` is set to 1:

The `oratt.customer` TimesTen cache table is initially empty:

```
Command> SELECT * FROM oratt.customer;
0 rows found.
```

The following `SELECT` statement with an equality condition on the primary key for the `oratt.customer` table results in a dynamic load:

```
Command> SELECT * FROM oratt.customer WHERE cust_num = 1;
< 1, West, Frank Edwards, 100 Pine St., Portland OR >
```

However, if you do not use an equality condition on the primary key, no dynamic load occurs:

```
Command> SELECT * FROM oratt.customer WHERE cust_num IN (1,2);
```

The following example contains equality expressions on all of the primary key columns for a primary key composite. The `orderdetails` table has a composite primary key of `orderid` and `itemid`.

```
UPDATE oratt.orderdetails SET quantity = 5 WHERE orderid=2280 AND itemid=663;
```

The following example shows an `INSERT` into the `orders` child table, which initiates a dynamic load. However, if you tried to insert into the `customer` table, which is the parent, no dynamic load occurs.

```
INSERT INTO orders VALUES(1,1, DATE '2012-01-25', DATE '2012-01-30');
```

The following `UPDATE` statement dynamically loads one cache instance from the cached Oracle Database table into the TimesTen cache table, updates the instance in the cache table, and then automatically propagates the update to the cached Oracle Database table:

```
Command> UPDATE oratt.customer SET name = 'Angela Peterson' WHERE cust_num = 2;
Command> SELECT * FROM oratt.customer;
< 1, West, Frank Edwards, 100 Pine St., Portland OR >
< 2, East, Angela Peterson, 356 Olive St., Boston MA >
```

The following is the updated data in the `oratt.customer` cached Oracle Database table:

```
CUST_NUM   REGION    NAME              ADDRESS
--------   -------   ---------------   ---------------------------
       1   West      Frank Edwards     100 Pine St., Portland OR
       2   East      Angela Peterson   356 Olive St., Boston MA
       3   Midwest   Stephen Johnson   7638 Walker Dr., Chicago IL
```

The following `DELETE` statement dynamically loads one cache instance from the cached Oracle Database table into the TimesTen cache table, deletes the instance from the cache table, and then automatically propagates the delete to the cached Oracle Database table:

```
Command> DELETE FROM oratt.customer WHERE cust_num = 3;
Command> SELECT * FROM oratt.customer;
< 1, West, Frank Edwards, 100 Pine St., Portland OR >
< 2, East, Angela Peterson, 356 Olive St., Boston MA >
```

The following is the updated data in the `oratt.customer` cached Oracle Database table.

```
CUST_NUM   REGION    NAME              ADDRESS
--------   -------   ---------------   ---------------------------
       1   West      Frank Edwards     100 Pine St., Portland OR
       2   East      Angela Peterson   356 Olive St., Boston MA
```

The following is an example of a dynamic load performed using all columns of a unique index on the root table. The `departments` table is defined in a dynamic AWT cache group. A unique index is created on this cache group consisting of the `manager_id` and `location_id`.

The following creates the departments table on the Oracle database.

```
Command> CREATE TABLE departments(
        department_id INT NOT NULL PRIMARY KEY,
        department_name VARCHAR(10) NOT NULL,
        technical_lead INT NOT NULL,
        manager_id INT,
        location_id INT NOT NULL);
```

The following creates the dynamic AWT cache group and a unique index on the `dept_cg` root table:

```
Command> CREATE DYNAMIC ASYNCHRONOUS WRITETHROUGH CACHE GROUP dept_cg
        FROM departments
        (department_id INT NOT NULL PRIMARY KEY,
         department_name VARCHAR(10) NOT NULL,
         technical_lead INT NOT NULL,
         manager_id INT, location_id INT NOT NULL);

Command> CREATE UNIQUE INDEX dept_idx ON departments(manager_id, location_id);
```

The following inserts three records into the departments table on the Oracle database:

```
Command> insert into departments values (1, 'acct', 1, 1, 100);
1 row inserted.
Command> insert into departments values (2, 'legal', 2, 2, 200);
1 row inserted.
Command> insert into departments values (3, 'owner', 3, NULL, 300);
1 row inserted.
Command> commit;
```

On TimesTen, dynamically load a cache instance based on the unique index:

```
Command> SELECT * FROM departments;
0 rows found.
Command> SELECT * FROM departments WHERE manager_id IS NULL AND location_id=300;
< 3, owner, 3, <NULL>, 300 >
1 row found.
Command> SELECT * FROM departments;
< 3, owner, 3, <NULL>, 300 >
1 row found.
Command> SELECT * FROM departments WHERE manager_id=2 AND location_id=200;
< 2, legal, 2, 2, 200 >
1 row found.
Command> SELECT * FROM departments;
< 2, legal, 2, 2, 200 >
< 3, owner, 3, <NULL>, 300 >
2 rows found.
```

Returning dynamic load errors

You can configure TimesTen to return an error if a `SELECT`, `UPDATE` or `DELETE` statement does not meet the requirements stated in "Dynamic load guidelines" on page 5-11. The `DynamicLoadErrorMode` connection attribute controls what happens when an application executes a SQL operation against a dynamic cache group and the SQL operation cannot use dynamic load in a particular connection.

- When `DynamicLoadErrorMode` is set to a value of 0, dynamic load happens to any cache group referenced in the query that is qualified for dynamic load. Cache groups that do not qualify are not dynamically loaded and no errors are returned. When `DynamicLoadEnable=1`, no dynamic load occurs if the query references more than one cache group.
- When `DynamicLoadErrorMode` is set to a value of 1, a query fails with an error if any dynamic cache group referenced in the query is not qualified for dynamic load. The error indicates the reason why the dynamic load cannot occur.

To set the connection attribute solely for a particular transaction, use one of the following:

- Use the `ttIsql` utility `set dynamicloaderrormode 1` command.
- Call the `ttOptSetFlag` built-in procedure with the `DynamicLoadErrorMode` flag and the optimizer value set to 1.

  ```
  call ttOptSetFlag('DynamicLoadErrorMode', 1)
  ```

  Call the `ttOptSetFlag` built-in procedure with the `DynamicLoadErrorMode` flag and the optimizer value set to 0 to suppress error reporting when a statement does not comply with dynamic load requirements.

Flushing a user managed cache group

The `FLUSH CACHE GROUP` statement manually propagates committed inserts and updates on TimesTen cache tables in a user managed cache group to the cached Oracle Database tables. Deletes are not flushed or manually propagated. Committed inserts and updates on cache tables that use the `PROPAGATE` cache table attribute cannot be flushed to the cached Oracle Database tables because these operations are already automatically propagated to the Oracle database.

With automatic propagation, committed inserts, updates and deletes are propagated to the Oracle database in the order they were committed in TimesTen. A flush operation can manually propagate multiple committed transactions on cache tables to the cached Oracle Database tables.

You cannot flush a user managed cache group that uses the `AUTOREFRESH` cache group attribute.

You can flush a user managed cache group if at least one of its cache tables uses neither the `PROPAGATE` nor the `READONLY` cache table attribute.

You can use a `WHERE` clause or `WITH ID` clause in a `FLUSH CACHE GROUP` statement to restrict the rows to be flushed to the cached Oracle Database tables. See the "FLUSH CACHE GROUP" statement in *Oracle TimesTen In-Memory Database SQL Reference* for more information.

Example 5–8 Flushing a cache group

The following statement manually propagates committed insert and update operations on the TimesTen cache tables in the `western_customers` cache group to the cached Oracle Database tables:

```
FLUSH CACHE GROUP western_customers;
```

Unloading a cache group

You can delete some or all cache instances from the cache tables in a cache group with the `UNLOAD CACHE GROUP` statement. Unlike the `DROP CACHE GROUP` statement, the cache tables themselves are not dropped when a cache group is unloaded.

Use caution when using the `UNLOAD CACHE GROUP` statement with autorefresh cache groups. An unloaded row can reappear in the cache table as the result of an autorefresh operation if the row, or its related parent or child rows, are updated in the cached Oracle Database table.

Execution of the `UNLOAD CACHE GROUP` statement for an AWT cache group waits until updates on the rows have been propagated to the Oracle database.

To prevent an unload operation from processing a large number of cache instances within a single transaction, which could reduce concurrency and throughput, use the `COMMIT EVERY` *n* `ROWS` clause to specify a commit frequency.

Note: For more information, see "UNLOAD CACHE GROUP" in the *Oracle TimesTen In-Memory Database SQL Reference*.

Example 5–9 Unloading cache groups

The following statement unloads all cache instances from all cache tables in the `customer_orders` cache group. A commit frequency is specified, so the operations is performed over several transactions by committing every 256 rows:

```
UNLOAD CACHE GROUP customer_orders COMMIT EVERY 256 ROWS;
```

The following statement unloads all cache instances from all cache tables in the `customer_orders` cache group in a single transaction. A single transaction should only be used if the data within `customer_orders` is small:

```
UNLOAD CACHE GROUP customer_orders;
```

The following equivalent statements delete the cache instance for customer number 227 from the cache tables in the `new_customers` cache group:

```
UNLOAD CACHE GROUP new_customers WITH ID (227);
UNLOAD CACHE GROUP new_customers WHERE (oratt.customer.cust_num = 227);
```

Determining the number of cache instances affected by an operation

You can use the following mechanisms to determine how many cache instances were loaded by a `LOAD CACHE GROUP` statement, refreshed by a `REFRESH CACHE GROUP` statement, flushed by a `FLUSH CACHE GROUP` statement, or unloaded by an `UNLOAD CACHE GROUP` statement:

- Call the `SQLRowCount()` ODBC function.
- Invoke the `Statement.getUpdateCount()` JDBC method.
- Call the `OCIAttrGet()` OCI function with the `OCI_ATTR_ROW_COUNT` option.

Setting a passthrough level

When an application issues statements on a TimesTen connection, the statement can be executed in the TimesTen database or passed through to the Oracle database for execution. Whether the statement is executed in the TimesTen or Oracle database depends on the composition of the statement and the setting of the `PassThrough` connection attribute. You can set the `PassThrough` connection attribute to define which statements are to be executed locally in TimesTen and which are to be redirected to the Oracle database for execution.

When appropriate within passthrough levels 1 through 3, TimesTen connects to the Oracle database using the current user’s credentials as the user name and the `OraclePwd` connection attribute as the Oracle password.

Note: A transaction that contains operations that are replicated with `RETURN TWOSAFE` cannot have a `PassThrough` setting greater than 0. If `PassThrough` is greater than 0, an error is returned and the transaction must be rolled back.

When `PassThrough` is set to 0, 1, or 2, the following behavior occurs when a dynamic load condition exists:

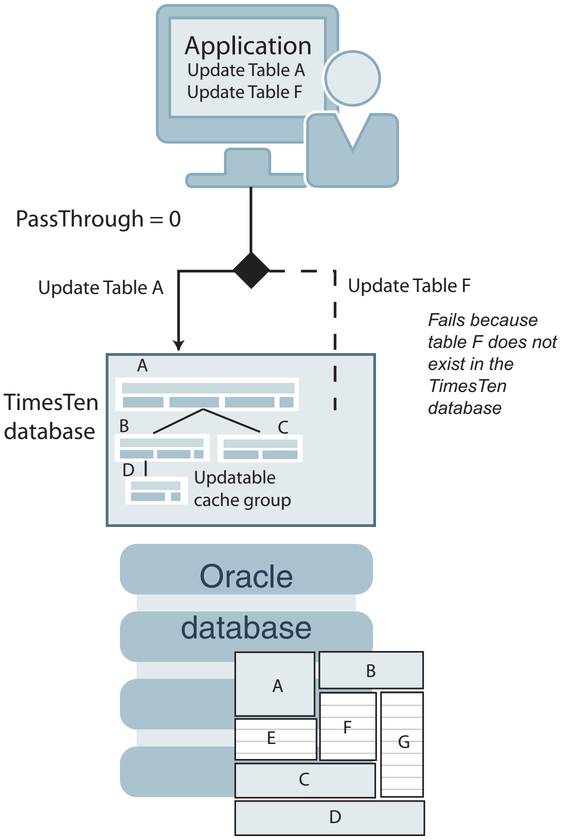
- A dynamic load can occur for a `SELECT` operation on cache tables in any dynamic cache group type.
- A dynamic load for an `INSERT`, `UPDATE`, or `DELETE` operation can only occur on cached tables with dynamic AWT or SWT cache groups.

See "Dynamically loading a cache instance" on page 5-10 for more details on dynamic load.

PassThrough=0

`PassThrough=0` is the default setting and specifies that all statements are to be executed in the TimesTen database. Figure 5–1 shows that Table A is updated on the TimesTen database. Table F cannot be updated because it does not exist in TimesTen.

Figure 5–1 PassThrough=0



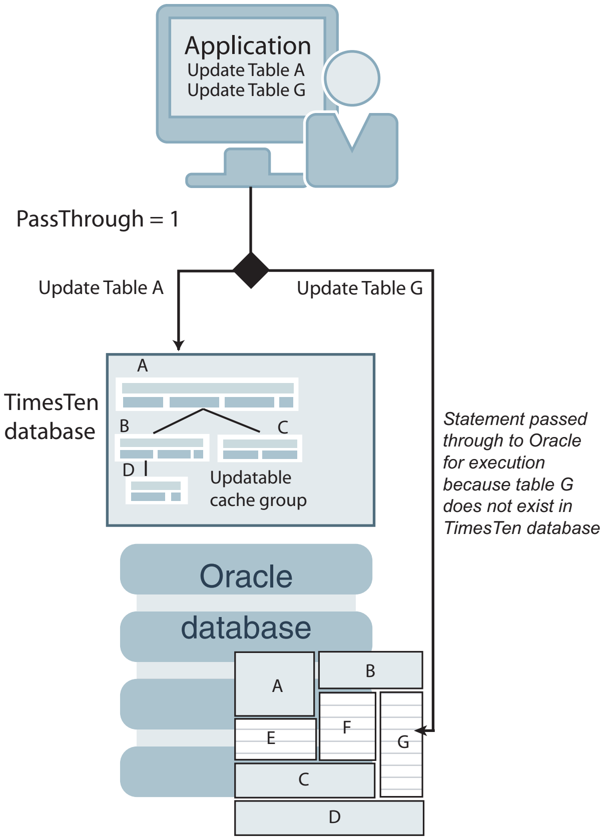
PassThrough=1

Set `PassThrough=1` to specify that a statement that references a table that does not exist in the TimesTen database is passed through to the Oracle database for execution. No DDL statements are passed through to the Oracle database.

If TimesTen cannot parse a `SELECT` statement because it includes keywords that do not exist in TimesTen SQL or because it includes syntax errors, it passes the statement to the Oracle database. If TimesTen cannot parse `INSERT`, `UPDATE` or `DELETE` statements, TimesTen returns an error and the statement is not passed through to the Oracle database.

Figure 5–2 shows that Table A is updated in the TimesTen database, while Table G is updated in the Oracle database because Table G does not exist in the TimesTen database.

Figure 5–2 PassThrough=1



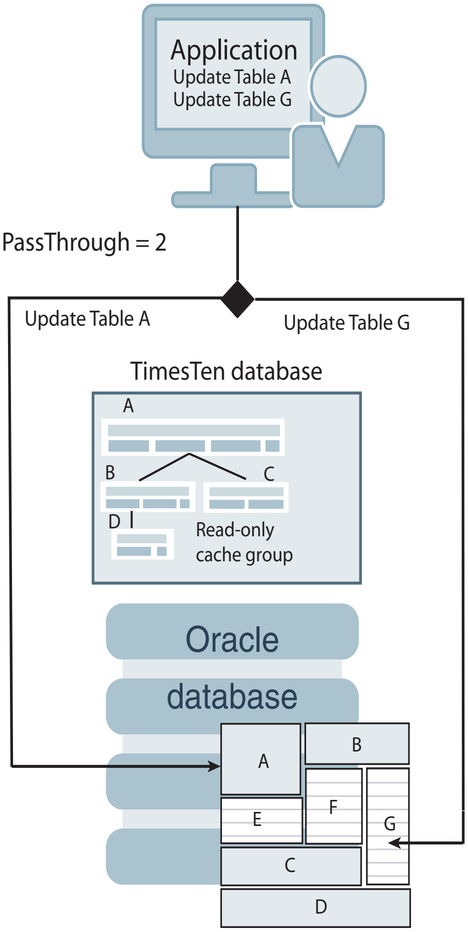
PassThrough=2

`PassThrough=2` specifies that `INSERT`, `UPDATE` and `DELETE` statements are passed through to the Oracle database for read-only cache groups and user managed cache groups that use the `READONLY` cache table attribute. Otherwise, `Passthrough=1` behavior applies.

Note: You are responsible in preventing conflicts that may occur if you update the same row in a TimesTen cache table as another user updates the cached Oracle Database table concurrently.

Figure 5–3 shows that updates to Table A and Table G in a read-only cache group are passed through to the Oracle database.

Figure 5–3 PassThrough=2



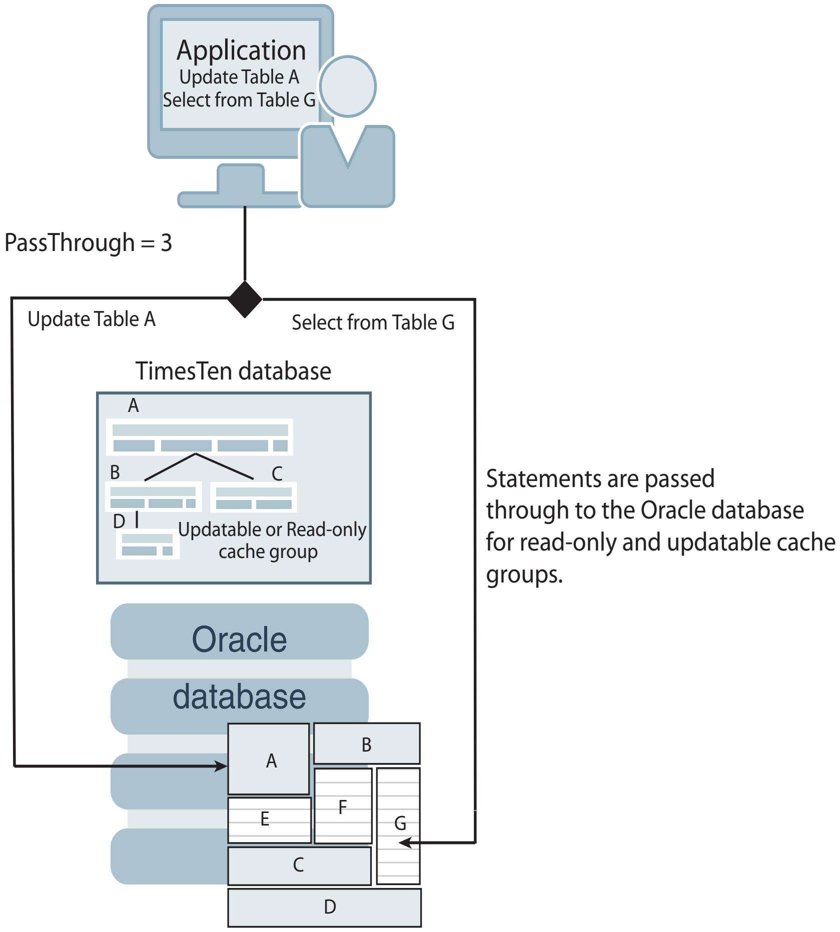
INSERT, UPDATE and DELETE statements are passed through to the Oracle database for read-only cache groups and read-only cache tables. SELECT statements are executed in TimesTen unless they contain invalid TimesTen syntax or reference tables that do not exist in TimesTen.

PassThrough=3

`PassThrough=3` specifies that all statements are passed through to the Oracle database for execution.

Figure 5–4 shows that Table A is updated on the Oracle database for a read-only or updatable cache group. A `SELECT` statement that references Table G is also passed through to the Oracle database.

Figure 5–4 PassThrough=3



Statements are passed through to the Oracle database for read-only and updatable cache groups.

Considerations for using passthrough

Passing through update operations to the Oracle database for execution is not recommended when issued on cache tables in an AWT or SWT cache group.

- Committed updates on cache tables in an AWT cache group are automatically propagated to the cached Oracle Database tables in asynchronous fashion. However, passing through an update operation to the Oracle database for execution within the same transaction as the update on the cache table in the AWT cache group renders the propagate of the cache table update synchronous, which may have undesired results.
- Committed updates on cache tables in an SWT cache group can result in self-deadlocks if, within the same transaction, updates on the same tables are passed through to the Oracle database for execution.

A PL/SQL block cannot be passed through to the Oracle database for execution. Also, you cannot pass through to Oracle Database for execution a reference to a stored procedure or function that is defined in the Oracle database but not in the TimesTen database.

For more information about how the `PassThrough` connection attribute setting determines which statements are executed in the TimesTen database and which are passed through to the Oracle database for execution and under what circumstances, see "PassThrough" in *Oracle TimesTen In-Memory Database Reference*.

Note: The passthrough feature uses OCI to communicate with the Oracle database. The OCI diagnostic framework installs signal handlers that may impact signal handling that you use in your application. You can disable OCI signal handling by setting `DIAG_SIGHANDLER_ENABLED=FALSE` in the `sqlnet.ora` file. Refer to "Fault Diagnosability in OCI" in *Oracle Call Interface Programmer's Guide* for information.

Changing the passthrough level for a connection or transaction

You can override the current passthrough level using the `ttIsql` utility's `set passthrough` command which applies to the current transaction.

You can also override the setting for a specific transaction by calling the `ttOptSetFlag` built-in procedure with the `PassThrough` flag. The following procedure call sets the passthrough level to 3:

```
CALL ttOptSetFlag('PassThrough', 3);
```

The `PassThrough` flag setting takes effect when a statement is prepared and it is the setting that is used when the statement is executed even if the setting has changed from the time the statement was prepared to when the statement is executed. After the transaction has been committed or rolled back, the original connection setting takes effect for all subsequently prepared statements.

6

Managing a Caching Environment

The following sections describe how to manage and monitor various aspects of a caching system such as cache groups and the cache agent process:

- Checking the status of the cache and replication agents
- Monitoring cache groups
- Managing a caching environment with Oracle Database objects
- Impact of failed autorefresh operations on TimesTen databases
- Dropping Oracle Database objects used by autorefresh cache groups
- Monitoring the cache administration user's tablespace
- Backing up and restoring a database with cache groups
- Changing cache user names and passwords

Checking the status of the cache and replication agents

You can use either the `ttAdmin` or `ttStatus` utility to check whether the TimesTen cache agent and replication agent processes are running as well as determine each agent's start policy.

Example 6–1 Using ttAdmin to determine the cache and replication agents status

You can use a `ttAdmin -query` utility command to determine whether the cache and replication agents are running, and the cache and replication agent start policies for a TimesTen database:

```
% ttAdmin -query cache1
RAM Residence Policy            : inUse
Replication Agent Policy        : manual
Replication Manually Started    : True
Cache Agent Policy              : always
Cache Agent Manually Started    : True
```

For more information about the `ttAdmin` utility, see "ttAdmin" in *Oracle TimesTen In-Memory Database Reference*.

Example 6–2 Using ttStatus to determine the cache and replication agents status

You can use the `ttStatus` utility to determine whether the cache and replication agents are running, and the cache and replication agent start policies for all TimesTen instances:

```
% ttStatus
```

```
TimesTen status report as of Thu May  7 13:42:01 2009

Daemon pid 9818 port 4173 instance myinst
TimesTen server pid 9826 started on port 4175
------------------------------------------------------------------------
Data store /users/OracleCache/ttcache
There are 38 connections to the data store
Shared Memory KEY 0x02011c82 ID 895844354
PL/SQL Memory KEY 0x03011c82 ID 895877123 Address 0x10000000
Type            PID     Context     Connection Name              ConnID
Cache Agent     1019    0x0828f840  Handler                           2
Cache Agent     1019    0x083a3d40  Timer                             3
Cache Agent     1019    0x0842d820  Aging                             4
Cache Agent     1019    0x08664fd8  Garbage Collector(-1580741728)    5
Cache Agent     1019    0x084d6ef8  Marker(-1580213344)               6
Cache Agent     1019    0xa5bb8058  DeadDsMonitor(-1579684960)        7
Replication     18051   0x08c3d900  RECEIVER                          8
Replication     18051   0x08b53298  REPHOLD                           9
Replication     18051   0x08af8138  REPLISTENER                      10
Replication     18051   0x08a82f20  LOGFORCE                         11
Replication     18051   0x08bce660  TRANSMITTER                      12
Subdaemon       9822    0x080a2180  Manager                        2032
Subdaemon       9822    0x080ff260  Rollback                       2033
Subdaemon       9822    0x08548c38  Flusher                        2034
Subdaemon       9822    0x085e3b00  Monitor                        2035
Subdaemon       9822    0x0828fc10  Deadlock Detector              2036
Subdaemon       9822    0x082ead70  Checkpoint                     2037
Subdaemon       9822    0x08345ed0  Aging                          2038
Subdaemon       9822    0x083a1030  Log Marker                     2039
Subdaemon       9822    0x083fc190  AsyncMV                        2040
Subdaemon       9822    0x084572f0  HistGC                         2041
Replication policy  : Manual
Replication agent is running.
Cache Agent policy  : Always
TimesTen's Cache agent is running for this data store
PL/SQL enabled.
------------------------------------------------------------------------
```

The information displayed by the `ttStatus` utility include the following that pertains to TimesTen Cache for each TimesTen instance:

- The names of the cache agent process threads that are connected to the TimesTen database
- The names of the replication agent process threads that are connected to the TimesTen database
- Status on whether the cache agent is running
- Status on whether the replication agent is running
- The cache agent start policy
- The replication agent start policy

For more information about the `ttStatus` utility, see "ttStatus" in *Oracle TimesTen In-Memory Database Reference*.

Cache agent and replication connections

When a connection from the cache agent to the Oracle database fails, the cache agent attempts to connect every 10 seconds. If the cache agent cannot connect to the Oracle database, the cache agent restarts after 10 minutes. This behavior repeats forever.

When a connection from the replication agent to the Oracle database fails, the replication agent attempts to reconnect to the Oracle database after 120 seconds. If it cannot reconnect after 120 seconds, the replication agent stops and does not restart.

If Fast Application Notification (FAN) is enabled on the Oracle database, the cache agent and the replication agent receive immediate notification of connection failures. If FAN is not enabled, the agents may wait until a TCP timeout occurs before becoming aware that the connection has failed.

If the Oracle Real Application Clusters (Oracle RAC) is enable on the Oracle database, along with FAN and Transparent Application Failover (TAF), then TAF manages the connection to a new Oracle Database instance. See Chapter 9, "Using TimesTen Cache in an Oracle RAC Environment".

Monitoring cache groups

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

- Using the ttIsql utility's cachegroups command
- Monitoring autorefresh operations on cache groups
- Monitoring AWT cache groups
- Tracking DDL statements issued on cached Oracle Database tables

Using the ttIsql utility's cachegroups command

You can obtain information about cache groups in a TimesTen database using the `ttIsql` utility's `cachegroups` command.

Example 6–3 ttIsql utility's cachegroups command

```
% ttIsql "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> cachegroups;

Cache Group CACHEUSER.RECENT_SHIPPED_ORDERS:

  Cache Group Type: Read Only
  Autorefresh: Yes
  Autorefresh Mode: Incremental
  Autorefresh State: On
  Autorefresh Interval: 1440 Minutes
  Autorefresh Status: ok
  Aging: Timestamp based uses column WHEN_SHIPPED lifetime 30 days cycle 24 hours
on

  Root Table: ORATT.ORDERS
  Table Type: Read Only


Cache Group CACHEUSER.SUBSCRIBER_ACCOUNTS:

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

  Root Table: ORATT.SUBSCRIBER
  Table Type: Propagate

Cache Group CACHEUSER.WESTERN_CUSTOMERS:

  Cache Group Type: User Managed
  Autorefresh: No
  Aging: No aging defined

  Root Table: ORATT.ACTIVE_CUSTOMER
  Where Clause: (oratt.active_customer.region = 'West')
  Table Type: Propagate

  Child Table: ORATT.ORDERTAB
  Table Type: Propagate

  Child Table: ORATT.ORDERDETAILS
  Where Clause: (oratt.orderdetails.quantity >= 5)
  Table Type: Not Propagate

  Child Table: ORATT.CUST_INTERESTS
  Table Type: Read Only

3 cache groups found.
```

The information displayed by the `ttIsql` utility's `cachegroups` command include:

- Cache group type, including whether the cache group is dynamic
- Autorefresh attributes (mode, state, interval) and status, if applicable
- Aging policy, if applicable
- Name of root table and, if applicable, name of child tables
- Cache table `WHERE` clause, if applicable
- Cache table attributes (read-only, propagate, not propagate)

For more information about the `ttIsql` utility's `cachegroups` command, see "ttIsql" in *Oracle TimesTen In-Memory Database Reference*.

Monitoring autorefresh operations on cache groups

TimesTen Classic offers several mechanisms to obtain information and statistics about autorefresh operations on cache groups. See "Monitoring autorefresh cache groups" in *Oracle TimesTen In-Memory Database Troubleshooting Guide*.

Monitoring AWT cache groups

TimesTen Classic offers several mechanisms to obtain information and statistics about operations in AWT cache groups. See "AWT performance monitoring" in *Oracle TimesTen In-Memory Database Troubleshooting Guide*.

Configuring a transaction log file threshold for AWT cache groups

The replication agent uses the transaction log to determine which updates on cache tables in AWT cache groups have been propagated to the cached Oracle Database tables and which updates have not. If updates are not being automatically propagated to the Oracle database because of a failure, transaction log files accumulate on the file system. Examples of a failure that prevents propagation are that the replication agent is not running or the Oracle database server is unavailable. For more information about accumulation of transaction log files, see "Monitoring accumulation of transaction log files" in *Oracle TimesTen In-Memory Database Operations Guide*.

You can call the `ttCacheAWTThresholdSet` built-in procedure as the cache manager user to set a threshold for the number of transaction log files that can accumulate before TimesTen Classic stops tracking updates on cache tables in AWT cache groups. The default threshold is 0. This built-in procedure can only be called if the TimesTen database contains AWT cache groups.

After the threshold has been exceeded, you need to manually synchronize the cache tables with the cached Oracle Database tables using an `UNLOAD CACHE GROUP` statement followed by a `LOAD CACHE GROUP` statement. TimesTen may purge transaction log files even if they contain updates that have not been propagated to the cached Oracle Database tables.

Example 6–4 Setting a transaction log file threshold for AWT cache groups

In this example, if the number of transaction log files that contain updates on cache tables in AWT cache groups exceeds 5, TimesTen stops tracking updates and can then purge transaction log files that may contain unpropagated updates:

```
% ttIsql "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> CALL ttCacheAWTThresholdSet(5);
```

You can call the `ttCacheAWTThresholdGet` built-in procedure to determine the current transaction log file threshold setting:

```
Command> CALL ttCacheAWTThresholdGet;
< 5 >
Command> exit
```

Tracking DDL statements issued on cached Oracle Database tables

When a DDL statement is issued on a cached Oracle Database table, this statement can be tracked in the Oracle Database `TT_`*`version`*`_DDL_L` table when the Oracle Database `TT_`*`version_schema-ID`*`_DDL_T` trigger is fired to insert a row into the table, where *`version`* is an internal TimesTen Classic version number and *`schema-ID`* is the ID of user that owns the cached Oracle Database table. A trigger is created for each Oracle Database user that owns cached Oracle Database tables. One DDL tracking table is created to store DDL statements issued on any cached Oracle Database table. The cache administration user owns the `TT_`*`version`*`_DDL_L` table and the `TT_`*`version_schema-ID`*`_DDL_T` trigger.

To enable tracking of DDL statements issued on cached Oracle Database tables, call the `ttCacheDDLTrackingConfig` built-in procedure as the cache manager user. By default, DDL statements are not tracked.

For more information about the `ttCacheDDLTrackingConfig` built-in procedure, see "ttCacheDDLTrackingConfig" in *Oracle TimesTen In-Memory Database Reference*.

Example 6–5 Enabling tracking of DDL statements issued on cached Oracle Database tables

```
% ttIsql "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> CALL ttCacheDDLTrackingConfig('enable');
```

The `TT_`*`version`*`_DDL_L` table and `TT_`*`version_schema-ID`*`_DDL_T` trigger are automatically created if the cache administration user has been granted the set of required privileges including `RESOURCE` and `CREATE ANY TRIGGER`. These Oracle Database objects are created when you create a cache group after tracking of DDL statements has been enabled.

If you manually created the Oracle Database objects used to manage the caching of Oracle Database data, you need to run the `ttIsql` utility's `cachesqlget` command with the `ORACLE_DDL_TRACKING` option and the `INSTALL` flag as the cache manager user. This command should be run for each Oracle Database user that owns cached Oracle Database tables that you want to track DDL statements on. Running this command generates a SQL*Plus script used to create the `TT_`*`version`*`_DDL_L` table and `TT_`*`version_schema-ID`*`_DDL_T` trigger in the Oracle database.

After generating the script, use SQL*Plus to run the script as the `sys` user.

Example 6–6 Creating DDL tracking table and trigger when Oracle Database objects were manually created

In this example, the SQL*Plus script generated by the `ttIsql` utility's `cachesqlget` command is saved to the `/tmp/trackddl.sql` file. The owner of the cached Oracle Database table `oratt` is passed as an argument to the command.

```
% ttIsql "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> cachesqlget ORACLE_DDL_TRACKING oratt INSTALL /tmp/trackddl.sql;
Command> exit

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

When you need to issue DDL statements such as `CREATE`, `DROP` or `ALTER` on cached Oracle Database tables in order to make changes to the Oracle Database schema, drop the affected cache groups before you modify the Oracle Database schema. Otherwise operations such as autorefresh may fail. You do *not* need to drop cache groups if you are altering the Oracle Database table to add a column. To issue other DDL statements for Oracle Database tables, first perform the following tasks:

1. Use `DROP CACHE GROUP` statements to drop all cache groups that cache the affected Oracle Database tables. If you are dropping an AWT cache group, use the `ttRepSubscriberWait` built-in procedure to make sure that all committed updates on the cache tables have been propagated to the cached Oracle Database tables before the cache group is dropped.

   ```
   % ttIsql "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
   Command> CALL ttRepSubscriberWait('_AWTREPSCHEME','TTREP','_ORACLE','sys1',-1);
   ```

2. Stop the cache agent.
3. Make the desired changes to the Oracle Database schema.
4. Use `CREATE CACHE GROUP` statements to re-create the cache groups, if feasible.

If you want to truncate an Oracle Database table that is cached in an autorefresh cache group, perform the following tasks:

1. Use an `ALTER CACHE GROUP` statement to set the cache group's autorefresh state to `PAUSED`.
2. Truncate the Oracle Database table.

3. Manually refresh the cache group using a `REFRESH CACHE GROUP` statement without a `WHERE` or `WITH ID` clause.

Autorefresh operations resume after you refresh the cache group.

You can run the *timesten_home*`/install/oraclescripts/cacheInfo.sql` SQL*Plus script as the cache administration user to display information about the Oracle Database objects used to track DDL statements issued on cached Oracle Database tables:

```
% cd timesten_home/install/oraclescripts
% sqlplus cacheuser/oracle
SQL> @cacheInfo
*************DDL Tracking Object Information  ***************
Common DDL Log Table Name: TT_05_DDL_L
DDL Trigger Name: TT_05_315_DDL_T
Schema for which DDL Trigger is tracking: ORATT
Number of cache groups using the DDL Trigger: 10
****************************
```

The information returned for each Oracle Database user that owns cached Oracle Database tables includes the name of the DDL tracking table, the name of its corresponding DDL trigger, the name of the user that the DDL trigger is associated with, and the number of cache groups that cache a table owned by the user associated with the DDL trigger.

If a cache group contains more than one cache table, each cache table owned by the user associated with the DDL trigger contributes to the cache group count.

Managing a caching environment with Oracle Database objects

For an autorefresh cache group, TimesTen Classic creates a change log table and trigger in the Oracle database for each cache table in the cache group. The trigger is fired for each committed insert, update, or delete operation on the cached Oracle Database table. The trigger records the primary key of the updated rows in the change log table. The cache agent periodically scans the change log table for updated keys and then joins this table with the cached Oracle Database table to get a snapshot of the latest updates.

Note: If you are caching the same Oracle table in more than one TimesTen database, see "Caching the same Oracle table on two or more TimesTen databases" on page 7-21 for performance considerations.

The Oracle Database objects used to process autorefresh writethrough operations can be automatically created by TimesTen Classic as described in "Automatically create Oracle Database objects used to manage data caching" on page 3-4 when you create a cache group with the `AUTOREFRESH MODE INCREMENTAL` cache group attribute. Alternatively, you can manually create these objects as described in "Manually create Oracle Database objects used to manage data caching" on page 3-5 before performing any cache group operation if, for security purposes, you do not want to grant the `RESOURCE` and `CREATE ANY TRIGGER` privileges to the cache administration user required to automatically create these objects.

Before the Oracle Database objects can be automatically or manually created, you must:

- Create a cache administration user in the Oracle database as described in "Create the Oracle database users" on page 3-2.
- Set the cache administration user name and password in the TimesTen database as described in "Set the cache administration user name and password" on page 3-9.
- Start the cache agent as described in "Managing the cache agent" on page 3-11.

For each cache administration user, TimesTen Classic creates the following Oracle Database tables, where `version` is an internal TimesTen Classic version number and `object-ID` is the ID of the cached Oracle Database table:

Table Name	**Description**
`TT_version_AGENT_STATUS`	Created when the first cache group is created. Stores information about each Oracle Database table cached in an autorefresh cache group.
`TT_version_AR_PARAMS`	Created when the cache administration user name and password is set. Stores the action to take when the cache administration user's tablespace is full.
`TT_version_CACHE_STATS`	Created when the cache administration user name and password is set.
`TT_version_DATABASES`	Created when the cache administration user name and password is set. Stores the autorefresh status for all TimesTen databases that cache data from the Oracle database.
`TT_version_DB_PARAMS`	Created when the cache administration user name and password is set. Stores the cache agent timeout, recovery method for dead cache groups, and the cache administration user's tablespace usage threshold.
`TT_version_DBSPECIFIC_PARAMS`	Internal use.
`TT_version_DDL_L`	Created when the cache administration user name and password is set. Tracks DDL statements issued on cached Oracle Database tables.
`TT_version_DDL_TRACKING`	Created when the cache administration user name and password is set. Stores a flag indicating whether tracking of DDL statements on cached Oracle Database tables is enabled or disabled.
`TT_version_REPACTIVESTANDBY`	Created when the first AWT cache group is created. Tracks the state and roles of TimesTen databases containing cache tables in an AWT cache group that are replicated in an active standby pair replication scheme.
`TT_version_REPPEERS`	Created when the first AWT cache group is created. Tracks the time and commit sequence number of the last update on the cache tables that was asynchronously propagated to the cached Oracle Database tables.
`TT_version_SYNC_OBJS`	Created when the first cache group is created.
`TT_version_USER_COUNT`	Created when the first cache group is created. Stores information about each cached Oracle Database table.

Table Name	Description
`TT_`*`version_object-ID`*`_L`	One change log table is created per Oracle Database table cached in an autorefresh cache group when the cache group is created. Tracks updates on the cached Oracle Database table.

For each cache administration user, TimesTen Classic creates the following Oracle Database triggers, where *`version`* is an internal TimesTen Classic version number, *`object-ID`* is the ID of the cached Oracle Database table, and *`schema-ID`* is the ID of user who owns the cached Oracle Database table:

Trigger Name	Description
`TT_`*`version`*`_REPACTIVESTANDBY_T`	Created when the first AWT cache group is created. When fired, inserts rows into the `TT_`*`version`*`_REPACTIVESTANDBY` table.
`TT_`*`version_object-ID`*`_T`	One trigger is created per Oracle Database table cached in an autorefresh cache group when the cache group is created. Fired for each insert, delete or update operation issued on the cached Oracle Database table to track operations in the `TT_`*`version_object-ID`*`_L` change log table.
`TT_`*`version_schema-ID`*`_DDL_T`	One trigger for each user who owns cached Oracle Database tables. Created when a cache group is created after tracking of DDL statements has been enabled. Fired for each DDL statement issued on a cached Oracle Database table to track operations in the `TT_`*`version`*`_DDL_L` table.

Impact of failed autorefresh operations on TimesTen databases

A change log table is created in the cache administration user's tablespace for each Oracle Database table that is cached in an autorefresh cache group. For each update operation issued on these cached Oracle Database tables, a row is inserted into their change log table to keep track of updates that need to be applied to the TimesTen cache tables upon the next incremental autorefresh cycle. TimesTen periodically deletes rows in the change log tables that have been applied to the cache tables.

An Oracle Database table cannot be cached in more than one cache group within a TimesTen database. However, an Oracle Database table can be cached in more than one TimesTen database. This results in an Oracle Database table corresponding to multiple TimesTen cache tables. If updates on cached Oracle Database tables are not being automatically refreshed into all of their corresponding cache tables because the cache agent is not running on one or more of the TimesTen databases that the Oracle Database tables are cached in, rows in their change log tables are not deleted by default. The cache agent may not be running on a particular TimesTen database because the agent was explicitly stopped or never started, the database was destroyed, or the TimesTen instance is down. As a result, rows accumulate in the change log tables and degrade the performance of autorefresh operations on cache tables in TimesTen databases where the cache agent is running. This can also cause the cache administration user's tablespace to fill up.

For example, if a single Oracle Database table is cached by two or more TimesTen databases where one of the TimesTen databases is unable to connect to the Oracle database, then autorefresh for the disconnected TimesTen database is not performed.

Instead, the records in the change log table accumulate (so that the disconnected TimesTen database can catch up once a connection to the Oracle database is established). If the `AgentTimeout` parameter is set to 0 (the default), then all change log records are kept indefinitely until they have been applied to all its cache tables. The change log records of the other TimesTen databases are not purged even though the transaction logs are already applied to the local TimesTen database. Alternatively, you can set the `AgentTimeout` parameter to define a specific timeout to wait before purging the saved change log records and stop the accumulation of these change log records.

The following criteria must be met in order for TimesTen to delete rows in the change log tables when the cache agent is not running on a TimesTen database and a cache agent timeout is set:

- Oracle Database tables are cached in autorefresh cache groups within more than one TimesTen database.
- The cache agent is running on at least one of the TimesTen databases but is not running on at least another database.
- Rows in the change log tables have been applied to the cache tables on all TimesTen databases where the cache agent is running.
- For those databases where the cache agent is not running, the agent process has been down for a period of time that exceeds the cache agent timeout.

To set the cache agent timeout and prevent rows from accumulating in the change log tables, set the `AgentTimeout` parameter with the `ttCacheConfig` built-in procedure as the cache manager user from any of the TimesTen databases that cache data from the Oracle database. Pass the `AgentTimeout` string to the *`Param`* parameter and the timeout setting as a numeric string to the *`Value`* parameter. Do not pass in any values to the *`tblOwner`* and *`tblName`* parameters as they are not applicable to setting a cache agent timeout.

Example 6–7 Setting a cache agent timeout

In the following example, the cache agent timeout is set to 900 seconds (15 minutes):

```
% ttIsql "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> CALL ttCacheConfig('AgentTimeout',,,'900');
```

To determine the current cache agent timeout setting, call `ttCacheConfig` passing only the `AgentTimeout` string to the *`Param`* parameter:

```
Command> CALL ttCacheConfig('AgentTimeout');
< AgentTimeout, <NULL>, <NULL>, 900 >
```

The default cache agent timeout setting is 0, which means that all change log records are kept indefinitely until they have been applied to all its cache tables. If you set the cache agent timeout to a value between 1 and 600 seconds, the timeout is set to 600 seconds. The cache agent timeout applies to all TimesTen databases that cache data from the same Oracle database and have the same cache administration user name setting.

When determining a proper cache agent timeout setting, consider the time it takes to load the TimesTen database into memory, the time to start the cache agent process, potential duration of network outages, and anticipated duration of planned maintenance activities.

Each TimesTen database, and all of its autorefresh cache groups have an autorefresh status to determine whether any deleted rows from the change log tables were not applied to the cache tables in the cache groups. If rows were deleted from the change log tables and not applied to some cache tables because the cache agent on the database was down for a period of time that exceeded the cache agent timeout, those cache tables are no longer synchronized with the cached Oracle Database tables. Subsequent updates on the cached Oracle Database tables are not automatically refreshed into the cache tables until the accompanying cache group is recovered.

The following are the possible statuses for an autorefresh cache group:

- `ok`: All of the deleted rows from the change log tables were applied to its cache tables. Incremental autorefresh operations continue to occur on the cache group.
- `dead`: Some of the deleted rows from the change log tables were not applied to its cache tables so the cache tables are not synchronized with the cached Oracle Database tables. Autorefresh operations have ceased on the cache group and do not resume until the cache group has been recovered.
- `recovering`: The cache group is being recovered. Once recovery completes, the cache tables are synchronized with the cached Oracle Database tables, the cache group's autorefresh status is set to `ok`, and incremental autorefresh operations resume on the cache group.

The following are the possible autorefresh statuses for a TimesTen database:

- `alive`: All of its autorefresh cache groups have an autorefresh status of OK.
- `dead`: All of its autorefresh cache groups have an autorefresh status of dead.
- `recovering`: At least one of its autorefresh cache groups have an autorefresh status of recovering.

If the cache agent on a TimesTen database is down for a period of time that exceeds the cache agent timeout, the autorefresh status of the database is set to `dead`. Also, the autorefresh status of all autorefresh cache groups within that database are set to `dead`.

Call the `ttCacheDbCgStatus` built-in procedure as the cache manager user to determine the autorefresh status of a cache group and its accompanying TimesTen database. Pass the owner of the cache group to the *`cgOwner`* parameter and the name of the cache group to the *`cgName`* parameter.

Example 6–8 Determining the autorefresh status of a cache group and TimesTen database

In the following example, the autorefresh status of the database is `alive` and the autorefresh status of the `cacheuser.customer_orders` read-only cache group is `ok`:

```
% ttIsql "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> CALL ttCacheDbCgStatus('cacheuser','customer_orders');
< alive, ok >
```

To view only the autorefresh status of the database and not of a particular cache group, call `ttCacheDbCgStatus` without any parameters:

```
Command> CALL ttCacheDbCgStatus;
< dead, <NULL> >
```

If the autorefresh status of a cache group is `ok`, its cache tables are being automatically refreshed based on its autorefresh interval. If the autorefresh status of a database is `alive`, the autorefresh status of all its autorefresh cache groups are `ok`.

If the autorefresh status of a cache group is `dead`, its cache tables are no longer being automatically refreshed when updates are committed on the cached Oracle Database tables. The cache group must be recovered in order to resynchronize the cache tables with the cached Oracle Database tables.

You can configure a recovery method for cache groups whose autorefresh status is `dead`.

Call the `ttCacheConfig` built-in procedure as the cache manager user from any of the TimesTen databases that cache data from the Oracle database. Pass the `DeadDbRecovery` string to the *`Param`* parameter and the recovery method as a string to the *`Value`* parameter. Do not pass in any values to the *`tblOwner`* and *`tblName`* parameters as they are not applicable to setting a recovery method for dead cache groups.

The following are the valid recovery methods:

- `Normal`: When the cache agent starts, a full autorefresh operation is performed on cache groups whose autorefresh status is `dead` in order to recover those cache groups. This is the default recovery method.
- `Manual`: For each explicitly loaded cache group whose autorefresh status is `dead`, a `REFRESH CACHE GROUP` statement must be issued in order to recover these cache groups after the cache agent starts.

  For each dynamic cache group whose autorefresh status is `dead`, a `REFRESH CACHE GROUP` or `UNLOAD CACHE GROUP` statement must be issued in order to recover these cache groups after the cache agent starts.
- `None`: Cache groups whose autorefresh status is `dead` must be dropped and then re-created after the cache agent starts in order to recover them.

Example 6–9 Configuring the recovery method for dead cache groups

In the following example, the recovery method is set to `Manual` for cache groups whose autorefresh status is `dead`:

```
% ttIsql "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> CALL ttCacheConfig('DeadDbRecovery',,,'Manual');
```

To determine the current recovery method for dead cache groups, call `ttCacheConfig` passing only the `DeadDbRecovery` string to the *`Param`* parameter:

```
Command> CALL ttCacheConfig('DeadDbRecovery');
< DeadDbRecovery, <NULL>, <NULL>, manual >
```

The recovery method applies to all autorefresh cache groups in all TimesTen databases that cache data from the same Oracle database and have the same cache administration user name setting.

When a cache group begins the recovery process, its autorefresh status is changed from `dead` to `recovering`, and the status of the accompanying TimesTen database is changed to `recovering`, if it is currently `dead`.

After the cache group has been recovered, its autorefresh status is changed from `recovering` to `ok`. Once all cache groups have been recovered and their autorefresh statuses are `ok`, the status of the accompanying TimesTen database is changed from `recovering` to `alive`.

A full autorefresh operation requires more system resources to process than an incremental autorefresh operation when there is a small volume of updates to refresh and a large number of rows in the cache tables. If you need to bring a TimesTen database down for maintenance activities and the volume of updates anticipated during the downtime on the Oracle Database tables that are cached in autorefresh cache groups is small, you can consider temporarily setting the cache agent timeout to 0. When the database is brought back up and the cache agent restarted, incremental autorefresh operations resumes on cache tables in autorefresh cache groups. Full autorefresh operations are avoided because the autorefresh status on the accompanying cache groups were not changed from `ok` to `dead` so those cache groups do not need to go through the recovery process. Make sure to set the cache agent timeout back to its original value once the database is back up and the cache agent has been started.

Dropping Oracle Database objects used by autorefresh cache groups

If a TimesTen database that contains autorefresh cache groups becomes unavailable, Oracle Database objects such as change log tables and triggers used to implement autorefresh operations continue to exist in the Oracle database. A TimesTen database is unavailable, for example, when the TimesTen Classic system is taken offline or the database has been destroyed without dropping its autorefresh cache groups.

Oracle Database objects used to implement autorefresh operations also continue to exist in the Oracle database when a TimesTen database is no longer being used but still contains autorefresh cache groups. Rows continue to accumulate in the change log tables. This impacts autorefresh performance on other TimesTen databases. Therefore, it is desirable to drop these Oracle Database objects associated with the unavailable or abandoned TimesTen database.

Run the `timesten_home/install/oraclescripts/cacheCleanUp.sql` SQL*Plus script as the cache administration user to drop the Oracle Database objects used to implement autorefresh operations. The host name of the TimesTen Classic system and the TimesTen database path name are passed as arguments to the `cacheCleanUp.sql` script. You can run the `cacheInfo.sql` script as the cache administration user to determine the host name of the TimesTen Classic system and the database path name. The `cacheInfo.sql` script can also be used to determine whether any objects used to implement autorefresh operations exist in the Oracle database.

Example 6–10 Dropping Oracle Database objects for autorefresh cache groups

In the following example, the TimesTen database still contained one read-only cache group `customer_orders` with cache tables `oratt.customer` and `oratt.orders` when the database was dropped. The `cacheCleanUp.sql` script drops the change log tables and triggers associated with the two cache tables.

```
% cd timesten_home/install/oraclescripts
% sqlplus cacheuser/oracle
SQL> @cacheCleanUp "sys1" "/users/OracleCache/ttcache"

*****************************OUTPUT**************************************
Performing cleanup for object_id: 69959 which belongs to table : CUSTOMER
Executing: delete from tt_05_agent_status where host = sys1 and datastore =
/users/OracleCache/ttcache and object_id = 69959
Executing: drop table tt_05_69959_L
Executing: drop trigger tt_05_69959_T
Executing: delete from tt_05_user_count where object_id = object_id1
Performing cleanup for object_id: 69966 which belongs to table : ORDERS
Executing: delete from tt_05_agent_status where host = sys1 and datastore =
/users/OracleCache/ttcache and object_id = 69966
Executing: drop table tt_05_69966_L
Executing: drop trigger tt_05_69966_T
Executing: delete from tt_05_user_count where object_id = object_id1
**************************************************************************
```

Monitoring the cache administration user's tablespace

The following sections describe how to manage the cache administration user’s tablespace:

- Defragmenting change log tables in the tablespace
- Receiving notification on tablespace usage
- Recovering from a full tablespace

Defragmenting change log tables in the tablespace

Prolonged use or a heavy workload of the change log tables for autorefresh cache groups can result in fragmentation of the tablespace. In order to prevent degradation of the tablespace from fragmentation of the change log tables, TimesTen Classic calculates the percentage of fragmentation for the change log tables as a ratio of used space to the total size of the space. If this ratio falls below a defined threshold, TimesTen alerts you of the necessity for defragmentation of the change log tables by logging a message. By default, this threshold is set to 40%. You can configure what the fragmentation threshold should be with the `ttCacheConfig` built-in procedure.

> **Note:** Messages are logged to the user and support error logs. For details, see "Error, warning, and informational messages" in the *Oracle TimesTen In-Memory Database Operations Guide*.

To set the fragmentation threshold, call the `ttCacheConfig` built-in procedure as the cache manager user from any of the TimesTen databases that cache data from the Oracle database. Pass the `AutoRefreshLogFragmentationWarningPCT` string to the *`Param`* parameter and the threshold setting as a numeric string to the *`Value`* parameter.

To set the time interval for how often to perform the calculation of the fragmentation percentage, call the `ttCacheConfig` built-in procedure as the cache manager user from any of the TimesTen databases that cache data from the Oracle database. Pass the `AutorefreshLogMonitorInterval` string to the *`Param`* parameter and the time interval (in seconds) as a numeric string to the *`Value`* parameter.

> **Note:** Do not pass in any values to the *`tblOwner`* and *`tblName`* parameters as they are not applicable to setting the fragmentation threshold or the time interval for the threshold calculation.

Example 6–11 Setting a fragmentation threshold

In the following example, the fragmentation threshold is set to 50% and the time interval for calculating the fragmentation threshold is set to 3600 seconds:

```
% ttIsql "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> CALL ttCacheConfig('AutoRefreshLogFragmentationWarningPCT',,,'50');
< AutoRefreshLogFragmentationWarningPCT, <NULL>, <NULL>, 50 >
1 row found.
Command> CALL ttCacheConfig('AutorefreshLogMonitorInterval',,,'3600');
< AutorefreshLogMonitorInterval, <NULL>, <NULL>, 3600 >
1 row found.
```

To determine the current fragmentation threshold setting, call `ttCacheConfig` passing the `AutoRefreshLogFragmentationWarningPCT` string to the *`Param`* parameter:

```
Command> CALL ttCacheConfig('AutoRefreshLogFragmentationWarningPCT');
< AutoRefreshLogFragmentationWarningPCT, <NULL>, <NULL>, 50 >
```

You can either configure TimesTen to perform defragmentation automatically or manually initiate defragmentation. To configure what action is taken when the ratio falls below the fragmentation threshold, call the `ttCacheConfig` built-in procedure with the `AutoRefreshLogDeFragmentAction` string to the *`Param`* parameter and the desired action as the *`Value`* parameter as follows:

Note: Do not pass in any values to the *`tblOwner`* and *`tblName`* parameters as they are not applicable to setting the defragmentation action.

- `Manual`. This is the default. No action is taken to defragment the change log tables. Any defragmentation must be performed manually by executing the `ttCacheAutoRefreshLogDeFrag` built-in procedure. See "Manually defragmenting the change log tables for autorefresh cache groups" on page 6-16 for more information.
- `Compact`: TimesTen defragments the change log tables.
- `CompactAndReclaim`: TimesTen defragments the change log tables and reclaims the space.

Note: When reclaiming space, the change log table is briefly locked, which temporarily suspends writing into the base table.

Example 6–12 Configuring action for fragmentation

In the following example, the action is set to `CompactAndReclaim` so that when the fragmentation ratio falls below the threshold, TimesTen defragments the change log tables and reclaims the space:

```
% ttIsql "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> CALL
ttCacheConfig('AutoRefreshLogDeFragmentAction',,,'CompactAndReclaim');
< AutoRefreshLogDeFragmentAction, <NULL>, <NULL>, compactandreclaim >
1 row found.
```

To determine the current fragmentation threshold setting, call `ttCacheConfig` passing the `AutoRefreshLogDeFragmentAction` string to the *`Param`* parameter:

```
Command> CALL ttCacheConfig('AutoRefreshLogDeFragmentAction');
< AutoRefreshLogDeFragmentAction , <NULL>, <NULL>, compactandreclaim >
```

You can discover the fragmentation percentage of the tablespace and when the last defragmentation operation was performed with the following returned columns from the `ttCacheAutorefreshStatsGet` built-in procedure:

- `AutoRefreshLogFragmentationPCT`: The current fragmentation percentage for the tablespace.
- `AutoRefreshLogFragmentationTS`: The timestamp of when the last fragmentation percentage was calculated.
- `autorefLogDeFragCnt`: The count for how many times the tables in this particular cache group have been defragmented.

For more details, see "ttCacheAutorefreshStatsGet" in the *Oracle TimesTen In-Memory Database Reference*.

Manually defragmenting the change log tables for autorefresh cache groups

To manually initiate a defragmentation of the change log tables, call the `ttCacheAutoRefreshLogDeFrag` built-in procedure as the cache manager user from any of the TimesTen databases that cache data from the Oracle database. Pass in one of the following strings as the parameter:

- `Compact`: Defragment the change log tables.
- `CompactAndReclaim`: Defragment the change log tables and reclaim the space.

> **Note:** When reclaiming space, the change log table is briefly locked, which temporarily suspends writing into the base table.

Example 6–13 Manually defragmenting the change log tables

In the following example, the user calls the `ttCacheAutoRefreshLogDeFrag` built-in procedure with the `CompactAndReclaim` option:

```
% ttIsql "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> CALL ttCacheAutoRefreshLogDeFrag('CompactAndReclaim');
```

Receiving notification on tablespace usage

In order to avoid the tablespace becoming full, you can configure TimesTen Classic to return a warning to the application when an update operation such as an `UPDATE`, `INSERT` or `DELETE` statement is issued on cached Oracle Database tables and causes the usage of the cache administration user's tablespace to exceed a specified threshold.

Call the `ttCacheConfig` built-in procedure as the cache manager user from any of the TimesTen databases that cache tables from the Oracle database. Pass the `AutoRefreshLogTblSpaceUsagePCT` string to the *`Param`* parameter and the threshold as a numeric string to the *`Value`* parameter. The threshold value represents the percentage of space used in the cache administration user's tablespace upon which a warning is returned to the application when an update operation is issued on a cached Oracle Database table. Do not pass in any values to the *`tblOwner`* and *`tblName`* parameters as they are not applicable to setting a warning threshold for the usage of the cache administration user's tablespace.

The cache administration user must be granted the `SELECT` privilege on the Oracle Database `SYS.DBA_DATA_FILES` table in order for the cache manager user to set a warning threshold on the cache administration user's tablespace usage, and for the cache administration user to monitor its tablespace to determine if the configured threshold has been exceeded.

Example 6–14 Setting a cache administration user's tablespace usage warning threshold

The following example configures a warning to be returned to the application that issues an update operation on a cached Oracle Database table if it results in the usage of the cache administration user's tablespace to exceed 80 percent:

```
% ttIsql "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> CALL ttCacheConfig('AutoRefreshLogTblSpaceUsagePCT',,,'80');
```

To determine the current cache administration user's tablespace usage warning threshold, call `ttCacheConfig` passing only the `AutoRefreshLogTblSpaceUsagePCT` string to the *`Param`* parameter:

```
Command> CALL ttCacheConfig('AutoRefreshLogTblSpaceUsagePCT');
< AutoRefreshLogTblSpaceUsagePCT, <NULL>, <NULL>, 80 >
```

The default cache administration user's tablespace usage warning threshold is 0 percent which means that no warning is returned to the application regardless of the tablespace usage. The cache administration user's tablespace usage warning threshold applies to all TimesTen databases that cache tables from the same Oracle database and have the same cache administration user name setting.

Recovering from a full tablespace

By default, when the cache administration user's tablespace is full, an error is returned to the Oracle Database application when it attempts a DML operation, such as an `UPDATE`, `INSERT` or `DELETE` statement, on a particular cached Oracle Database table.

Rather than TimesTen returning an error to the Oracle Database application when the cache administration user's tablespace is full, you can configure TimesTen to delete existing rows from the change log tables to make space for new rows when an update operation is issued on a particular cached Oracle Database table. If some of the deleted change log table rows have not been applied to the TimesTen cache tables, a full autorefresh operation is performed on those cache tables in each TimesTen database that contains the tables upon the next autorefresh cycle.

Call the `ttCacheConfig` built-in procedure as the cache manager user from any of the TimesTen databases that cache tables from the Oracle database. Pass the `TblSpaceFullRecovery` string to the *`Param`* parameter, the owner and name of the cached Oracle Database table to the *`tblOwner`* and *`tblName`* parameters, respectively, on which you want to configure an action to take if the cache administration user's tablespace becomes full, and the action itself as a string to the *`Value`* parameter.

The following are the valid actions:

- `None`: Return an Oracle Database error to the application when an update operation is issued on the cached Oracle Database table. This is the default action.
- `Reload`: Delete rows from the change log table and perform a full autorefresh operation on the cache table upon the next autorefresh cycle when an update operation is issued on the cached Oracle Database table.

Example 6–15 Configuring an action when the cache administration user's tablespace becomes full

In the following example, rows are deleted from the change log table and a full autorefresh operation is performed on the cache table upon the next autorefresh cycle when an update operation is issued on the `oratt.customer` cached Oracle Database table while the cache administration user's tablespace is full:

```
% ttIsql "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> CALL ttCacheConfig('TblSpaceFullRecovery','oratt','customer','Reload');
```

To determine the current action to take when an update operation is issued on a particular cached Oracle Database table if the cache administration user's tablespace is full, call `ttCacheConfig` passing only the `TblSpaceFullRecovery` string to the *`Param`* parameter, and the owner and name of the cached Oracle Database table to the *`tblOwner`* and *`tblName`* parameters, respectively:

```
Command> CALL ttCacheConfig('TblSpaceFullRecovery','oratt','customer');
< TblSpaceFullRecovery, ORATT, CUSTOMER, reload >
```

The action to take when update operations are issued on a cached Oracle Database table while the cache administration user's tablespace is full applies to all TimesTen databases that cache tables from the same Oracle database and have the same cache administration user name setting.

Backing up and restoring a database with cache groups

Databases containing cache groups can be backed up and restored with either the `ttBackup` or `ttMigrate` utilities.

- If the restored database connects to the same backend Oracle database, then use the `ttBackup` and `ttRestore` utilities, then drop and recreate all cache groups in the restored TimesTen database. If they are static cache groups, you may be required to reload them. For dynamic cache groups, the reload is optional as data is pulled in from the Oracle database as it is referenced.

  Note: If another TimesTen database is used to connect to the original backend Oracle database (and now no longer connects) and if all cache groups in the TimesTen database were not cleanly dropped, then execute the `cacheCleanUp.sql` SQL*Plus script against the original Oracle database to remove all leftover objects. Specify the host and path for the original TimesTen database.

- If the restored database connects to a different backend Oracle database than what it had originally connected with, then perform one of the following:
  - Backing up and restoring using the ttBackup and ttRestore utilities
  - Backing up and restoring with the ttMigrate utility

Backing up and restoring using the ttBackup and ttRestore utilities

When you use the `ttBackup` utility, it backs up the TimesTen database with all of its data at a particular time. Thus, if you want to use these cache groups again, restoring this backup requires additional action as the restored data within the cache groups are out of date and out of sync with the data in the backend Oracle database. See "Backup, Restore, and Migrate Data in TimesTen Classic" in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.

Note: See "ttBackup" and "ttRestore" in the *Oracle TimesTen In-Memory Database Reference* for more information on these tools.

If the restored database connects to a different backend Oracle database than what it had originally connected with and you want to use the `ttBackup` and `ttRestore` utilities to backup and restore your database, then perform the following:

1. Execute the `ttBackup` utility command to backup the database and its objects into a binary file. For example, to backup the `cache1` database using the `/tmp/dump` directory for temporary storage:

   ```
   % ttBackup -dir /tmp/dump -connstr "DSN=cache1"
   ```

2. Drop all cache groups and destroy the database. Since the database still exists with its cache groups, drop the cache groups and then destroy the database before restoring in the same or another location.

   ```
   % ttIsql -connstr "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
   Command> call ttCacheStop;
   Command> DROP CACHE GROUP readcache;
   Command> exit;
   Disconnecting...
   Done.

   % ttDestroy cache1
   ```

3. Restore the database with the `ttRestore` utility and then delete the temporary directory.

   ```
   % ttRestore -dir /tmp/dump -connstr "DSN=cache1"
   Restore started ...
   Restore complete

   % rm -r /tmp/dump
   ```

4. In order to re-synchronize the data within the cache groups, you must drop and recreate the cache groups:

   a. Connect to the TimesTen database.

   b. Drop the cache groups that were restored with the `ttRestore` utility. Because the data is out of sync, you may see errors.

   c. Specify the cache administrator user name and password with the `ttCacheUidPwdSet` built-in procedure.

   d. Start the cache agent.

   e. Recreate and, if required, reload the cache groups.

   ```
   % ttIsql -connstr "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"

   Command> DROP CACHE GROUP readcache;
   Command> call ttCacheUidPwdSet('cacheuser','oracle');
   Command> call ttCacheStart;
   Command> CREATE READONLY CACHE GROUP readcache
           AUTOREFRESH INTERVAL 5 SECONDS
           FROM oratt.readtab
           (keyval NUMBER NOT NULL PRIMARY KEY, str VARCHAR2(32));
   Command> LOAD CACHE GROUP readcache COMMIT EVERY 256 ROWS;
   2 cache instances affected.
   ```

   Note: If the restored TimesTen database is not able to connect to any backend Oracle database, then TimesTen cannot autorefresh the data for the read-only cache groups.

Backing up and restoring with the ttMigrate utility

The `ttMigrate` utility saves tables and indexes from a TimesTen database into a binary file. When a cache group is migrated and included in the binary file, it includes the cache group definition and schema; however, the data of the cache group is not migrated.

Note: See "Backup, Restore, and Migrate Data in TimesTen Classic" in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide* and "ttMigrate" in the *Oracle TimesTen In-Memory Database Reference* for more information on these tools.

If the restored database connects to a different backend Oracle database than what it had originally connected with and you want to use the `ttMigrate` utility for backing up and restoring the database, then perform the following:

1. Execute the `ttMigrate -c` utility command to save the database and its objects into a binary file.

   ```
   % ttMigrate -c "DSN=cache1" cache1.ttm
   ...
   Saving user CACHEUSER
   User successfully saved.

   Saving user ORATT
   User successfully saved.

   Saving table CACHEUSER.READTAB
     Saving rows...
     2/2 rows saved.
   Table successfully saved.

   Saving cache group CACHEUSER.READCACHE
     Saving cached table ORATT.READTAB
   Cache group successfully saved.
   ```

2. Drop all cache groups and destroy the TimesTen database:

   a. Stop the cache agent.

   b. Drop all cache groups. You may see errors reported, which can be ignored. When you drop all cache groups before destroying the TimesTen database, all metadata on the Oracle Database for these cache groups is deleted.

   c. Destroy the TimesTen database.

      ```
      Command> call ttCacheStop;
      Command> DROP CACHE GROUP readcache;
      Command> exit
      Disconnecting...
      Done.
      % ttDestroy cache1
      ```

3. Create and restore the database:

   a. Create the TimesTen database with a first connection request.

   b. Create the TimesTen cache table user and the TimesTen cache manager user. Grant appropriate privileges to these users.

Note: Depending on which TimesTen Classic release you are migrating from, the users and privileges may or may not be migrated. See "ttMigrate" in the *Oracle TimesTen In-Memory Database Reference* for more information.

---

c. Restore the database from the saved binary file with the `ttMigrate -r` utility command.

```
% ttIsql cache1
Command> CREATE USER cacheuser IDENTIFIED BY timesten;
 User created.

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

Command> exit
Disconnecting...
Done.

% ttMigrate -r -relaxedUpgrade -cacheuid cacheuser -cachepwd oracle
 -connstr "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
 cache1.ttm
...
Restoring table CACHEUSER.READTAB
  Restoring rows...
  2/2 rows restored.
Table successfully restored.

Restoring cache group CACHEUSER.READCACHE
  Restoring cached table ORATT.READTAB
  1/1 cached table restored.
Cache group successfully restored.
```

4. Connect to the restored database and reset the cache autorefresh state:

   a. Connect to the TimesTen database with ttIsql.

   b. Specify the cache administrator user name and password with the `ttCacheUidPwdSet` built-in procedure.

   c. Start the cache agent.

   d. Alter the cache groups to set autorefresh state to `ON`.

```
% ttIsql -connstr "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> call ttCacheUidPwdSet('cacheuser','oracle');
Command> call ttCacheStart;
Command> ALTER CACHE GROUP readcache SET AUTOREFRESH STATE ON;
```

Note: If the restored TimesTen database is not able to connect to any backend Oracle database, then TimesTen cannot autorefresh the data for the read-only cache groups.

Changing cache user names and passwords

Perform the following to change any of the user names or passwords for the TimesTen cache manager user, its companion Oracle user, or the cache administration user:

1. If you want to modify the cache manager user or password, perform the following:

Note: Passwords for both the TimesTen cache manager user and its companion Oracle user can be changed at any time.

The name for the cache manager user on TimesTen must be the same as its companion Oracle user; however, the passwords may be different. For more details on the cache manager user and its companion Oracle user, see "Create the TimesTen users" on page 3-7.

---

a. On the TimesTen database, if you want to modify the password of the cache manager user, then use the `ALTER USER` statement on the active master.

   ```
   Command> ALTER USER cacheuser IDENTIFIED BY newpwd;
   ```

b. On the back-end Oracle database, you can modify the cache manager companion Oracle password with the `ALTER USER` statement. If you are working on TimesTen, you can use `Passthrough 3` to execute this directly on the Oracle database.

   ```
   Command> passthrough 3;
   Command> ALTER USER cacheuser IDENTIFIED BY newpwd;
   ```

Note: If you have modified the password for the companion Oracle user, reconnect to the TimesTen database as the cache manager user providing passwords for the cache manager user and its companion Oracle user.

---

c. If you want to change the cache manager user, you must first drop all cache groups that the cache manager user owns before dropping the existing user and creating a new user.

Note: Alternatively, if you want to use a different user as the cache manager user, ensure that it has the correct privileges and a companion Oracle user with the correct privileges.

   In addition, since the cache manager user must have a companion Oracle user with the same name, you must either:

   - Drop all tables owned by the current companion Oracle user, drop the user, and then re-create it with the same name as the new cache manager user. If the current companion Oracle user is the cache administration user, see Step 3.
   - Choose another Oracle user that has the same name as the cache manager user and provides the same functionality.

   For full details on how to create a cache manager user and its companion Oracle user, see "Create the TimesTen users" on page 3-7.

d. If the TimesTen cache manager user name or password are defined in the `sys.odbc.ini` (or `odbc.ini`) file, update the new cache manager user name or password in the `sys.odbc.ini` (or `odbc.ini`) file on both the active and standby masters.

2. If you want to modify the cache administration user or its password, perform the following:

a. On the back-end Oracle database, you can modify the cache administration password with the `ALTER USER` statement. The password of the cache administration user can be changed at any time.

If you are working on TimesTen, you can use `Passthrough 3` to execute this directly on the Oracle database.

```
Command> passthrough 3;
Command> ALTER USER cacheuser IDENTIFIED BY newpwd;
```

b. If you want to change the cache administration user, you must first drop all cache groups on the TimesTen database that the cache administration user manages before you can drop the cache administration user on the Oracle database and create a new user. Dropping the cache groups on TimesTen removes all metadata associated with those cache groups.

When you create a new cache administration user on the Oracle database, you must follow the same instructions for creating a cache adminstration user that are provided in the "Create the Oracle database users" on page 3-2.

c. Set the new user name or password for the cache administration user by executing the `ttCacheUidPwdSet` built-in procedure on the active master database.

Note: See "Set the cache administration user name and password" on page 3-9.

```
Command> call ttCacheUidPwdSet('cacheuser','newpwd');
```

7

Cache Performance

The following sections contain information about cache performance.

> **Note:**
>
> See *Oracle TimesTen In-Memory Database Troubleshooting Guide* for extensive information about monitoring autorefresh operations and improving autorefresh performance. See "Monitoring autorefresh cache groups" and "Poor autorefresh performance".
>
> See "AWT performance monitoring" and "Possible causes of poor AWT performance" in the *Oracle TimesTen In-Memory Database Troubleshooting Guide* for more information about AWT cache group performance.

Dynamic load performance

Dynamic loading based on a primary key search of the root table has faster performance than primary key searches on a child table or foreign key searches on a child table. For more details, see "Dynamically loading a cache instance" on page 5-10.

If you combine dynamic load operations with autorefresh operations, you may experience some contention. See "Improving performance for autorefresh operations" on page 7-9 for details on how to improve your performance in this situation.

Also, there can be a performance cost when opening a new connection for a dynamic load operation. See "Managing a cache connection pool to the Oracle database for dynamic load requests" on page 7-1 for details on how to reduce the cost of opening new connections by creating a connection pool.

Managing a cache connection pool to the Oracle database for dynamic load requests

When a qualifying `SELECT` statement is issued on any dynamic read-only cache table and the data does not exist in the TimesTen cache table (but does exist in the base Oracle database table), this results in a cache miss. After which, Timesten performs a dynamic load to retrieve the data from the Oracle database (either over an existing or a new connection to the Oracle database) and inserts the rows into the cache group. There can be a performance cost when opening a new connection for the dynamic load.

By default, the connection to the Oracle database remains open until the application's connection to TimesTen is closed. When your application initiates a dynamic load, each client connection is associated with a connection to the Oracle database (when using TimesTen Cache). If you use several connections, TimesTen's requests for new connections to the Oracle database could exceed the maximum number of connections allowed to the Oracle database.

Applications can have multiple dynamic load requests to the Oracle database, which could result in too many open connections to the back-end Oracle database. However, for client/server applications with multiple connections per server, you can configure TimesTen to use the cache connection pool for all connections to the Oracle database. The cache connection pool can only be utilized by an application using a client/server connection as the pooled connections are shared across all client/server connections.

Dynamic load requests will use an existing connection to the Oracle database from the cache connection pool (rather than creating a new one) to reduce the total number of open connections. Once the dynamic load request completes, the connection is returned to the cache connection pool.

Using an existing connection from the cache connection pool increases your application performance by:

- Reducing the cost of starting a dedicated Oracle server process (or thread) for each newly requested connection.
- Reducing the total number of Oracle server processes (threads) by sharing them amongst connections rather than having each process (thread) dedicated to a single connection.
- Enabling the sharing of session level server resources, such as memory, between connections.

Once the connection is returned to the cache connection pool, the application logically sees the connection as disconnected. Thus, if your application contains passthrough statements (DDL or DML statements executed on the Oracle database), any passthrough statement must be committed or rolled back before the dynamic load is requested or an error is thrown. You can set autocommit to `ON` or explicitly execute the commit or rollback within the transaction before the dynamic load.

Note: If an application will execute a higher than normal number of dynamic load requests and performance is critical, then you might consider either:

- Removing or minimizing passthrough statements with DDL or DML statements (which can slow down your performance) from any application using the cache connection pool.
- Maintaining a completely separate connection directly to the Oracle Database to execute its SQL directly against Oracle, rather than using passthrough statements to execute SQL indirectly through TimesTen.

To decide whether to use the cache connection pool, evaluate if your applications request a high number of dynamic load requests of data from the Oracle database (resulting in too many open connections to the Oracle database).

The following sections describe how to use the cache connection pool for your dynamic read-only cache groups:

- Enable the cache connection pool
- Size the cache connection pool
- Use the ChildServer connection attribute to identify a child server process
- Apply cache connection pool sizing to currently executing database
- Example demonstrating management of the cache connection pool
- Limit the number of connections to the Oracle database
- Restrictions for the cache connection pool

Enable the cache connection pool

You can specify that TimesTen creates a cache connection pool on the TimesTen server when it starts up. All TimesTen servers share parameters that are stored on the Oracle database.

A dynamic load request from a client/server connection acquires a connection from the cache connection pool, performs the dynamic load, then returns the connection to the cache connection pool after the dynamic load request completes. The cache connection pool is destroyed when the TimesTen server shuts down.

> **Note:** The cache connection pool can only be initiated from client-server applications and is used only for dynamic loads initiated for dynamic read-only cache groups.

To enable client/server connection requests to use the cache connection pool, your application must specify the following connection attributes when connecting.

- `MaxConnsPerServer` connection attribute: Value must be set to > 1 to use the cache connection pool.

  A TimesTen server can create and assign work to multiple TimesTen child server processes (in multithreaded mode) when this connection attribute is greater than 1. This connection attribute sets the maximum number of client connections that can be created in a connection pool for each child server process.

- `ServersPerDSN` connection attribute: Value designates the number of child server processes to spawn for the TimesTen server. Default is 1.

  Each new incoming connection spawns a new child server process up to the value specified by the `ServersPerDSN` connection attribute. When the maximum number of child server processes is reached, the existing child server processes handle multiple connections (up to the number specified in `MaxConnsPerServer`) in a round-robin method. That is, if you specify `ServersPerDSN` = 2 and `MaxConnsPerServer` = 3, then the first two connections would spawn two child server processes. The third through the sixth connections would be handled by these child server processes, where each child server process would service every other connection.

Once all of the child server processes have the maximum allowed number of connections, the next incoming connection starts a new set of child server processes.

The `ServersPerDSN` and `MaxConnsPerServer` connection attributes are used to designate how to distribute connections across multiple child server processes.

- `UseCacheConnPool` connection attribute: Set the value = 2 to enable the use of the cache connection pool. See "UseCacheConnPool" in the *Oracle TimesTen In-Memory Database Reference* for more details.

> **Note:** You may also want to limit the number of connections to the Oracle database, which can be specified with the `Connections` connection attribute. See "Limit the number of connections to the Oracle database" on page 7-7 for details.

The following describes the expected behavior when a dynamic load is initiated from different types of connections:

- Direct connection: The cache connection pool is not used. The dynamic load performs using the existing behavior.
- Single threaded client/server connection (when `MaxConnsPerServer=1`): The cache connection pool is not used. The dynamic load performs using the existing behavior.
- Multithreaded client/server connection (when `MaxConnsPerServer>1`): Uses the cache connection pool for dynamic load if it is created. Otherwise, an error is returned.

You can set connection attributes for the connection either within a DSN definition or on a connect string.

Example 7–1 Specifying connection attributes for the cache connection pool in the DSN definition

The `cache1` DSN definition in the `sys.odbc.ini` file specifies `UseCacheConnPool=2`, `ServersPerDSN=2` and `MaxConnsPerServer=3`.

```
[cache1]
DataStore=/users/OracleCache/database1
PermSize=64
OracleNetServiceName=oracledb
DatabaseCharacterSet=AL32UTF8
UseCacheConnPool=2
ServersPerDSN=2
MaxConnsPerServer=3
```

Example 7–2 Specifying connection attributes for the cache connection pool on the connect string

Alternatively, you can specify both of the connection attributes on the command line when connecting from the application.

```
ttIsql "DSN=cache1; OracleNetServiceName=oracledb; UseCacheConnPool=2;
ServersPerDSN=2; MaxConnsPerServer=3"
```

Note: For more information, see the "MaxConnsPerServer", "ServersPerDSN", and "UseCacheConnPool" sections in the *Oracle TimesTen In-Memory Database Reference*.

Size the cache connection pool

You can appropriately size the cache connection pool to avoid contention for connections with the `ttCacheConnPoolSet` built-in procedure. The `ttCacheConnPoolSet` built-in procedure saves the values of these parameters on the Oracle database, which are then used as the default values when restarting the TimesTen server.

Note: If you are dynamically changing the sizing, you can apply the changes to each TimesTen server by executing the `ttCacheConnPoolApply` built-in procedure. See "Apply cache connection pool sizing to currently executing database" on page 7-6 for details on the `ttCacheConnPoolApply` built-in procedure.

Once applied to each TimesTen server, the values specified are used for the cache connection pool across all client/server applications for a TimesTen database.

You can execute the `ttCacheConnPoolSet` built-in procedure from a direct connection, a single-threaded client/server connection or a multithreaded client/server connection.

Note: For more information, see the "ttCacheConnPoolSet" section in the *Oracle TimesTen In-Memory Database Reference*.

For example, the following initiates the minimum and maximum number of pooled connections to be between 10 and 32 connections and the increment is 1. The maximum idle time by the client is set to 10 seconds. And all dynamic load operations will wait for an available connection from the cache connection pool.

```
Command> call ttCacheConnPoolSet(10, 32, 1, 10, 0);
```

Set the minimum and maximum size of the cache connection pool to levels where connections are available when needed. If no connections are available in the pool, then TimesTen performs the following depending on the setting for `ConnNoWait` parameter in the `ttCacheConnPoolSet` built-in procedure:

- `ConnNoWait=0`: TimesTen stalls until a connection from the pool is available or until a timeout occurs. If the Oracle database is down, applications wait until the Oracle database comes back up, or a timeout occurs.

  If a connection to the Oracle database times out, you receive an error denoting a loss of the connection, sometimes requiring a rollback on TimesTen.

- `ConnNoWait=1`: Any dynamic load operations fail with an error if there are no available connections in the cache connection pool.

You can query what the cache connection pool parameters are with the `ttCacheConnPoolGet` built-in procedure.

See "Example demonstrating management of the cache connection pool" on page 7-6 for a full example of how to use this built-in procedure.

Use the ChildServer connection attribute to identify a child server process

In a client/server environment, TimesTen can have multiple TimesTen child server processes to handle incoming requests from clients. You provide the `ChildServer` connection attribute to identify a specific child server process for certain cache connection pool built-in procedures. Each child server process is identified by a number assigned with the `ChildServer=`*`n`* connection attribute, where *`n`* is a number ranging from 1 to the number of running child server processes. Once connected to the child server process, you can execute either the `ttCacheConnPoolGet('current')` or `ttCacheConnPoolApply` built-in procedures that are meant for a specific child server process.

See "ttCacheConnPoolApply" and "ttCacheConnPoolGet" sections in the *Oracle TimesTen In-Memory Database Reference* for details on the built-in procedures that require this connection attribute. See "Example demonstrating management of the cache connection pool" on page 7-6 for an example of how to use this connection attribute.

Apply cache connection pool sizing to currently executing database

Since the cache connection pool parameters are saved on the Oracle database, these parameters are used to initialize the cache connection pool for the TimesTen database every time that the TimesTen server restarts.

However, if you set the cache connection parameters while the database is already running, then you can dynamically resize the cache connection pool parameters on each child server process with the `ttCacheConnPoolApply` built-in procedure. After which, the cache connection pool parameters are associated with the child server process.

For example, the following connects to the child server process identified as 1 and applies the saved cache connection pool configuration to this child server process. It does the same process for child server process 2 (given that `ServersPerDSN`=2).

```
Command> connect "DSN=cache1;ChildServer=1;";
Command> call ttCacheConnPoolApply;
Command> disconnect;

Command> connect "DSN=cache1;ChildServer=2;";
Command> call ttCacheConnPoolApply;
Command> disconnect;
```

You can execute the `ttCacheConnPoolApply` built-in procedure only from a multithreaded client/server connection.

If the cache connection pool fails, you can recreate the pool by executing the `ttCacheConnPoolApply` built-in procedure from any child server process.

See "Example demonstrating management of the cache connection pool" on page 7-6 for a full example of how to use this built-in procedure.

Example demonstrating management of the cache connection pool

Using the `cache1` DSN as shown in Example 7–1 that enables the cache connection pool and assuming that you have set the cache administrator and password as described in "Set the cache administration user name and password" on page 3-9, the following example sets new values for the cache connection pool and applies them to two separate child server processes.

```
/* Since ServerPerDSN is set to two and MaxConnsPerServer is set to 3, the first
 and second connections spawn off both child server processes. And then you can
 create four more connections to reach the MaxConnsPerServer maximum, which are
 routed by the TimesTen server to the appropriate child server process (using a
 round robin method).*/
Command> connect "DSN=cache1;" as conn1;
Command> connect "DSN=cache1;" as conn2;
Command> connect "DSN=cache1;" as conn3;
Command> connect "DSN=cache1;" as conn4;
Command> connect "DSN=cache1;" as conn5;
Command> connect "DSN=cache1;" as conn6;

Command> use conn1;

/* Query the values for the cache connection pool that are saved on the Oracle
database*/
Command> call ttCacheConnPoolGet('saved');
< 1, 10, 1, 10, 0, -1, -1, -1>

/* Change the configuration of the cache connection pool */
Command> call ttCacheConnPoolSet(1, 20, 1, 10, 0);

/* Query existing values for cache connection pool saved on the Oracle data base.
 Since these are the saved values, this returns -1 for OpenCount, BusyCount
 and LastOraErr. */
Command> call ttCacheConnPoolGet('saved');
< 1, 20, 1, 10, 0, -1, -1, -1 >

/* Query existing values for the current cache connection pool on this TimesTen
database */
Command> call ttCacheConnPoolGet('current');
< 1, 10, 1, 10, 0, 1, 0, 0 >

/* Connect to the child server process 1 using the ChildServer=1 connection
 attribute. Apply the saved values as the current values to the cache connection
 pool for child server process identified as ChildServer 1. */
Command> connect "DSN=cache1;ChildServer=1;";
Command> call ttCacheConnPoolApply;
Command> disconnect;

/* Connect to the child server process 1 using the ChildServer=1 connection
 attribute. Apply the saved values as the current values to the cache connection
 pool for child server process identified as ChildServer 2. */
Command> connect "DSN=cache1;ChildServer=2;";
Command> call ttCacheConnPoolApply;
Command> disconnect;

/* Query values for the cache connection pool in ChildServer 1 */
Command> use conn1;
Command> call ttCacheConnPoolGet('current');
< 1, 20, 1, 10, 0, 1, 0, 0 >

/* Query values for the cache connection pool in ChildServer 2 */
Command> use conn2;
Command> call ttCacheConnPoolGet('current');
< 1, 20, 1, 10, 0, 1, 0, 0 >
```

Limit the number of connections to the Oracle database

You can tune your performance while ensuring a limit to the number of connections to the Oracle database. Tuning the number of total connections depends on the following:

- **N**: The number of connections to the Oracle database.

- **P**: The limit on the number of connections for each cache connection pool, where each TimesTen child server process has a cache connection pool, which you can set with the `MaxSize` cache connection pool parameter in the `ttCacheConnPoolSet` built-in procedure.
- **S**: The number of child server processes that can be spawned for new connections. Currently, there is no direct way to limit the number of child server processes. Indirectly, you can limit the number of child server processes by setting the `MaxSize` parameter, `MaxConnsPerServer` connection attribute, and `Connections` connection attributes, as shown in the formula below.
- **M**: The maximum number of connections for each child server process, which you can set with the `MaxConnsPerServer` connection attribute.
- **D**: The maximum number of connections to a DSN, which is set with the `Connections` connection attribute.

The number of connections (**N**) to the Oracle database is equal to the number of TimesTen child server processes (**S**) times the number of connections for each cache connection pool (**M**).

```
N=S*P
```

While the maximum number of connections (**D**) to the DSN is equal to the maximum number of connections for each child server process (**M**) times the number of TimesTen child server processes (**S**).

```
D=M*S
```

With the above calculation, you can also state:

```
S=D/M
```

Since there is no hard limit that we can configure for the number of TimesTen child server processes, we merging the two equations together to eliminate **S** to get the following equation:

```
N=(D*P)/M
```

Thus, the number of connections to the Oracle database is set to the maximum number of connections to the DSN (set by the `Connections` connection attribute) times the number of connections for each cache connection pool (set by the `MaxSize` cache connection pool parameter), which is then divided by the maximum number of connections for each child server process (set by the `MaxConnsPerServer` connection attribute).

Restrictions for the cache connection pool

Restrictions when using the cache connection pool:

- You cannot use this in conjunction with the Oracle Database Resident Connection Pooling feature.
- This is only supported for multithreaded client/server connections, where the `MaxConnsPerServer` connection attribute must be greater than 1.
- This can only be used for dynamic load operations for dynamic read-only cache groups.

Improving AWT throughput

Use the following methods to improve through put for AWT cache groups:

- Improving AWT throughput with parallel propagation
- Improving AWT throughput with SQL array execution

Improving AWT throughput with parallel propagation

To improve throughput for an AWT cache group, you can configure multiple threads that act in parallel to propagate and apply transactional changes to the Oracle database. Parallel propagation enforces transactional dependencies and applies changes in AWT cache tables to Oracle Database tables in commit order. For full details, see "Configuring parallel propagation to Oracle Database tables" on page 4-15.

Improving AWT throughput with SQL array execution

The `CacheAWTMethod` connection attribute setting determines whether to use the PL/SQL execution method or SQL array execution method for asynchronous writethrough propagation when applying changes to the Oracle database.

- PL/SQL execution method: AWT bundles all pending operations into a single PL/SQL collection that is sent to the Oracle database server to be executed. This execution method is appropriate when there are mixed transactions and network latency between TimesTen and the Oracle database server. It is efficient for most use cases when the workload consists of mixed `INSERT`, `UPDATE`, and `DELETE` statements to the same or different tables. By default, TimesTen uses the PL/SQL execution method (`CacheAWTMethod=1`).
- SQL array execution method: Consider changing `CacheAWTMethod` to 0 when the changes consist of mostly repeated sequences of the same operation (`INSERT`, `UPDATE`, or `DELETE`) against the same table. For example, SQL array execution is very efficient when a user does an update that affects several rows of a table. Updates are grouped together and sent to the Oracle database in a single batch.

The PL/SQL execution method transparently falls back to SQL array execution mode temporarily when it encounters one of the following:

- A statement that is over 32761 bytes in length.
- A statement that references a column of type `BINARY FLOAT`, `BINARY DOUBLE` and `VARCHAR/VARBINARY` of length greater than 4000 bytes.

Note: You can also set this value with the `ttDBConfig` built-in procedure with the `CacheAwtMethod` parameter. For details, see "ttDBConfig" in the *Oracle TimesTen In-Memory Database Reference*.

For more information, see "CacheAWTMethod" in *Oracle TimesTen In-Memory Database Reference*.

Improving performance for autorefresh operations

The following sections describe how to improve performance for autorefresh operations:

- Minimizing delay for cached data with continuous autorefresh

- Reducing contention on TimesTen for dynamic read-only cache groups with incremental autorefresh
- Reducing lock contention for read-only cache groups that use autorefresh and dynamic load
- Improving performance when reclaiming memory during autorefresh operations
- Executing large transactions with incremental autorefresh read-only cache groups
- Configuring a select limit when using incremental autorefresh for read-only cache groups

Minimizing delay for cached data with continuous autorefresh

You can specify continuous autorefresh with an autorefresh interval of 0 milliseconds. With continuous autorefresh, the next autorefresh cycle is scheduled as soon as possible after the last autorefresh cycle has ended.

Continuous autorefresh could result in a higher resource usage when there is a low workload rate on the Oracle database, since the cache agent could be performing unnecessary round-trips to the Oracle database.

See "CREATE CACHE GROUP" and "ALTER CACHE GROUP" in the *Oracle TimesTen In-Memory Database SQL Reference* for details on how to set the autorefresh interval.

Reducing contention on TimesTen for dynamic read-only cache groups with incremental autorefresh

For most cache group operations, autorefresh and dynamic load operations coordinate their access to the Oracle database for correctness. The default TimesTen coordination behavior could result in contention between autorefresh and dynamic load operations (in extreme cases).

If you have dynamic read-only cache groups with incremental autorefresh, then:

- Multiple dynamic load operations could be blocked by autorefresh operations.
- Autorefresh operations are frequently delayed while waiting for dynamic load operations to complete.

Enabling the `DynamicLoadReduceContention` database system parameter changes the way that autorefresh and dynamic load operations coordinate, which results in reduced contention between autorefresh and dynamic load operations.

- Dynamic load operations are never blocked by autorefresh operations (due to additional synchronization).
- Autorefresh operations are not completely delayed by dynamic load operations. Instead, autorefresh operations will wait a short while for concurrently executing dynamic load operations to be notified that a new autorefresh operation is starting. This enables dynamic load operations to synchronize in tandem with concurrently executing autorefresh operations.

Note: You cannot change the value of the `DynamicLoadReduceContention` database system parameter if there are any dynamic read-only cache groups or if the cache or replication agents are running. You must unload or drop (and recreate later) any existing dynamic read only cache groups before you can change this value.

The following example sets `DynamicLoadReduceContention=1`:

```
call ttDbConfig('DynamicLoadReduceContention','1');
```

You can query the current value of the `DynamicLoadReduceContention` parameter.

```
call ttDbConfig('DynamicLoadReduceContention');
```

> **Note:** For more details, see "ttDBConfig" in the *Oracle TimesTen In-Memory Database Reference*.

Requirements for setting DynamicLoadReduceContention

The `DynamicLoadReduceContention` database system parameter is supported on TimesTen Release 11.2.2.8.39 and following. In addition, the following are requirements when enabling the `DynamicLoadReduceContention` database system parameter:

- Required Oracle Database privileges
- Unsupported Oracle Database feature
- Required settings for active standby pair replication scheme

Required Oracle Database privileges You must grant two additional Oracle Database privileges to the cache administration user:

- `EXECUTE ON SYS.DBMS_FLASHBACK`
- `SELECT ANY TRANSACTION`

These are granted to the cache administration user when you execute the `grantCacheAdminPrivileges.sql` and `initCacheAdminSchema.sql` scripts.

Unsupported Oracle Database feature This feature requires the use of the Oracle Database Flashback Transaction Queries. However, if you are using Oracle Database 12.2.0.1 with Multitenant option, Flashback Transaction Queries only supports Local Undo. It does not support Oracle Database 12.2.0.1 Multitenant option with Shared Undo.

Required settings for active standby pair replication scheme When you are using an active standby pair replication scheme:

- Both active and standby masters must be installed with TimesTen Release 11.2.2.8.39 or following. If you are replicating between and active and standby masters where each is installed with different TimesTen versions, then this parameter cannot be enabled if one of the TimesTen versions does not support this feature.
- The `DynamicLoadReduceContention` database system parameter must be set to the same value on both the active and standby masters.

Otherwise, an error is written to the daemon log. Replication will not progress until the settings and TimesTen versions conform on both the active and standby masters.

Reducing lock contention for read-only cache groups that use autorefresh and dynamic load

An autorefresh operation automatically loads committed updates on cached Oracle Database tables into the TimesTen cache tables. A dynamic load operation requests data from the Oracle database (originating from a `SELECT` statement) and inserts the rows into the cache group. Both the autorefresh and dynamic load operations require access to the TimesTen Cache metadata, which could cause a lock contention.

At the end of an autorefresh operation, TimesTen updates the metadata to track the autorefresh progress. If you have requested guaranteed durability by setting the `DurableCommits` connection attribute to 1, then the autorefresh updates to the metadata are always durably committed. If you have requested delayed durability by setting the `DurableCommits` connection attribute to 0 (the default), then TimesTen must ensure that the autorefresh updates to the metadata are durably committed before the garbage collector can clean up the autorefresh tracking tables stored on the Oracle database.

When a durable commit is initiated for the metadata, any previous non-durable committed transactions in the log buffer that have not been flushed to the file system are also a part of the durable commit. On hosts with busy or slow file systems, the durable commit could be slow enough to lock out dynamic load requests for an undesirable amount of time.

If you notice that your application is timing out because of a lock contention between autorefresh and dynamic load requests, you can set the `CacheCommitDurable` cache configuration parameter to 0 with the `ttCacheConfig` built-in procedure. This reduces the occurrence of lock contention between autorefresh and dynamic load requests in the same application by:

- Executing a non-durable commit of the autorefresh changes made to the metadata.
- Uses a separate thread in the cache agent to durably commit the autorefresh changes before the garbage collector cleans up the autorefresh tracking tables stored on the Oracle database.

By starting a new thread to perform the durable commit of the transaction log, the lock is removed after the non-durable commit of the autorefresh changes to the metadata. After which, there is no longer a lock held on the metadata and any dynamic load requests for the recently refreshed tables can continue processing without waiting.

The following example sets `CacheCommitDurable`=0:

```
call ttCacheConfig('CacheCommitDurable',,,'0');
```

You can query the current value of the `CacheCommitDurable` parameter.

```
call ttCacheConfig('CacheCommitDurable');
```

See "ttCacheConfig" in the *Oracle TimesTen In-Memory Database Reference* for more details.

Note: Since setting `CacheCommitDurable`=0 spawns a new thread to perform the durable commit of the transaction log buffer, the initiation of the garbage collection for the autorefresh change log records starts later than when `CacheCommitDurable`=1.

Improving performance when reclaiming memory during autorefresh operations

As described "Transaction reclaim operations" in the *Oracle TimesTen In-Memory Database Operations Guide*, TimesTen Classic resource cleanup occurs during the reclaim phase of a transaction commit. To improve performance, a number of transaction log records are cached in memory to reduce the need to access the transaction log file in the commit buffer. However, TimesTen must access the transaction log if the transaction is larger than the reclaim buffer.

When you are using autorefresh for your cache groups, the cache agent has its own reclaim buffer to manage the transactions that are committed within autorefresh operations. If the cache agent reclaim buffer is too small, the commit operations during autorefresh can take longer than expected as it must access the transaction log file. To avoid any performance issues, you can configure a larger reclaim buffer for the cache agent so that the cache agent can handle larger transactions in memory at reclaim time.

When using an active standby pair replication scheme to replicate autorefresh operations, the replication agent applies the same autorefresh operations as part of the replication. Thus, the replication agents on both the active and standby nodes have their own reclaim buffers that should be configured to be the same size or greater than the cache agent reclaim buffer.

The `ttDbConfig` built-in procedure provides the following parameters for setting the maximum size for the reclaim buffers for both the cache agent and the replication agent. (The memory for the reclaim buffers are allocated out of temporary memory.)

- `CacheAgentCommitBufSize` sets the maximum size for the reclaim buffer for the cache agent.
- `RepAgentCommitBufSize` sets the maximum size for the reclaim buffer for the replication agent. You should configure the maximum size for the reclaim buffer on both the active and standby nodes. It is recommended that you set the size for the reclaim buffers to the same value on both nodes, but not required.

Note: For more details, see "ttDBConfig" in the *Oracle TimesTen In-Memory Database Reference*.

To determine if you should increment the size for the cache agent reclaim buffer, evaluate the `CommitBufMaxReached` and `CommitBufNumOverflows` statistics provided by the `ttCacheAutorefIntervalStatsGet` built-in procedure. For more details, see "Retrieving statistics on autorefresh transactions" on page 7-20.

Executing large transactions with incremental autorefresh read-only cache groups

At certain times, you may execute large transactions, such as for the end of the month, the end of a quarter, or the end of the year transactions. You may also have situations where you modify or add a large amount of data in the Oracle database over a short period of time. For read-only cache groups with incremental autorefresh, TimesTen could potentially run out of permanent space when an autorefresh operation applies either one of these cases. Therefore, for these situations, you can configure an autorefresh transaction limit, where the large amount of data is broken up, applied, and committed over several smaller transactions.

Note: The autorefresh transaction limit can only be set for static read-only cache groups.

The `ttCacheAutorefreshXactLimit` built-in procedure enables you to direct autorefresh to commit after executing a specific number of operations. This option applies to all incremental autorefresh read-only cache groups that are configured with the same autorefresh interval.

Since the single transaction is broken up into several smaller transactions, transactional consistency cannot be maintained while autorefresh is in progress. Once the autorefresh cycle completes, the data is transactionally consistent. To protect instance consistency, we recommend that you set the autorefresh transaction limit only on cache groups with only a single table, since instance consistency between the parent and child tables is not guaranteed. When the autorefresh transaction limit is turned on, TimesTen does not enforce the foreign key relationship that protects instance consistency. Once you turn off the autorefresh transaction limit for incremental autorefresh read-only cache groups, both instance and transactional consistency are maintained again.

Note: If you are using an active standby pair, you must call the `ttCacheAutorefreshXactLimit` built-in procedure for the same values on both the active and standby masters.

Using ttCacheAutorefreshXactLimit

Note: For more information, such as the syntax and the returned result set, see "ttCacheAutorefreshXactLimit" in the *Oracle TimesTen In-Memory Database Replication Guide*.

For the month end processing, there can be a large number updates in a single transaction for the Oracle tables that are cached in autorefresh cache groups. In order to ensure that the large transaction does not fill up permanent memory, you can enable autorefresh to commit after every 256 (or any other user specified number) operations with the `ttCacheAutorefreshXactLimit` built-in procedure.

Turn on an autorefresh transaction limit for incremental autorefresh read-only cache groups before a large transaction with the `ttCacheAutorefreshXactLimit` built-in procedure where the *`value`* is set to `ON` or to a specific number of operations. Then, when autorefresh finishes updating the cached tables in TimesTen, turn off the autorefresh transaction limit for incremental autorefresh read-only cache groups with the `ttCacheAutorefreshXactLimit` built-in procedure.

The following example sets up the transaction limit to commit after every 256 operations for all incremental autorefresh read-only cache groups that are defined with an interval value of 10 seconds.

```
call ttCacheAutorefreshXactLimit('10000', 'ON');
```

After the month end process has completed and the incremental autorefresh read-only cache groups are refreshed, disable the transaction limit for incremental autorefresh read-only cache groups that are defined with the interval value of 10 seconds.

```
call ttCacheAutorefreshXactLimit('10000', 'OFF');
```

To enable the transaction limit for incremental autorefresh read-only cache groups to commit after every 1024 operations, provide 1024 as the value as follows:

```
call ttCacheAutorefreshXactLimit('10000', '1024');
```

Example of potential transactional inconsistency

The following example uses the employee and departments table, where the department id of the department table is a foreign key that points to the department id of the employee table.

The following example creates two incremental autorefresh read-only cache groups, where each is in its own cache group. The autorefresh transaction limit is enabled with `ttCacheAutorefreshXactLimit` before a large transaction and is disabled after it completes.

1. Before you initiate the large transaction, invoke `ttCacheAutorefreshXactLimit` to set the interval value and the number of operations after which to automatically commit. The following sets the number of operations to three (which is intentionally low to show a brief example) for all incremental autorefresh read-only cache groups with a two second interval.

   ```
   CALL ttCacheAutorefreshXactLimit('2000', '3');
   < 2000, 3 >
   1 row found.
   ```

2. Create the incremental autorefresh read-only cache groups with interval of two seconds. This example creates two static (non-dynamic) read-only cache groups, where each contains a single table.

   ```
   CREATE READONLY CACHE GROUP cgDepts AUTOREFRESH MODE INCREMENTAL
    INTERVAL 2 SECONDS
   FROM departments
       ( department_id    NUMBER(4) PRIMARY KEY
       , department_name  VARCHAR2(30) NOT NULL
       , manager_id       NUMBER(6)
       , location_id      NUMBER(4)
       );

   CREATE READONLY CACHE GROUP cgEmpls AUTOREFRESH MODE INCREMENTAL
    INTERVAL 2 SECONDS
   FROM employees
       ( employee_id    NUMBER(6) PRIMARY KEY
       , first_name     VARCHAR2(20)
       , last_name      VARCHAR2(25) NOT NULL
       , email          VARCHAR2(25) NOT NULL UNIQUE
       , phone_number   VARCHAR2(20)
       , hire_date      DATE NOT NULL
       , job_id         VARCHAR2(10) NOT NULL
       , salary         NUMBER(8,2)
       , commission_pct NUMBER(2,2)
       , manager_id     NUMBER(6)
       , department_id  NUMBER(4)
       );
   ```

3. Perform a manual `LOAD CACHE GROUP` for both autorefresh cache groups.

   ```
   LOAD CACHE GROUP cgDepts COMMIT EVERY 256 ROWS;
   27 cache instances affected.

   LOAD CACHE GROUP cgEmpls COMMIT EVERY 256 ROWS;
   107 cache instances affected.
   ```

You can have inconsistency within the table during an autorefresh as shown with the employees table.

1. On TimesTen, select the minimum and maximum salary of all employees.

   ```
   SELECT MIN(salary), MAX(salary) FROM employees;
   < 2100, 24000 >
   1 row found.
   ```

2. On the Oracle database, add 100,000 to everyone's salary.

   ```
   UPDATE employees SET salary = salary + 100000;
   ```

```
107 rows updated.
```

3. On TimesTen, when you perform the `SELECT` again (while the autorefresh transactions are commmitted after every 3 records), it shows that while the maximum salary has updated, the minimum salary is still the old value.

   ```
   SELECT MIN(salary), MAX(salary) FROM employees;
   < 2100, 124000 >
   1 row found.
   ```

4. However, once the autorefresh completes, transactional consistency is maintained. For this example, once the autorefresh process completes, all salaries have increased by 100,000.

   ```
   SELECT MIN(salary), MAX(salary) FROM employees;
   < 102100, 124000 >
   1 row found.
   ```

5. The large transaction is complete, so disable the transaction limit for autorefresh cache groups with a 2 second interval.

   ```
   call ttCacheAutorefreshXactLimit('2000', 'OFF');
   ```

You can have transactional inconsistency between cache groups if you perform a SQL statement while the autorefresh process is progressing. The following `SELECT` statement example executes against the employees and department table in the `cgDepts` autorefresh cache group. With this example, since the foreign key is not enforced on TimesTen and the autorefresh process applies several transactions, the employee table updates may be inserted before the department updates.

In addition, all of the updates for both tables in the cache group are not applied until the autorefresh cycle has completed. In the following example, the `SELECT` statement is executed before the autorefresh process is complete. Thus, the results do not show all of the expected data, such as the department name and several employees (some of the lawyers in the legal department 1000) are missing.

```
SELECT e.department_id, d.DEPARTMENT_NAME, e.FIRST_NAME, e.LAST_NAME
       FROM employees e, departments d
       WHERE e.DEPARTMENT_ID  = d.DEPARTMENT_ID (+)
       AND e.department_id >= 1000 ORDER BY 1,2,3,4;
< 1000, <NULL>, Alan, Dershowitz >
< 1000, <NULL>, F. Lee, Bailey >
< 1000, <NULL>, Johnnie, Cochran >
3 rows found.
```

However, after the autorefresh process completes, transactional consistency is maintained. The following shows the same `SELECT` statement executed after the autorefresh is complete. All expected data, the department information and all of the new lawyers, are updated.

```
SELECT e.department_id, d.DEPARTMENT_NAME, e.FIRST_NAME, e.LAST_NAME
       FROM employees e, departments d
       WHERE e.DEPARTMENT_ID  = d.DEPARTMENT_ID (+)
       AND e.department_id >= 1000 ORDER BY 1,2,3,4;
< 1000, Legal, Alan, Dershowitz >
< 1000, Legal, Barry, Scheck >
< 1000, Legal, F. Lee, Bailey >
< 1000, Legal, Johnnie, Cochran >
< 1000, Legal, Robert, Kardashian >
< 1000, Legal, Robert, Shapiro >
6 rows found.
```

For autorefresh cache groups that have more than one table, you can also experience transactional inconsistency if you execute SQL statements while the autorefresh process is in progress.

1. Initiate the transaction limit for incremental autorefresh cache groups of 2 seconds with the `ttCacheAutorefreshXactLimit` built-in procedure and create a single autorefresh cache group with two tables: the employees and departments tables.

```
CALL ttCacheAutorefreshXactLimit('2000', '3');
< 2000, 3 >
1 row found.

CREATE READONLY CACHE GROUP cgDeptEmpls AUTOREFRESH MODE INCREMENTAL
 INTERVAL 2 SECONDS
FROM departments
     ( department_id    NUMBER(4) PRIMARY KEY
     , department_name  VARCHAR2(30) NOT NULL
     , manager_id       NUMBER(6)
     , location_id      NUMBER(4)
     )
   , employees
     ( employee_id    NUMBER(6) PRIMARY KEY
     , first_name     VARCHAR2(20)
     , last_name      VARCHAR2(25) NOT NULL
     , email          VARCHAR2(25) NOT NULL UNIQUE
     , phone_number   VARCHAR2(20)
     , hire_date      DATE NOT NULL
     , job_id         VARCHAR2(10) NOT NULL
     , salary         NUMBER(8,2)
     , commission_pct NUMBER(2,2)
     , manager_id     NUMBER(6)
     , department_id  NUMBER(4)
     , foreign key(department_id) references departments(department_id)
     );
```

2. Manually load the cache group.

```
LOAD CACHE GROUP cgDeptEmpls COMMIT EVERY 256 ROWS;
27 cache instances affected.
```

3. Perform a `SELECT` statement on TimesTen that uploads all of the legal department data.

```
SELECT e.department_id, d.department_name, count(*)
       FROM employees e, departments d
       WHERE e.department_id  = d.department_id (+)
       GROUP BY e.department_id, d.department_name
       ORDER BY 1 desc;
< 110, Accounting, 2 >
< 100, Finance, 6 >
< 90, Executive, 3 >
< 80, Sales, 34 >
< 70, Public Relations, 1 >
< 60, IT, 5 >
< 50, Shipping, 45 >
< 40, Human Resources, 1 >
< 30, Purchasing, 6 >
< 20, Marketing, 2 >
< 10, Administration, 1 >
11 rows found.
```

4. On Oracle, insert a new legal department, numbered 1000, with 6 new lawyers in both the employee and department tables.

5. When performing a `SELECT` statement on TimesTen during the autorefresh process, only data on two of the lawyers in department 1000 have been uploaded into TimesTen.

```
SELECT e.department_id, d.department_name, count(*)
       FROM employees e, departments d
       WHERE e.department_id  = d.department_id (+)
       GROUP BY e.department_id, d.department_name
       ORDER BY 1 desc;
< 1000, Legal, 2 >
< 110, Accounting, 2 >
< 100, Finance, 6 >
< 90, Executive, 3 >
< 80, Sales, 34 >
< 70, Public Relations, 1 >
< 60, IT, 5 >
< 50, Shipping, 45 >
< 40, Human Resources, 1 >
< 30, Purchasing, 6 >
< 20, Marketing, 2 >
< 10, Administration, 1 >
12 rows found.
```

6. However, after the autorefresh process completes, all 6 employees (lawyers) in the legal department have been uploaded to TimesTen. Now, it is transactionally consistent.

```
SELECT e.department_id, d.department_name, COUNT(*)
       FROM employees e, departments d
       WHERE e.department_id  = d.department_id (+)
       GROUP BY e.department_id, d.department_name
       ORDER BY 1 desc;
< 1000, Legal, 6 >
< 110, Accounting, 2 >
< 100, Finance, 6 >
< 90, Executive, 3 >
< 80, Sales, 34 >
< 70, Public Relations, 1 >
< 60, IT, 5 >
< 50, Shipping, 45 >
< 40, Human Resources, 1 >
< 30, Purchasing, 6 >
< 20, Marketing, 2 >
< 10, Administration, 1 >
12 rows found.
```

7. The large transaction is complete, so disable the transaction limit for autorefresh cache groups with a 2 second interval.

```
call ttCacheAutorefreshXactLimit('2000', 'OFF');
```

Retrieving statistics to evaluate performance when a transaction limit is set

To see how a autorefresh transaction limit for a particular autorefresh interval is performing, you can retrieve statistics for the last 10 incremental autorefresh transactions for this autorefresh interval with the `ttCacheAutorefIntervalStatsGet` built-in procedure. See "Retrieving statistics on autorefresh transactions" on page 7-20 for more information.

Configuring a select limit when using incremental autorefresh for read-only cache groups

To facilitate incremental autorefresh for read-only cache groups, TimesTen executes a table join query on both the Oracle database base table and its corresponding change log table to retrieve the incremental changes. However, if both tables are very large, the join query can be slow. In addition, if the Oracle database base table is continuously updated while the join-query is executing, you may receive the `ORA-01555` "Snapshot too old" error from a long-running autorefresh query.

To avoid this situation, you can configure incremental autorefresh with a select limit for static read-only cache groups, which joins the Oracle database base table with a limited number of rows from the autorefresh change log table. You can configure a select limit with the `ttCacheAutorefreshSelectLimit` built-in procedure.

Note: The select limit can only be set for static read-only cache groups.

Autorefresh continues to apply changes to the cached table incrementally until all the rows in the autorefresh change log table have been applied. When there are no rows left to apply, the autorefresh thread sleeps for the rest of the interval period.

Note: For details on the syntax, parameters, result set, and restrictions, see "ttCacheAutorefreshSelectLimit" in the *Oracle TimesTen In-Memory Database Reference*.

For example, before a large transaction, you can call the `ttCacheAutorefreshSelectLimit` built-in procedure to set a select limit to 1000 rows for incremental autorefresh cache groups with an interval value of 10 seconds. The following example sets the *`value`* to `ON`.

```
Command> call ttCacheAutorefreshSelectLimit('10000', 'ON');
< 10000, ON >
1 row found.
```

The following example set a select limit to 2000 rows for incremental autorefresh cache groups with an interval value of 7 seconds.

```
Command> call ttCacheAutorefreshSelectLimit('7000', '2000');
< 7000, 2000 >
1 row found.
```

You can disable any select limit for incremental autorefresh cache groups with an interval value of 10 seconds by setting the *`value`* to `OFF`.

```
Command> call ttCacheAutorefreshSelectLimit('10000', 'OFF');
< 10000, OFF >
1 row found.
```

To see how a select limit for a particular autorefresh interval is performing, you can retrieve statistics for incremental autorefresh transactions for this autorefresh interval. See "Retrieving statistics on autorefresh transactions" on page 7-20 for more information.

How to determine the cache group name for a particular select limit

To determine the interval for a cache group, use `ttIsql` and run the `cachegroups` command:

```
> cachegroups cgowner.cgname;
```

This returns all attributes for the *`cgowner.cgname`* cache group including the interval.

To determine which intervals have a select limit, you can run the following query on the Oracle database where *`<cacheAdminUser>`* is the cache administrator, *`<hostName>`* is the host name of the machine where the TimesTen database is located, *`<databaseFileName>`* is the database path taken from the `DataStore` attribute, and substitute the version number (such as 06) for the *xx*.

```
SELECT * FROM <cacheAdminUser>.tt_xx_arinterval_params
 WHERE param='AutorefreshSelectEveryN'
   AND host='<hostName>'
   AND database like '%<databaseFileName>%'
 ORDER BY arinterval;
```

For example, if the cache administrator user name is `pat`, the host name is `myhost`, the database file name is `myTtDb`, and 06 is substituted for *xx* that is the TimesTen Classic minor release number then:

```
SELECT * FROM pat.tt_06_arinterval_params
 WHERE param='AutorefreshSelectEveryN'
   AND host='myhost'
   AND database like '%myTtDb%'
 ORDER BY arinterval;
```

The interval is stored in milliseconds.

Retrieving statistics to evaluate performance when using a select limit

To see how a select limit for a particular autorefresh interval is performing, you can retrieve statistics for incremental autorefresh transactions for this autorefresh interval with the `ttCacheAutorefIntervalStatsGet` built-in procedure. See "Retrieving statistics on autorefresh transactions" on page 7-20 for more information.

Retrieving statistics on autorefresh transactions

Call the `ttCacheAutorefIntervalStatsGet` built-in procedure for statistical information about the last 10 autorefresh cycles for a particular autorefresh interval defined for an incremental autorefresh read-only cache group.

Note: For more information on syntax and the returned result set for this built-in procedure, see "ttCacheAutorefIntervalStatsGet" in the *Oracle TimesTen In-Memory Database Reference*.

This built-in procedure is useful if you have set an transaction limit or a select limit for incremental, autorefresh read-only cache groups. See "Executing large transactions with incremental autorefresh read-only cache groups" on page 7-13 and "Configuring a select limit when using incremental autorefresh for read-only cache groups" on page 7-19 for details.

The following example shows how to call the `ttCacheAutorefIntervalStatsGet` built-in procedure to retrieve statistics for incremental autorefresh read-only cache groups that have been defined as static and have the interval of 2 seconds:

```
Command> call ttCacheAutorefIntervalStatsGet(2000, 1);

< 2000, 1, 21, 2013-04-30 06:05:38.000000, 100, 3761, 3761, 822, 1048576,
1280, 0, 58825, 63825, 13590, 0, 0, 0, 0, 0 >
< 2000, 1, 20, 2013-04-30 06:05:37.000000, 100, 85, 85, 18, 1048576, 1280,
0, 55064, 60064, 12768, 0, 0, 0, 0, 0 >
< 2000, 1, 19, 2013-04-30 06:05:32.000000, 100, 3043, 3043, 666, 1048576,
1280, 0, 54979, 59979, 12750, 0, 0, 0, 0, 0 >
< 2000, 1, 18, 2013-04-30 06:05:30.000000, 100, 344, 344, 74, 1048576,
1280, 0, 51936, 56936, 12084, 0, 0, 0, 0, 0 >
< 2000, 1, 17, 2013-04-30 06:05:28.000000, 100, 1826, 1826, 382, 1048576,
1280, 0, 51592, 56592, 12010, 0, 0, 0, 0, 0 >
< 2000, 1, 16, 2013-04-30 06:05:26.000000, 100, 55, 55, 12, 1048576,
1280, 0, 49766, 54766, 11628, 0, 0, 0, 0, 0 >
< 2000, 1, 15, 2013-04-30 06:05:22.000000, 100, 2901, 2901, 634, 1048576,
1280, 0, 49711, 54711, 11616, 0, 0, 0, 0, 0 >
< 2000, 1, 14, 2013-04-30 06:05:21.000000, 100, 55, 55, 12, 1048576,
1280, 0, 46810, 51810, 10982, 0, 0, 0, 0, 0 >
< 2000, 1, 13, 2013-04-30 06:05:10.000000, 100, 5844, 5844, 1263, 1048576,
1280, 0, 46755, 51755, 10970, 0, 0, 0, 0, 0 >
< 2000, 1, 12, 2013-04-30 06:05:08.000000, 100, 607, 607, 132, 1048576,
1280, 0, 40911, 45911, 9707, 0, 0, 0, 0, 0 >

10 rows found.
```

Caching the same Oracle table on two or more TimesTen databases

For each cache administration user, TimesTen creates a change log table and trigger (as part of what is created to manage caching) in the Oracle database for each cache table in the cache group. A trigger is fired for each committed insert, update, or delete operation on the cached Oracle Database table; the action is logged in the change log table.

If you cache the same Oracle database table in a cache group on two different TimesTen databases, we recommend that you use the same cache administration user name on both TimesTen databases as the owner of the cache table on each TimesTen database. When you use the same cache administration user, only one trigger and change log table are created to manage the changes to the base table. Thus, it is efficient and does not slow down the application.

If you create separate cache administration users on each TimesTen database to own the cache group that caches the same Oracle table, then separate triggers and change log tables exist on the Oracle database for the same table: one for each cache administration user. For example, if you have two separate TimesTen databases, each with their own cache administration user, two triggers fire for each DML operation on the base table, each of which are stored in a separate change log table. Firing two triggers and managing the separate change log tables can slow down the application.

The only reason to create separate cache administration users is if one of the TimesTen databases that caches the same table has a slow autorefresh rate or a slow connection to the Oracle database. In this case, having a single cache administration user on both TimesTen databases slows down the application on the faster connection, as it waits for the updates to be propagated to the slower database.

8

Cleaning up the Caching Environment

The following sections describe the various tasks that need to be performed in the TimesTen and Oracle databases to drop cache groups. It also includes a recommendation for shutting down all components when using AWT cache groups.

- Stopping the replication agent
- Dropping a cache group
- Stopping the cache agent
- Destroying the TimesTen databases
- Dropping Oracle Database users and objects
- Scheduling a shutdown of active standby pair with AWT cache groups

Stopping the replication agent

Call the `ttRepStop` built-in procedure to stop the replication agent. This must be done on each TimesTen database of the active standby pair including any read-only subscriber databases, and any standalone TimesTen databases that contain AWT cache groups.

From the `cache1`, `cache2`, `cacheactive`, `cachestandby` and `rosubscriber` databases, call the `ttRepStop` built-in procedure as the cache manager user to stop the replication agent on the database:

```
Command> CALL ttRepStop;
```

Dropping a cache group

Use the `DROP CACHE GROUP` statement to drop a cache group and its cache tables. Oracle Database objects used to manage the caching of Oracle Database data are automatically dropped when you use the `DROP CACHE GROUP` statement to drop a cache group, or an `ALTER CACHE GROUP` statement to set the autorefresh state to `OFF` for autorefresh cache groups.

If you issue a `DROP CACHE GROUP` statement on a cache group that has an autorefresh operation in progress:

- The autorefresh operation stops if the `LockWait` connection attribute setting is greater than 0. The `DROP CACHE GROUP` statement preempts the autorefresh operation.
- The autorefresh operation continues if the `LockWait` connection attribute setting is 0. The `DROP CACHE GROUP` statement is blocked until the autorefresh operation completes or the statement fails with a lock timeout error.

If cache tables are being replicated in an active standby pair and the cache tables are the only elements that are being replicated, you must drop the active standby pair using a `DROP ACTIVE STANDBY PAIR` statement before dropping the cache groups.

Execute the following statement as the cache manager user on the `cacheactive`, `cachestandby` and `rosubscriber` databases to drop the active standby pair replication scheme:

```
Command> DROP ACTIVE STANDBY PAIR;
Command> exit
```

Before you can drop a cache group, you must grant the `DROP ANY TABLE` privilege to the cache manager user. Execute the following statement as the instance administrator on the `cache1`, `cache2`, `cacheactive` and `cachestandby` databases to grant the `DROP ANY TABLE` privilege to the cache manager user. The following example shows the SQL statement issued from the `cache1` database:

```
% ttIsql cache1
Command> GRANT DROP ANY TABLE TO cacheuser;
Command> exit
```

If you are dropping an AWT cache group, use the `ttRepSubscriberWait` built-in procedure to make sure that all committed updates on its cache tables have been propagated to the cached Oracle Database tables before dropping the cache group.

```
% ttIsql "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> CALL ttRepSubscriberWait('_AWTREPSCHEME','TTREP','_ORACLE','sys1',-1);
```

The replication scheme that was created for the AWT cache group to enable committed updates on its cache tables to be asynchronously propagated to the cached Oracle tables is automatically dropped when you drop the cache group.

Use a `DROP CACHE GROUP` statement to drop the cache groups from the standalone TimesTen databases and the active and standby databases.

Execute the following statement as the cache manager user on the `cache1`, `cache2`, `cacheactive` and `cachestandby` databases to drop the `subscriber_accounts` cache group. The following example shows the SQL statement issued from the `cache1` database:

```
% ttIsql "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
Command> DROP CACHE GROUP subscriber_accounts;
```

Note: If the cache agent is stopped immediately after dropping a cache group, or altering the cache group's autorefresh state to `OFF`, the Oracle Database objects used to manage the caching of Oracle Database data *may* not have been dropped. When the cache agent is restarted, it drops the Oracle Database objects that were created for the dropped or altered cache group.

Stopping the cache agent

Call the `ttCacheStop` built-in procedure to stop the cache agent. This must be done on the active and standby databases of the active standby pair, and all standalone TimesTen databases.

From the `cache1`, `cache2`, `cacheactive` and `cachestandby` databases, issue the following built-in procedure call to stop the cache agent on the database:

```
Command> CALL ttCacheStop;
Command> exit
```

Destroying the TimesTen databases

If the TimesTen databases are no longer needed, you can use the `ttDestroy` utility to destroy the databases.

Note: If the RAM policy designates that the database stays in memory, then this may prevent you from destroying the database. For example, if the RAM policy is set to `always`, then you must change the RAM policy to `manual` and run the `ttAdmin -ramunload` command to unload the database before destroying the database. For details on the RAM policy settings, see "Specifying a RAM policy" section in the *Oracle TimesTen In-Memory Database Operations Guide*.

The following example shows the `ttDestroy` utility connecting to and then destroying the `cache1` database:

```
% ttDestroy cache1
```

Dropping Oracle Database users and objects

Use SQL*Plus as the `sys` user to drop the `timesten` user, the schema user `oratt`, and the cache administration user `cacheuser`, and all objects such as tables and triggers owned by these users. Then drop the `TT_CACHE_ADMIN_ROLE` role, and the default tablespace `cachetblsp` used by the timesten user and the cache administration user including the contents of the tablespace and its data file.

```
% sqlplus sys as sysdba
Enter password: password
SQL> DROP USER timesten CASCADE;
SQL> DROP USER oratt CASCADE;
SQL> DROP USER cacheuser CASCADE;
SQL> DROP ROLE tt_cache_admin_role;
SQL> DROP TABLESPACE cachetblsp INCLUDING CONTENTS AND DATAFILES;
SQL> exit
```

Scheduling a shutdown of active standby pair with AWT cache groups

When you are using active standby pairs with AWT cache groups, the environment includes both an active and a standby master, potentially one or more subscribers, and at least one Oracle Database. The following is the recommended method when you initiate a scheduled shutdown of outstanding transactions in this environment. This order of events provides the time needed to finish applying outstanding transactions before shut down and minimizes the time needed to restart all components.

1. Shut down all applications.
2. Ensure that all transactions have propagated to the Oracle database.
3. Shut down TimesTen.
4. Shut down the Oracle Database.

Then, when you are ready to restart all components:

1. Restart the Oracle Database.

2. Restart TimesTen.
3. Restart any applications.

You can shut down all of these products in any order without error. The order matters only to maximize performance and reduce the need for preserving unapplied transactions. For example, when you are using AWT cache groups within the active standby pair and if you shut down the Oracle database before TimesTen, then all unapplied transactions accumulate in the TimesTen transaction logs. Thus, when you restart TimesTen and Oracle, you could potentially have a lower throughput while pending transactions are applied to the Oracle database. Thus, shutting down TimesTen before the Oracle database provides the most efficient method for your scheduled shutdown and startup. In addition, shutting down the applications before TimesTen stops any additional requests from being sent to an unavailable TimesTen database.

9

Using TimesTen Cache in an Oracle RAC Environment

The following sections describe how to use TimesTen Cache in an Oracle Real Application Clusters (Oracle RAC) environment:

- How TimesTen Cache works in an Oracle RAC environment
- Restrictions on using TimesTen Cache in an Oracle RAC environment
- Setting up TimesTen Cache in an Oracle RAC environment

How TimesTen Cache works in an Oracle RAC environment

Oracle RAC enables multiple Oracle Database 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 file systems. Oracle RAC handles read/write consistency and load balancing while providing high availability.

Fast Application Notification (FAN) is an Oracle RAC feature that is integrated with Oracle Call Interface (OCI) in Oracle Database. FAN publishes information about changes in the cluster to applications that subscribe to FAN events. FAN prevents unnecessary operations such as the following:

- Attempts to connect when services are down
- Attempts to finish processing a transaction when the server is down
- Waiting for TCP/IP timeouts

See *Oracle Real Application Clusters Administration and Deployment Guide* for more information about Oracle RAC and FAN.

TimesTen Cache uses OCI integrated with FAN to receive notification of Oracle Database events. With FAN, TimesTen Cache detects connection failures within a minute. Without FAN, it can take several minutes for TimesTen Cache to receive notification of an Oracle Database failure. Without FAN, TimesTen Cache detects a connection failure the next time the connection is used or when a TCP/IP timeout occurs. TimesTen Cache can recover quickly from Oracle Database failures without user intervention.

TimesTen Cache also uses Transparent Application Failover (TAF), which is a feature of Oracle Net Services that enables you to specify how you want applications to reconnect after a failure. See *Oracle Database Net Services Administrator's Guide* for more information about TAF. TAF attempts to reconnect to the Oracle database for four minutes. If this is not successful, the cache agent restarts and attempts to reconnect with the Oracle database every minute.

Note: You can configure how long TAF retries when establishing a connection with the `AgentFailoverTimeout` parameter. For details, see "Setting up TimesTen Cache in an Oracle RAC environment" on page 9-4.

OCI applications can use one of the following types of Oracle Net failover functionality:

- `None`: No failover functionality is used. This can also be explicitly specified to prevent failover from happening. This is the default failover functionality.
- `Session`: If an application's connection is lost, a new connection is automatically created for the application. This type of failover does not attempt to recover selects.
- `Select`: This type of failover enables applications that began fetching rows from a cursor before failover to continue fetching rows after failover.

The behavior of TimesTen Cache depends on the actions of TAF and how TAF is configured. By default, TAF and FAN callbacks are installed if you are using TimesTen Cache in an Oracle RAC environment. If you do not want TAF and FAN capabilities, set the `RACCallback` connection attribute to 0.

Table 9–1 shows the behaviors of TimesTen Cache operations in an Oracle RAC environment with different TAF failover types.

Table 9–1 Behavior of TimesTen Cache operations in an Oracle RAC environment

Operation	TAF Failover Type	Behavior After a Failed Connection on the Oracle Database
Autorefresh	None	The cache agent automatically stops, restarts and waits until a connection can be established on the Oracle database. This behavior is the same as in a non-Oracle RAC environment. No user intervention is needed.
Autorefresh	`Session`	One of the following occurs: ▪ All failed connections are recovered. Autorefresh operations that were in progress are rolled back and retried. ▪ If TAF times out or cannot recover the connection, the cache agent automatically stops, restarts and waits until a connection can be established on the Oracle database. ▪ In all cases, no user intervention is needed.
Autorefresh	`Select`	One of the following occurs: ▪ Autorefresh operations resume from the point of connection failure. ▪ Autorefresh operations that were in progress are rolled back and retried. ▪ If TAF times out or cannot recover the connection, the cache agent automatically stops, restarts and waits until a connection can be established on the Oracle database. ▪ In all cases, no user intervention is needed.

Table 9–1 (Cont.) Behavior of TimesTen Cache operations in an Oracle RAC

Operation	**TAF Failover Type**	**Behavior After a Failed Connection on the Oracle Database**
AWT	None	The receiver thread of the replication agent for the AWT cache group exits. A new thread is spawned and tries to connect to the Oracle database. No user intervention is needed.
AWT	`Session`, `Select`	One of the following occurs: ▪ If the connection is recovered and there are uncommitted DML operations in the transaction, the transaction is rolled back and then reissued. ▪ If the connection is recovered and there are no uncommitted DML operations, new operations can be issued without rolling back. In all cases, no user intervention is needed.
SWT, propagate, flush, and passthrough	None	The application is notified of the connection loss. The cache agent disconnects from the Oracle database and the current transaction is rolled back. All modified session attributes are lost. During the next passthrough operation, the cache agent tries to reconnect to the Oracle database. This behavior is the same as in a non-Oracle RAC environment. No user intervention is needed.
SWT, propagate, flush and passthrough	`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, an error is returned and the user must roll back the transaction before continuing. Otherwise, the user can continue without rolling back. ▪ If TAF times out or cannot recover the connection, the application is notified of the connection loss. The cache agent disconnects from the Oracle database and the current transaction is rolled back. All modified session attributes are lost. During the next passthrough operation, the cache agent tries to reconnect to the Oracle database. In this case, no user intervention is needed.
SWT, propagate and flush	`Select`	
Passthrough	`Select`	The connection to the Oracle database is recovered. If there were DML or lock operations on the lost connection, an error is returned and the user must roll back the transaction before continuing. Otherwise, the user can continue without rolling back.
Load and refresh	None	The application receives a loss of connection error.

Table 9–1 (Cont.) Behavior of TimesTen Cache operations in an Oracle RAC

Operation	**TAF Failover Type**	**Behavior After a Failed Connection on the Oracle Database**
Load and refresh	`Session`	One of the following occurs: - The load or refresh operation succeeds. - An error is returned stating that a fetch operation on Oracle Database cannot be executed.
Load and refresh	`Select`	One of the following occurs: - If the Oracle Database cursor is open and the cursor is recovered, or if the Oracle Database cursor is not open, then the load or refresh operation succeeds. - An error is returned if TAF was unable to recover either the session or open Oracle Database cursors. **Note**: An error is less likely to be returned than if the TAF failover type is Session.

Restrictions on using TimesTen Cache in an Oracle RAC environment

TimesTen Cache support of Oracle RAC has the following restrictions:

- TimesTen Cache behavior is limited to Oracle RAC, FAN and TAF capabilities. For example, if all nodes for a service fail, the service is not restarted. TimesTen Cache waits for the user to restart the service.
- TAF does not recover `ALTER SESSION` operations. The user is responsible for restoring changed session attributes after a failover.
- TimesTen Cache uses OCI integrated with FAN. This interface automatically spawns a thread to wait for an Oracle Database event. This is the only TimesTen Cache feature that spawns a thread in a TimesTen Classic application with the direct driver. Adapt your application to account for this thread creation. If you do not want the extra thread, set the `RACCallback` connection attribute to 0 so that TAF and FAN are not used.

Setting up TimesTen Cache in an Oracle RAC environment

After you install Oracle RAC and TimesTen Cache, perform the following to set up an TimesTen Cache for an Oracle RAC environment:

1. On TimesTen Classic, set the TAF timeout, in minutes, with the `ttCacheConfig AgentFailoverTimeout` parameter. The `AgentFailoverTimeout` parameter configures how long TAF retries when establishing a connection. TAF attempts to reconnect to the Oracle database for the duration of this timeout. The default is four minutes. If this is not successful, the cache agent restarts and attempts to reconnect with the Oracle database every minute; the replication agent restarts any threads that cannot connect to the Oracle database. For more details, see "ttCacheConfig" in the *Oracle TimesTen In-Memory Database Reference*.
2. Make sure that the TimesTen daemon, the cache agent, and the following Oracle Database components are started:
    - Oracle Database instances
    - Oracle Database listeners

- Oracle Database service that is used for TimesTen Cache

3. Verify that the TimesTen `RACCallback` connection attribute is set to 1 (default). For more details, see "RACCallback" in the *Oracle TimesTen In-Memory Database Reference*.
4. Use the `DBMS_SERVICE.MODIFY_SERVICE` function or Oracle Enterprise Manager to enable publishing of FAN events. This changes the value in the `AQ_HA_NOTIFICATIONS` column of the Oracle Database `ALL_SERVICES` view to `YES`.

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

     See *Oracle Database Net Services Administrator's Guide* for more information about enabling TAF.
6. If you have a TimesTen Classic application that uses the direct driver, link it with a thread library so that it receives FAN notifications. FAN spawns a thread to monitor for failures.

10

Using TimesTen Cache with Data Guard

This chapter describes how to configure TimesTen Cache to work with either synchronous or asynchronous Data Guard. It includes the following topics:

- Components of MAA for TimesTen Cache
- TimesTen Cache works with asynchronous Active Data Guard
- TimesTen Cache works with synchronous Data Guard

Components of MAA for TimesTen Cache

Oracle Maximum Availability Architecture (MAA) is Oracle Database's best practices blueprint based on proven Oracle Database high availability (HA) technologies and recommendations. The goal of MAA is to achieve the optimal high availability architecture at the lowest cost and complexity.

To be compliant with MAA, TimesTen Cache must support Oracle Real Application Clusters (Oracle RAC) and Oracle Data Guard, as well as have its own HA capability. TimesTen Cache provides its own HA capability through active standby pair replication of cache tables in read-only and AWT cache groups. See "Using TimesTen Cache in an Oracle RAC Environment" on page 9-1 for more information on TimesTen Cache and Oracle RAC.

Oracle Data Guard provides the management, monitoring, and automation software infrastructure to create and maintain one or more synchronized standby Oracle databases to protect data from failures, disasters, errors, and corruptions. If the primary Oracle database becomes unavailable because of a planned or an unplanned outage, Data Guard can switch any standby Oracle database to the primary role, thus minimizing downtime and preventing any data loss. For more information about Data Guard, see *Oracle Data Guard Concepts and Administration*.

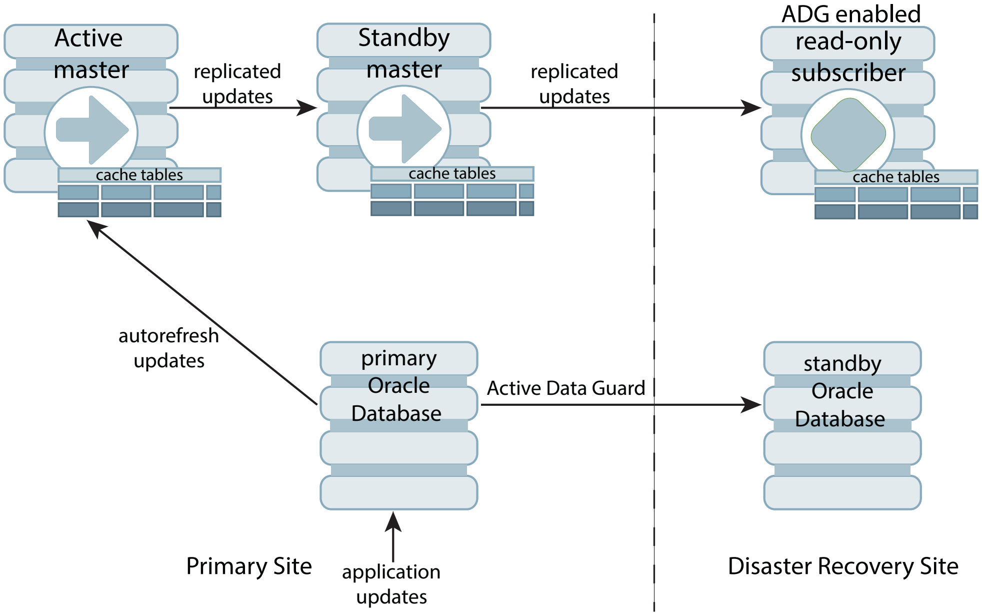
The MAA framework for TimesTen Cache supports cache tables in explicitly loaded read-only and AWT cache groups. For cache tables in dynamic cache groups of any cache group type, SWT cache groups, and user managed cache groups that use the `AUTOREFRESH` cache group attribute, TimesTen Cache cannot access the Oracle database during a failover and switchover because cache applications wait until the failover and switchover completes.

In general, however, all cache groups types are supported with synchronous Data Guard or Data Guard during planned maintenance.

TimesTen Cache works with asynchronous Active Data Guard

You can cache tables from an Oracle Active Data Guard with the asynchronous redo transport mode into read-only cache groups. The read-only cache groups are replicated within an active standby pair replication scheme. The Active Data Guard configuration includes a primary Oracle database that communicates over an asynchronous transport to a single physical standby Oracle database. As shown in Figure 10–1, the primary Oracle database is located on the primary site, while the standby Oracle database is located on a disaster recovery site.

Figure 10–1 Recommended configuration for asynchronous Active Data Guard



On TimesTen, the read-only cache groups on the primary site are autorefreshed from the primary Oracle database; however, the only transactions that are autorefreshed are those whose changes have been successfully replicated to the standby Oracle database. Once refreshed to the active master, all changes are then propagated to the TimesTen standby master and a read-only subscriber using normal TimesTen replication processes.

For the best failover and recovery action, you should locate the read-only subscriber on the same disaster recovery site as the standby Oracle database. Create this read-only subscriber with the `ttRepAdmin -duplicate -activeDataGuard` utility option, which replicates the read-only cache groups directly to the subscriber as it would to a standby master database. That is, instead of the cache groups being converted to tables when replicated to a subscriber, the cache groups themselves are replicated to the read-only subscriber. This is to provide a recovery and failover option if the primary site fails. For more details, see "Recovery after failure when using asynchronous Active Data Guard" on page 10-8.

The following sections provide more details on the environment for asynchronous Active Data Guard when using replicated read-only cache groups:

- Configuring the primary and standby Oracle databases
- Configuring the active standby pair with read-only cache groups
- Recovery after failure when using asynchronous Active Data Guard

Configuring the primary and standby Oracle databases

When you create and configure Active Data Guard with primary and standby Oracle databases, ensure that the configuration includes the following to support the TimesTen cache environment.

1. Configure both the primary and standby Oracle databases to use Flashback queries. For more information, see "Configuring Recovery Settings" in the *Oracle Database 2 Day DBA guide*.
2. The Data Guard configuration must be managed by the Data Guard Broker so that the TimesTen Cache daemon processes and application clients respond faster to failover and switchover events. For more information, see the *Data Guard Broker* guide.
3. Create two Oracle Database Services, where one points to the primary Oracle Database and the other points to the physical standby Oracle Database. See "Creating two Oracle Database services" on page 10-3 for details.

Creating two Oracle Database services

Create supporting database services on both the primary and standby Oracle databases in the Oracle Cluster, where one points to the primary Oracle Database and the other points to the physical standby Oracle Database. You can create these either through role based services or through system triggers.

- Configuring Oracle Database services through role based services
- Configuring Oracle Database services through system triggers

Configuring Oracle Database services through role based services You can automatically control the startup of Oracle database services on both the primary and standby Oracle databases by assigning a database role to each service. An Oracle database service automatically starts when the Oracle database starts if the Oracle database policy is set to `AUTOMATIC` and if the service role matches the current role of the database. In this case, the role for the Oracle database is either in the primary or standby role as part of the Active Data Guard configuration.

Configure services with the `srvctl` utility identically on all Oracle databases in the Data Guard configuration. The following example shows two services created identically on both the primary and the standby Oracle databases. For more information on the `srvctl` utility, see the "srvctl add service" section in the *Oracle Database Administrator's Guide*.

The following steps add the `primaryrole` and `standbyrole` database services to both the primary and standby Oracle databases when the primary Oracle database is located in Austin and the standby Oracle database is located in Houston.

1. On the primary Oracle database, add the `primaryrole` database service. While this Oracle database acts as the primary, this service is started.

   ```
   srvctl add service -d Austin -s primaryrole -r ssa1,ssa2,ssa3,
    ssa4 -l PRIMARY -q TRUE -e SESSION -m BASIC -w 10 -z 150
   ```

2. On the primary Oracle database, add the `standbyrole` database service. This service starts only if this Oracle database switches to the standby role and then provides real-time reporting on the standby Oracle database.

   ```
   srvctl add service -d Austin -s standbyrole -r ssa1,ssa2,ssa3,
    ssa4 -l PHYSICAL_STANDBY -q TRUE -e SESSION -m BASIC -w 10 -z 150
   ```

3. On the standby Oracle database, add the `primaryrole` database service. This service starts only if this Oracle database switches to the primary role.

   ```
   srvctl add service -d Houston -s primaryrole -r ssb1,ssb2,ssb3,
    ssb4 -l PRIMARY -q TRUE -e SESSION -m BASIC -w 10 -z 150
   ```

4. On the standby Oracle database, add the `standbyrole` database service. While this Oracle database acts as the standby, this service is started and then provides real-time reporting on the standby Oracle database.

   ```
   srvctl add service -d Houston -s standbyrole -r ssb1,ssb2,ssb3,
    ssb4 -l PHYSICAL_STANDBY -q TRUE -e SESSION -m BASIC -w 10 -z 150
   ```

5. Execute the following SQL statement on the primary Oracle database so that the service definitions are transmitted and applied to the physical standby Oracle database.

   ```
   EXECUTE DBMS_SERVICE.CREATE_SERVICE('standbyrole', 'standbyrole', NULL,
    NULL, TRUE, 'BASIC', 'SESSION', 150, 10, NULL);
   ```

6. Add connection aliases in the appropriate `tnsnames.ora` files to identify the primary and standby Oracle databases and specify the database service names for each.

   ```
   primaryinstance=
     (DESCRIPTION_LIST=
       (LOAD_BALANCE=off)
       (FAILOVER=on)
       (DESCRIPTION=(ADDRESS_LIST=(LOAD_BALANCE=on)
              (ADDRESS=(PROTOCOL=TCP)(HOST=myhost1)(PORT=1521)))
                  (CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=primaryrole)))

       (DESCRIPTION=(ADDRESS_LIST=(LOAD_BALANCE=on)
              (ADDRESS=(PROTOCOL=TCP)(HOST=myhost2)(PORT=1521)))
                  (CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=primaryrole))))

   standbyinstance=
      (DESCRIPTION_LIST=
        (LOAD_BALANCE=off)
        (FAILOVER=on)
        (DESCRIPTION=(ADDRESS_LIST=(LOAD_BALANCE=on)
               (ADDRESS=(PROTOCOL=TCP)(HOST=myhost1)(PORT=1521)))
                   (CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=standbyrole)))

        (DESCRIPTION=(ADDRESS_LIST=(LOAD_BALANCE=on)
               (ADDRESS=(PROTOCOL=TCP)(HOST=myhost2)(PORT=1521)))
                  (CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=standbyrole))))
   ```

7. On the primary Oracle database, start the `primaryrole` database service.

   ```
   srvctl start service -d Austin -s primaryrole
   ```

8. On the standby Oracle database, start the `standbyrole` database service.

   ```
   srvctl start service -d Houston -s standbyrole
   ```

Configuring Oracle Database services through system triggers Perform the following steps to create the `primaryrole` and `standbyrole` database services on the primary Oracle database using triggers. After creation, these are replicated to the standby Oracle database.

1. Create the `primaryrole` and `standbyrole` database services in the primary Oracle database.

   ```
   exec DBMS_SERVICE.CREATE_SERVICE(
    service_name => 'primaryrole',
    network_name => 'primaryrole',
    aq_ha_notifications => true, failover_method => 'BASIC',
    failover_type => 'SELECT', failover_retries => 180, failover_delay => 1 );

   exec DBMS_SERVICE.CREATE_SERVICE(
    service_name => 'standbyrole',
    network_name => 'standbyrole',
    aq_ha_notifications => true, failover_method => 'BASIC',
    failover_type => 'SELECT', failover_retries => 180, failover_delay => 1 );
   ```

2. Create the `primaryrole` and `standbyrole` triggers in the primary Oracle database for when the database starts.

   ```
   CREATE OR REPLACE TRIGGER manage_OCIService
   after startup on database
   DECLARE
     role VARCHAR(30);
   BEGIN
     SELECT DATABASE_ROLE INTO role FROM V$DATABASE;
     IF role = 'PRIMARY' THEN
       BEGIN
         DBMS_SERVICE.START_SERVICE('primaryrole');
       EXCEPTION
         WHEN OTHERS THEN
           NULL;
       END;
       BEGIN
         DBMS_SERVICE.STOP_SERVICE('standbyrole');
       EXCEPTION
         WHEN OTHERS THEN
           NULL;
       END;
     ELSE
       BEGIN
         DBMS_SERVICE.STOP_SERVICE('primaryrole');
       EXCEPTION
         WHEN OTHERS THEN
           NULL;
       END;
       BEGIN
         DBMS_SERVICE.START_SERVICE('standbyrole');
       EXCEPTION
         WHEN OTHERS THEN
           NULL;
       END;
     END IF;
   END;
   ```

3. Create the following trigger on the primary Oracle database to execute when the database changes roles:

   ```
   CREATE OR REPLACE TRIGGER manage_OCIService2
   AFTER DB_ROLE_CHANGE ON DATABASE
   DECLARE
     role VARCHAR(30);
   BEGIN
   ```

```
  SELECT DATABASE_ROLE INTO role FROM V$DATABASE;
  IF role = 'PRIMARY' THEN
    BEGIN
      DBMS_SERVICE.START_SERVICE('primaryrole');
    EXCEPTION
      WHEN OTHERS THEN
        NULL;
    END;
    BEGIN
      DBMS_SERVICE.STOP_SERVICE('standbyrole');
    EXCEPTION
      WHEN OTHERS THEN
        NULL;
    END;
  ELSE
    BEGIN
      DBMS_SERVICE.STOP_SERVICE('primaryrole');
    EXCEPTION
      WHEN OTHERS THEN
        NULL;
    END;
    BEGIN
      DBMS_SERVICE.START_SERVICE('standbyrole');
    EXCEPTION
      WHEN OTHERS THEN
        NULL;
    END;
  END IF;
END;
```

4. Add connection aliases in the appropriate `tnsnames.ora` files to identify the primary and standby Oracle databases and specify the database service names for each.

```
primaryinstance=
  (DESCRIPTION_LIST=
    (LOAD_BALANCE=off)
    (FAILOVER=on)
    (DESCRIPTION=(ADDRESS_LIST=(LOAD_BALANCE=on)
           (ADDRESS=(PROTOCOL=TCP)(HOST=myhost1)(PORT=1521)))
               (CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=primaryrole)))

    (DESCRIPTION=(ADDRESS_LIST=(LOAD_BALANCE=on)
           (ADDRESS=(PROTOCOL=TCP)(HOST=myhost2)(PORT=1521)))
               (CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=primaryrole))))

standbyinstance=
   (DESCRIPTION_LIST=
     (LOAD_BALANCE=off)
     (FAILOVER=on)
     (DESCRIPTION=(ADDRESS_LIST=(LOAD_BALANCE=on)
            (ADDRESS=(PROTOCOL=TCP)(HOST=myhost1)(PORT=1521)))
                (CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=standbyrole)))

     (DESCRIPTION=(ADDRESS_LIST=(LOAD_BALANCE=on)
            (ADDRESS=(PROTOCOL=TCP)(HOST=myhost2)(PORT=1521)))
               (CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=standbyrole))))
```

5. Restart both of the Oracle databases to enable the trigger to start and stop the correct database services. Alternatively, if you do not want to restart both Oracle

databases, you can start and stop the appropriate database services on each Oracle database as follows:

On the primary Oracle database:

```
exec DBMS_SERVICE.START_SERVICE('primaryrole');
exec DBMS_SERVICE.STOP_SERVICE('standbyrole');
```

On the standby Oracle database:

```
exec DBMS_SERVICE.STOP_SERVICE('primaryrole');
exec DBMS_SERVICE.START_SERVICE('standbyrole');
```

Configuring the active standby pair with read-only cache groups

The Active Data Guard with asynchronous redo transport mode supports an active standby pair replication scheme that only contains replicated read-only cache groups. All replicated read-only cache groups must be created before you create the active standby pair. You cannot exclude a replicated read-only cache group when you are creating the active standby pair and you cannot add another replicated read-only cache group to the active standby pair after creation.

When you create and configure an active standby pair to support replicated read-only cache groups, perform the following to support asynchronous Active Data Guard:

1. When you create the active standby pair, we recommend that you keep both the active and standby masters within the same physical site. They can be on different hosts within the same site.
2. If you want a read-only subscriber for disaster recovery, you can add a read-only subscriber on the same disaster recovery site as the standby Oracle database and enable the subscriber for cache groups. The subscriber that you should create when using Active Data Guard is created with a duplicate operation with the `ttRepAdmin -duplicate -activeDataGuard` options.

   The `-activeDataGuard` option, which is solely for the Active Data Guard environment, enables the subscriber to keep replicated read-only cache groups intact as it would for a standby master. Since the subscriber retains these cache groups, you must provide the cache user name and cache user password on the `ttRepAdmin` utility command line.

   Note: Alternatively, you can use the `ttRepDuplicateEx` C function setting the `TT_REPDUP_ADG` flag in `ttRepDuplicateExArg.flags`.

   The following example creates a read-only subscriber on the disaster recovery site duplicating from the standby master providing the `-activeDataGuard` option, the cache user name, and the cache user password.

   ```
   ttRepAdmin -duplicate -from master2 -host node1
    -uid cacheuser -pwd timesten -cacheuid cacheuser -cachepwd oracle
    -activeDataGuard adgsubscriber
   ```

3. Create the cache environment on the primary Oracle database. You do not need to perform any of these steps on the standby Oracle database.
4. On the primary Oracle database, grant the cache admin user the `EXECUTE` privilege for the `SYS.DBMS_FLASHBACK` package. This privilege is granted as part of the `initCacheAdminSchema.sql` and `grantCacheAdminPrivileges.sql` scripts as of the TimesTen Classic 18.1 release.

5. Configure the same connection attributes that you would for a TimesTen database that caches data from an Oracle database. In addition, since we are also monitoring transactions from the standby Oracle database, configure the `StandbyNetServiceName` connection attribute with the Oracle net service name of the standby Oracle database instance.

   On Microsoft Windows systems, the net service name of the Oracle database instance is specified in the **Oracle Net Service Name** field of the TimesTen Cache tab within the TimesTen ODBC Setup dialog box. The standby Oracle database instance is specified in the **Standby Oracle Net Service Name** field on the same page.

   Configure the `StandbyNetServiceName` `ODBC.INI` attribute on the active master to configure the net service name of the physical standby Oracle database:

   ```
   [cachedb]
   DataStore=/myDb/cachedb
   PermSize=256
   TempSize=256
   PLSQL=0
   DatabaseCharacterSet=WE8DEC
   OracleNetServiceName=primaryinstance
   StandbyNetServiceName=standbyinstance
   ```

Recovery after failure when using asynchronous Active Data Guard

The following sections describe what to do if the primary Oracle database fails, the standby Oracle database fails, or the entire primary site fails taking down the primary Oracle database as well as the active and standby masters:

- Failure of the standby Oracle database
- Failure of the primary Oracle database
- Failure of the primary site

Failure of the standby Oracle database

When the standby Oracle database in an Active Data Guard configuration fails, the cache agent retries the connection to the standby Oracle database in one of the following ways:

Note: You can notify the cache agent of whether the standby Oracle database is active or has failed by calling the `ttCacheADGStandbyStateSet` built-in procedure with either the `ON` or the `FAILED` arguments.

- If a timeout is set, then the cache agent waits for the amount of time specified with the `ttCacheADGStandbyTimeoutSet` built-in procedure. If the standby Oracle database has not recovered after this period, then the cache agent sets the state of the standby Oracle database by calling the `ttCacheADGStandbyStateSet` built-in procedure with the `FAILED` argument and then facilitates autorefresh using only the primary Oracle database.
- If no timeout has been set with the `ttCacheADGStandbyTimeoutSet` built-in procedure (default value is 0), then the cache agent continues to wait on the standby Oracle database, unless you inform the cache agent that the standby Oracle database is not recovering by calling the `ttCacheADGStandbyStateSet` built-in procedure with the `FAILED` argument.

Once the state of the standby Oracle database is set to `FAILED`, the cache agent resumes autorefresh with only the primary Oracle database until you reset the state of the standby Oracle database by calling the `ttCacheADGStandbyStateSet` built-in procedure with the `ON` argument. Even if the standby Oracle database eventually does recover, the cache agent does not recognize that the standby Oracle database is active until you reset its state to `ON`.

Once the state of the standby Oracle database is set to `ON`, the cache agent pauses to wait for the standby Oracle database to catch up to the primary Oracle database. After which, the cache agent resumes autorefresh from the primary Oracle database for those transactions that have successfully replicated to the standby Oracle database.

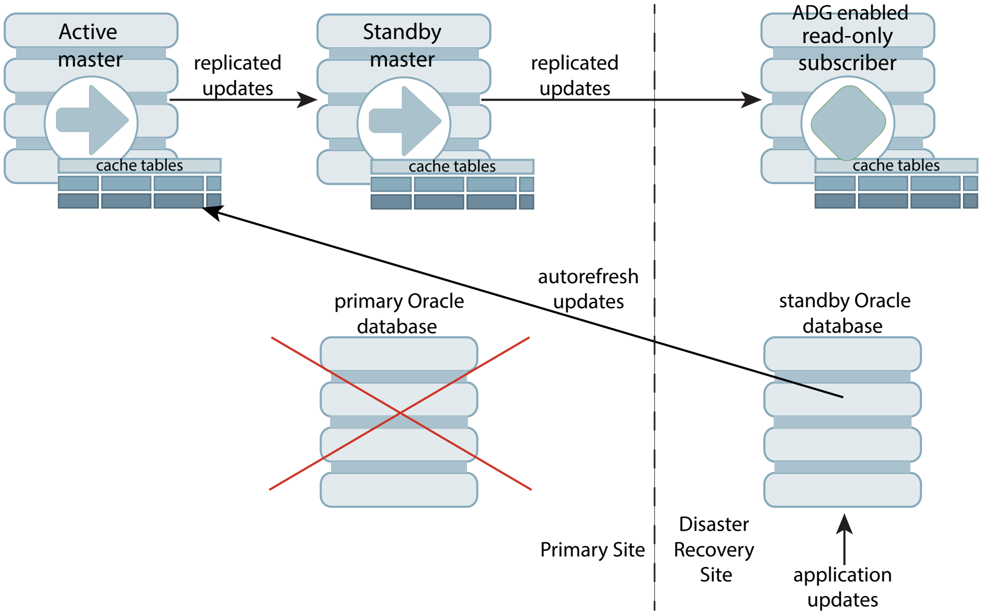
You can restore the original Active Data Guard configuration by dropping the active standby pair and then loading the cache groups.

For more details, see "ttCacheADGStandbyTimeoutSet" and "ttCacheADGStandbyStateSet" in the *Oracle TimesTen In-Memory Database Reference*.

Failure of the primary Oracle database

If the primary Oracle database fails, then Data Guard switches over to the standby Oracle database and the TimesTen cache agent switches autorefresh over to the new primary Oracle database.

Figure 10–2 Failure of the primary Oracle database



Failure of the primary site

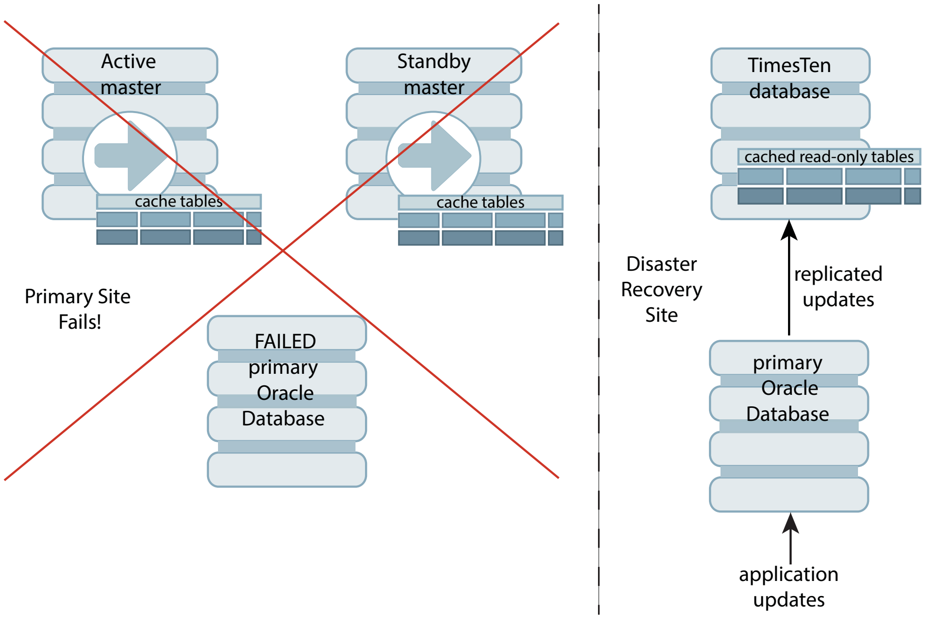
If the entire site where the primary Oracle database as well as the active and standby master databases are located fails, then the standby Oracle database becomes the primary Oracle database. Then, you may want the disaster recovery site to become the primary TimesTen database. Thus, on the disaster recovery site, the standby Oracle database is now a sole Oracle database and the read-only subscriber becomes a single TimesTen database that caches data in the Oracle database.

Transform the subscriber into a single TimesTen database with cached tables by:

1. Drop the active standby pair on the TimesTen database on the disaster recovery site.
2. Alter the existing read-only cache groups on the disaster recovery site to set the autorefresh state to on.

After which, the cache tables on the TimesTen database in the disaster recovery site receive updates from the new primary Oracle database.

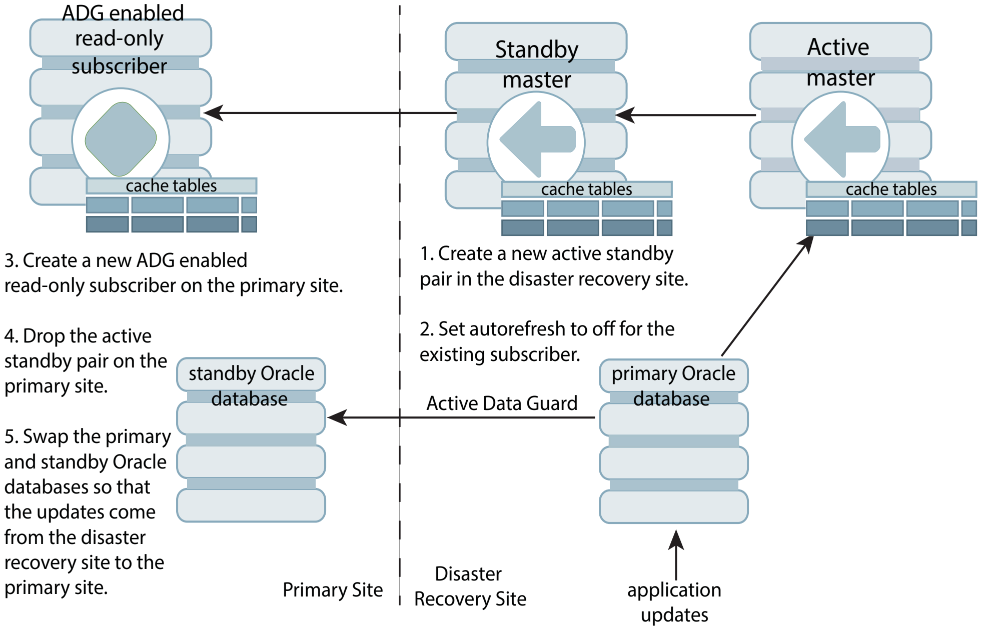
Figure 10–3 Recovery after failure of primary site



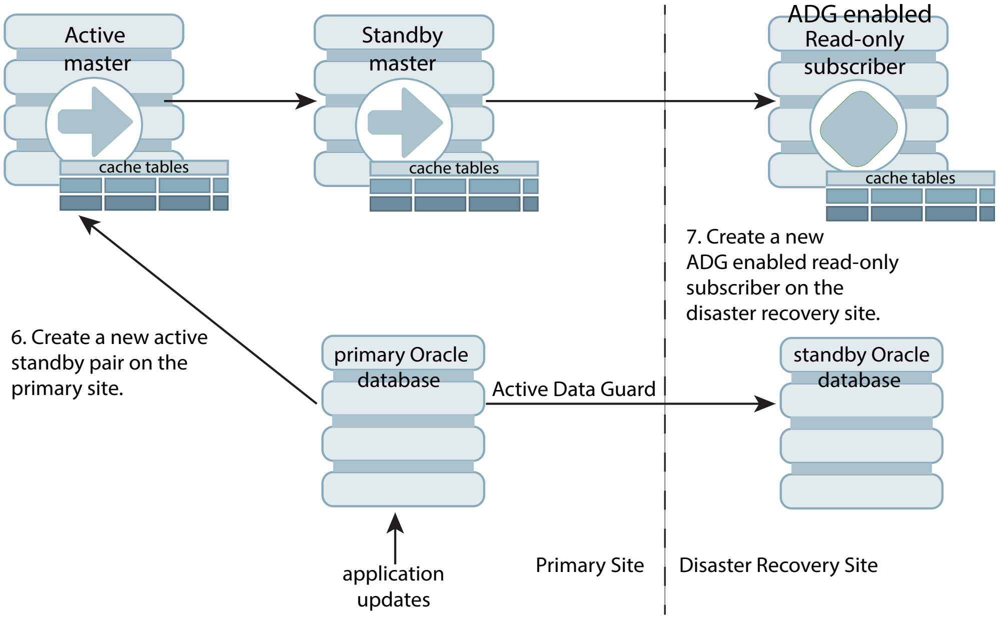
Recovering from a failure of the primary site The following is the process to recover the primary site and rebuild your environment to the original state:

1. Create a new active standby pair on the disaster recovery site.
2. Alter the existing read-only cache groups on the disaster recovery site to set the autorefresh state to off to stop any future updates from the primary Oracle database.
3. Create the ADG enabled read-only subscriber on the recovered primary site.
4. Drop the active standby pair on the ADG enabled read-only subscriber on the primary site, if it still exists after recovering the primary site.
5. Switch over the Oracle databases in the Active Data Guard. Currently, the applications are updating the primary Oracle database on the disaster recovery site. However, once you recover the Oracle database on the primary site, we want it to take over again as the primary and to make the Oracle database on the disaster recovery site as the secondary.

   The TimesTen database starts to receive updates from the Oracle database on the primary site.



6. Create a new active standby pair on the primary site.
7. Create a new ADG enabled read-only subscriber on the disaster recovery site.



TimesTen Cache works with synchronous Data Guard

TimesTen Cache works with synchronous physical standby failover and switchover and logical standby switchover as long as the object IDs for cached Oracle Database tables remain the same on the primary and standby Oracle databases. Object IDs can change if the table is dropped and re-created, altered, or a truncated flashback operation or online segment shrink is executed.

During a transient upgrade, a physical standby Oracle database is transformed into a logical standby Oracle database. For the time that the standby Oracle database is logical, the user must ensure that the object IDs of the cached Oracle Database tables do not change. Specifically, tables that are cached should not be dropped and re-created, truncated, altered, flashed back or have an online segment shrunk.

Configuring the Oracle databases

In order for TimesTen Cache to fail over and switch over properly, configure the primary and standby Oracle databases using the following steps:

1. The Data Guard configuration must be managed by the Data Guard Broker so that the TimesTen Cache daemon processes and application clients respond faster to failover and switchover events.
2. If you are configuring an Oracle RAC database, use the Oracle Enterprise Manager Cluster Managed Database Services Page to create Oracle database services that TimesTen Cache and its client applications use to connect to the Oracle primary database. See "Workload Management with Dynamic Database Services" in *Oracle Real Application Clusters Administration and Deployment Guide* for information about creating database services.
3. If you created the Oracle database service in step 2, use the `MODIFY_SERVICE` function of the `DBMS_SERVICE` PL/SQL package to modify the service to enable high availability notification to be sent through Advanced Queuing (AQ) by setting the `aq_ha_notifications` attribute to `TRUE`. To configure server side TAF settings, set the failover attributes, as shown in the following example:

```
BEGIN
DBMS_SERVICE.MODIFY_SERVICE
(service_name => 'DBSERV',
 goal => DBMS_SERVICE.GOAL_NONE,
 dtp => false,
 aq_ha_notifications => true,
 failover_method => 'BASIC',
 failover_type => 'SELECT',
 failover_retries => 180,
 failover_delay => 1);
END;
```

4. If you did not create the database service in step 2, use the `CREATE_SERVICE` function of the `DBMS_SERVICE` PL/SQL package to create the database service, enable high availability notification, and configure server side TAF settings:

```
BEGIN
DBMS_SERVICE.CREATE_SERVICE
(service_name => 'DBSERV',
 network_name => 'DBSERV',
 goal => DBMS_SERVICE.GOAL_NONE,
 dtp => false,
 aq_ha_notifications => true,
 failover_method => 'BASIC',
 failover_type => 'SELECT',
 failover_retries => 180,
 failover_delay => 1);
END;
```

5. Create two triggers to relocate the database service to a Data Guard standby database (Oracle RAC or non-Oracle RAC) after it has switched to the primary

role. The first trigger fires on the system start event and starts up the `DBSERV` service:

```
CREATE OR REPLACE TRIGGER manage_service
AFTER STARTUP ON DATABASE
DECLARE
  role VARCHAR(30);
BEGIN
  SELECT database_role INTO role FROM v$database;
  IF role = 'PRIMARY' THEN
    dbms_service.start_service('DBSERV');
  END IF;
END;
```

The second trigger fires when the standby database remains open during a failover and switchover upon a database role change. It relocates the `DBSERV` service from the old primary to the new primary database and disconnects any connections to that service on the old primary database so that TimesTen Cache and its client applications can reconnect to the new primary database:

```
CREATE OR REPLACE TRIGGER relocate_service
AFTER DB_ROLE_CHANGE ON DATABASE
DECLARE
  role VARCHAR(30);
BEGIN
  SELECT database_role INTO role FROM v$database;
  IF role = 'PRIMARY' THEN
    dbms_service.start_service('DBSERV');
  ELSE
    dbms_service.stop_service('DBSERV');
  dbms_lock.sleep(2);
  FOR x IN (SELECT s.sid, s.serial#
            FROM v$session s, v$process p
            WHERE s.service_name='DBSERV' AND s.paddr=p.addr)
    LOOP
      BEGIN
        EXECUTE IMMEDIATE
           'ALTER SYSTEM DISCONNECT SESSION
           ''' || x.sid || ','|| x.serial# || ''' IMMEDIATE';
        EXCEPTION WHEN OTHERS THEN
        BEGIN
          DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);
        END;
      END;
    END LOOP;
  END IF;
END;
```

6. As an option, to reduce the performance impact to TimesTen Cache applications and minimize the downtime during a physical or logical standby database switchover, run the following procedure right before initiating the Data Guard switchover to a physical or logical standby database:

```
DECLARE
  role varchar(30);
BEGIN
  SELECT database_role INTO role FROM v$database;
  IF role = 'PRIMARY' THEN
    dbms_service.stop_service('DBSERV');
  dbms_lock.sleep(2);
  FOR x IN (SELECT s.sid, s.serial#
            FROM v$session s, v$process p
            WHERE s.service_name='DBSERV' AND s.paddr=p.addr)
    LOOP
      BEGIN
        EXECUTE IMMEDIATE
            'ALTER SYSTEM DISCONNECT SESSION
            ''' || x.sid || ',' || x.serial# || ''' IMMEDIATE';
        EXCEPTION WHEN OTHERS THEN
        BEGIN
          DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);
        END;
      END;
    END LOOP;
  ELSE
    dbms_service.start_service('DBSERV');
  END IF;
END;
```

This procedure should be executed first on the physical or logical standby database, and then on the primary database, right before the switchover process. Before executing the procedure for a physical standby database switchover, Active Data Guard must be enabled on the physical standby database.

Before performing a switchover to a logical standby database, stop the Oracle Database service for TimesTen on the primary database and disconnect all sessions connected to that service. Then start the service on the standby database.

At this point, the cache applications try to reconnect to the standby database. If a switchover occurs, there is no wait required to migrate the connections from the primary database to the standby database. This eliminates the performance impact on TimesTen Cache and its applications.

See the *Maximum Availability Architecture, Oracle Best Practices for High Availability* white paper for more information.

Configuring the TimesTen database

Configure TimesTen to receive notification of FAN HA events and to avoid reconnecting to a failed Oracle Database instance. Use the Oracle Client shipped with TimesTen Cache.

1. Create an Oracle Net service name that includes all primary and standby hosts in `ADDRESS_LIST`. For example:

```
DBSERV =
(DESCRIPTION =
  (ADDRESS_LIST =
  (ADDRESS = (PROTOCOL = TCP)(HOST = PRIMARYDB)(PORT = 1521))
  (ADDRESS = (PROTOCOL = TCP)(HOST = STANDBYDB)(PORT = 1521))
  (LOAD_BALANCE = yes)
  )
  (CONNECT_DATA= (SERVICE_NAME=DBSERV))
)
```

2. In the client's sqlnet.ora file, set the `SQLNET.OUTBOUND_CONNECT_TIMEOUT` parameter to enable clients to quickly traverse an address list in the event of a failure. For example, if a client attempts to connect to a host that is unavailable, the connection attempt is bounded to the time specified by the `SQLNET.OUTBOUND_CONNECT_TIMEOUT` parameter, after which the client attempts to connect to the next host in

the address list. Connection attempts continue for each host in the address list until a connection is made.

Setting the `SQLNET.OUTBOUND_CONNECT_TIMEOUT` parameter to a value of 3 seconds suffices in most environments. For example, add the following entry to the sqlnet.ora file:

```
SQLNET.OUTBOUND_CONNECT_TIMEOUT=3
```

A

Procedure and Privileges for Caching Oracle Database Data in TimesTen Classic

The following sections provide a quick reference on the steps for creating a cache environment as well as the privileges required to do so:

- Quick reference to cache Oracle Database data in TimesTen Classic
- Required privileges for the cache administration user and the cache manager user

Quick reference to cache Oracle Database data in TimesTen Classic

The following section provides a quick reference on the steps necessary when setting up an environment that caches Oracle Database data into a TimesTen database. For a detailed explanation and examples for each step, see Chapter 2, "Getting Started", Chapter 3, "Setting Up a Caching Infrastructure", and Chapter 4, "Defining Cache Groups".

Perform the following on the Oracle database:

1. Create a default tablespace to be used for storing TimesTen Cache management objects.
2. As the `sys` user, create one or more schema users to own the cached Oracle Database tables (may be existing users).
3. As the `sys` user, create the cache administration user that creates, owns, and maintains Oracle Database objects that store information used to enforce predefined behaviors of particular cache group types. In the `CREATE USER` statement for the cache administration user, designate the tablespace that was created for the `timesten` user as the default tablespace for the cache administration user.

   See "Create the Oracle database users" on page 3-2 for more information about the Oracle Database users.
4. As the `sys` user, run the *`timesten_home`*`/install/oraclescripts/grantCacheAdminPrivileges.sql` script to grant the cache administration user the privileges required to create the desired types of cache groups and perform operations on the cache groups. Alternatively, you can manually create each Oracle Database object.

   See "Automatically create Oracle Database objects used to manage data caching" on page 3-4 or "Manually create Oracle Database objects used to manage data caching" on page 3-5 to determine the appropriate script to run.

5. Some privileges cannot be granted until the cached Oracle Database tables have been created. To grant these privileges, execute `GRANT` statements as the `sys` user.

   See "Required privileges for the cache administration user and the cache manager user" on page A-3 for more information about the privileges that must be granted to the cache administration user to perform particular cache operations.

Perform the following on the TimesTen database:

1. Define a DSN that references the TimesTen database that is to be used to cache data from an Oracle database.

   a. Set the `OracleNetServiceName` connection attribute to the Oracle Net service name that references the Oracle database instance.

   b. Set the `DatabaseCharacterSet` connection attribute to the Oracle database character set. The TimesTen database character set must match the Oracle database character set.

   c. Then, connect to the DSN to create the database if this is a standalone database or is to be an active database of an active standby pair.

   See "Define a DSN for the TimesTen database" on page 3-6 for more information about defining a DSN for a TimesTen database that is to be used to cache data from an Oracle database.

2. Create the following users in the TimesTen database:

   - Cache manager user

     This user must have the same name as a companion Oracle Database user that can access the cached Oracle Database tables. The companion Oracle Database user can be the cache administration user, a schema user, or some other existing user. The password of the cache manager user and the Oracle Database user with the same name can be different.

   - One or more cache table users who own the TimesTen cache tables

     These users must have the same name as the Oracle Database schema users who own the cached Oracle Database tables. The password of a cache table user and the Oracle Database user with the same name can be different.

   Execute `CREATE USER` statements as the instance administrator.

   See "Create the TimesTen users" on page 3-7 for more information about the TimesTen users.

3. Grant the cache manager user the privileges required to create the desired types of cache groups and perform operations on the cache groups. Execute `GRANT` statements as the instance administrator.

   See "Required privileges for the cache administration user and the cache manager user" on page A-3 for more information about the privileges that must be granted to the cache manager user to perform particular cache operations.

4. Set the cache administration user name and password in the TimesTen database either by calling the `ttCacheUidPwdSet` built-in procedure as the cache manager user or running a `ttAdmin -cacheUidPwdSet` utility command as a TimesTen external user with the `CACHE_MANAGER` privilege.

   See "Set the cache administration user name and password" on page 3-9 for more information about setting the cache administration user name and password in a TimesTen database.

5. Start the cache agent on the TimesTen database either by calling the `ttCacheStart` built-in procedure as the cache manager user or running a `ttAdmin -cacheStart` utility command as a TimesTen external user with the `CACHE_MANAGER` privilege.

   See "Managing the cache agent" on page 3-11 for more information about starting a cache agent on a TimesTen database.

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

   See "Mappings between Oracle Database and TimesTen data types" on page C-12 for a list of valid data type mappings between the Oracle and TimesTen databases.

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

   Then, create the cache groups. See "Creating a cache group" on page 4-7 for more information about creating a cache group.

7. If this TimesTen database is intended to be an active database of an active standby pair, create an active standby pair replication scheme in the database. For more information on creating an active standby pair replication scheme, see "Defining an Active Standby Pair Replication Scheme" in the *Oracle TimesTen In-Memory Database Replication Guide*.

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

   See "Managing the replication agent" on page 4-13 for more information about starting a replication agent on a TimesTen database.

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

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

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

Required privileges for the cache administration user and the cache manager user

The privileges that the Oracle Database users require depends on the types of cache groups you create and the operations that you perform on the cache groups. The privileges required for the cache administration user are listed in the first column and the privileges required for the TimesTen cache manager user for each cache operation are listed in the second column in Table A–1.

Table A–1 Oracle Database and TimesTen user privileges required for cache operations

Cache operation	Privileges required for Oracle Database cache administration user[1]	Privileges required for TimesTen cache manager user[2]

Initialize the cache administration user. The `grantCacheAdminPrivileges.sql` script grants these privileges to the cache administration user.	`CREATE SESSION`
	`TT_CACHE_ADMIN_ROLE`
	`EXECUTE ON SYS.DBMS_LOCK`
	`RESOURCE`[4]
	`CREATE PROCEDURE`
	`CREATE ANY TRIGGER`[3,4]
	`EXECUTE ON SYS.DBMS_LOB`
	EXECUTE ON SYS.DBMS_FLASHBACK
	SELECT ANY TRANSACTION
	`SELECT ON SYS.ALL_OBJECTS`
	`SELECT ON SYS.ALL_SYNONYMS`
	`CREATE TYPE`
	`SELECT ON SYS.GV_$LOCK`
	`SELECT ON SYS.GV_$SESSION`
	`SELECT ON SYS.DBA_DATA_FILES`
	`SELECT ON SYS.USER_USERS`
	`SELECT ON SYS.USER_FREE_SPACE`
	`SELECT ON SYS.USER_TS_QUOTAS`
	`SELECT ON SYS.USER_SYS_PRIVS`
	Permissions for the default tablespace	

Set the cache administration user or cache manager user name and password with either: ■ Call the `ttCacheUidPwdSet` built-in procedure. ■ Run the `ttAdmin -cacheUidPwdSet` utility command.	■ `CREATE SESSION` ■ `RESOURCE`[4]	`CACHE_MANAGER`
Get the cache administration user or cache manager user name with either: ■ Call the `ttCacheUidGet` built-in procedure ■ Run the `ttAdmin -cacheUidGet` utility command	None	`CACHE_MANAGER`
Start the cache agent with either: ■ Call the `ttCacheStart` built-in procedure. ■ Run the `ttAdmin -cacheStart` utility command.	`CREATE SESSION`	`CACHE_MANAGER`

Table A–1 (Cont.) Oracle Database and TimesTen user privileges required for cache operations

Cache operation	Privileges required for Oracle Database cache administration user[1]	Privileges required for TimesTen cache manager user[2]
Stop the cache agent - Call the `ttCacheStop` built-in procedure - Run the `ttAdmin -cacheStop` utility command	None	`CACHE_MANAGER`
Set a cache agent start policy with either: - Call the `ttCachePolicySet` built-in procedure. - Run the `ttAdmin -cachePolicy` utility command.	`CREATE SESSION`[5]	`CACHE_MANAGER`
Return the cache agent start policy setting: - Call the `ttCachePolicyGet` built-in procedure.	`CREATE SESSION`	None
Start the replication agent with either: - Call the `ttRepStart` built-in procedure. - Run the `ttAdmin -repStart` utility command.	None	`CACHE_MANAGER`
Stop the replication agent with either: - Call the `ttRepStop` built-in procedure. - Run the `ttAdmin -repStop` utility command.	None	`CACHE_MANAGER`
Set a replication agent start policy - Call the `ttRepPolicySet` built-in procedure - Run the `ttAdmin -repPolicy` utility command	None	`ADMIN`
`CREATE ACTIVE STANDBY PAIR` with `INCLUDE CACHE GROUP` when the cache group created is an AWT cache group	`CREATE TRIGGER`	
Duplicate the database with `ttRepAdmin -duplicate` when using an AWT cache group within an active standby pair replication scheme	`CREATE TRIGGER`	
`CREATE [DYNAMIC] READONLY CACHE GROUP` with `AUTOREFRESH MODE INCREMENTAL`	- `CREATE SESSION` - `SELECT ON` *`table_name`*[6] - `RESOURCE`[4] - `CREATE ANY TRIGGER`[4]	- `CREATE [ANY] CACHE GROUP`[7] - `CREATE [ANY] TABLE`[8]

Table A–1 (Cont.) Oracle Database and TimesTen user privileges required for cache operations

Cache operation	Privileges required for Oracle Database cache administration user[1]	Privileges required for TimesTen cache manager user[2]
CREATE [DYNAMIC] READONLY CACHE GROUP with AUTOREFRESH MODE FULL	■ CREATE SESSION ■ SELECT ON *table_name*[6]	■ CREATE [ANY] CACHE GROUP[7] ■ CREATE [ANY] TABLE[8]
CREATE [DYNAMIC] ASYNCHRONOUS WRITETHROUGH CACHE GROUP	■ CREATE SESSION ■ SELECT ON *table_name*[6] ■ RESOURCE[4]	■ CREATE [ANY] CACHE GROUP[7] ■ CREATE [ANY] TABLE[8]
CREATE [DYNAMIC] SYNCHRONOUS WRITETHROUGH CACHE GROUP	■ CREATE SESSION ■ SELECT ON *table_name*[6]	■ CREATE [ANY] CACHE GROUP[7] ■ CREATE [ANY] TABLE[8]
CREATE [DYNAMIC] USERMANAGED CACHE GROUP (see variants in following rows)	■ CREATE SESSION ■ SELECT ON *table_name*[6]	■ CREATE [ANY] CACHE GROUP[7] ■ CREATE [ANY] TABLE[8]
CREATE [DYNAMIC] USERMANAGED CACHE GROUP with AUTOREFRESH MODE INCREMENTAL	■ CREATE SESSION ■ SELECT ON *table_name*[6] ■ RESOURCE[4] ■ CREATE ANY TRIGGER[4]	■ CREATE [ANY] CACHE GROUP[7] ■ CREATE [ANY] TABLE[8]
CREATE [DYNAMIC] USERMANAGED CACHE GROUP with AUTOREFRESH MODE FULL	■ CREATE SESSION ■ SELECT ON *table_name*[6]	■ CREATE [ANY] CACHE GROUP[7] ■ CREATE [ANY] TABLE[8]
CREATE [DYNAMIC] USERMANAGED CACHE GROUP with READONLY	■ CREATE SESSION ■ SELECT ON *table_name*[6]	■ CREATE [ANY] CACHE GROUP[7] ■ CREATE [ANY] TABLE[8]
CREATE [DYNAMIC] USERMANAGED CACHE GROUP with PROPAGATE	■ CREATE SESSION ■ SELECT ON *table_name*[6]	■ CREATE [ANY] CACHE GROUP[7] ■ CREATE [ANY] TABLE[8]
ALTER CACHE GROUP SET AUTOREFRESH STATE PAUSED	■ CREATE SESSION ■ SELECT ON *table_name*[6,9] ■ RESOURCE[4, 9] ■ CREATE ANY TRIGGER[4 ,9]	ALTER ANY CACHE GROUP[10]
ALTER CACHE GROUP SET AUTOREFRESH STATE ON	■ CREATE SESSION ■ SELECT ON *table_name*[6, 9] ■ RESOURCE[4, 9] ■ CREATE ANY TRIGGER[4, 9]	ALTER ANY CACHE GROUP[10]
ALTER CACHE GROUP SET AUTOREFRESH STATE OFF	CREATE SESSION	ALTER ANY CACHE GROUP[10]
ALTER CACHE GROUP SET AUTOREFRESH MODE FULL	CREATE SESSION	ALTER ANY CACHE GROUP[10]
ALTER CACHE GROUP SET AUTOREFRESH MODE INCREMENTAL	■ CREATE SESSION ■ SELECT ON *table_name*[6] ■ RESOURCE[4] ■ CREATE ANY TRIGGER[4]	ALTER ANY CACHE GROUP[10]

Table A–1 (Cont.) Oracle Database and TimesTen user privileges required for cache operations

Cache operation	Privileges required for Oracle Database cache administration user[1]	Privileges required for TimesTen cache manager user[2]
`ALTER CACHE GROUP SET AUTOREFRESH INTERVAL`	■ `CREATE SESSION` ■ `SELECT ON` *`table_name`*[6, 11]	`ALTER ANY CACHE GROUP`[10]
`LOAD CACHE GROUP`	■ `CREATE SESSION` ■ `SELECT ON` *`table_name`*[6]	`LOAD {ANY CACHE GROUP | ON` *`cache_group_name`*`)` [10]
`REFRESH CACHE GROUP`	■ `CREATE SESSION` ■ `SELECT ON` *`table_name`*[6]	`REFRESH {ANY CACHE GROUP | ON` *`cache_group_name`*`)`[10]
`FLUSH CACHE GROUP`	■ `CREATE SESSION` ■ `UPDATE ON` *`table_name`*[6] ■ `INSERT ON` *`table_name`*[6]	`FLUSH {ANY CACHE GROUP | ON` *`cache_group_name`*`)` [10]
`UNLOAD CACHE GROUP`	None	`UNLOAD {ANY CACHE GROUP | ON` *`cache_group_name`*`)`[10]
`DROP CACHE GROUP`	`CREATE SESSION`	■ `DROP ANY CACHE GROUP`[10] ■ `DROP ANY TABLE`[12]
Synchronous writethrough or propagate	■ `CREATE SESSION` ■ `INSERT ON` *`table_name`*[6, 13] ■ `UPDATE ON` *`table_name`*[6, 13] ■ `DELETE ON` *`table_name`*[6, 13]	■ `INSERT ON` *`table_name`*[14] ■ `UPDATE ON` *`table_name`*[14] ■ `DELETE ON` *`table_name`*[14]
Asynchronous writethrough	■ `CREATE SESSION` ■ `INSERT ON` *`table_name`*[6] ■ `UPDATE ON` *`table_name`*[6] ■ `DELETE ON` *`table_name`*[6]	■ `INSERT ON` *`table_name`*[14] ■ `UPDATE ON` *`table_name`*[14] ■ `DELETE ON` *`table_name`*[14]
Asynchronous writethrough when the `CacheAWTMethod` connection attribute is set to 1	`CREATE PROCEDURE` **Note:** This privilege is an addition to the privileges needed for any asynchronous writethrough cache group.	No additional privileges
Asynchronous writethrough cache for Oracle Database `CLOB`, `BLOB` and `NCLOB` fields when the `CacheAWTMethod` connection attribute is set to 1	`EXECUTE` privilege on the Oracle Database `DBMS_LOB` PL/SQL package **Note:** This privilege is an addition to the privileges needed for any asynchronous writethrough cache group.	No additional privileges
Incremental autorefresh	`SELECT ON` *`table_name`*[6]	None
Full autorefresh	`SELECT ON` *`table_name`*[6]	None
Dynamic load	■ `CREATE SESSION` ■ `SELECT ON` *`table_name`*[6]	■ `SELECT ON` *`table_name`*[14] ■ `UPDATE ON` *`table_name`*[14] ■ `DELETE ON` *`table_name`*[14] ■ `INSERT ON` *`table_name`*[14]
Aging	None	`DELETE {ANY TABLE | ON` *`table_name`*`)`[14]

Table A–1 (Cont.) Oracle Database and TimesTen user privileges required for cache operations

Cache operation	Privileges required for Oracle Database cache administration user[1]	Privileges required for TimesTen cache manager user[2]
Set the LRU aging attributes - Call the `ttAgingLRUConfig` built-in procedure	None	`ADMIN`
Generate Oracle Database SQL statements to manually install or uninstall Oracle Database objects - Run the `ttIsql` utility's `cachesqlget` command - Call the `ttCacheSQLGet` built-in procedure	`CREATE SESSION`	`CACHE_MANAGER`
Disable or enable propagation of committed cache table updates to the Oracle database - Call the `ttCachePropagateFlagSet` built-in procedure	None	`CACHE_MANAGER`
Configure cache agent timeout and recovery method for autorefresh cache groups - Call the `ttCacheConfig` built-in procedure	`CREATE SESSION`	`CACHE_MANAGER`
Set the AWT transaction log file threshold - Call the `ttCacheAWTThresholdSet` built-in procedure	None	`CACHE_MANAGER`
Enable or disable monitoring of AWT cache groups - Call the `ttCacheAWTMonitorConfig` built-in procedure	None	`CACHE_MANAGER`
Enable or disable tracking of DDL statements issued on cached Oracle Database tables - Call the `ttCacheDDLTrackingConfig` built-in procedure	`CREATE SESSION`	`CACHE_MANAGER`

---

[1] At minimum, the cache administration user must have the `CREATE TYPE` privilege.

[2] At minimum, the cache manager user must have the `CREATE SESSION` privilege.

[3] If the cache administration user will not create autorefresh cache groups, then you can grant the `CREATE TRIGGER` privilege instead of the `CREATE ANY TRIGGER` privilege.

[4] Not required if the Oracle Database objects used to manage the caching of Oracle Database data are manually created with the `initCacheAdminSchema.sql` script.

[5] Required if the cache agent start policy is being set to `always` or `norestart`.

[6] Required on all Oracle Database tables cached in the TimesTen cache group except for tables owned by the cache administration user.

[7] The `CACHE_MANAGER` privilege includes the `CREATE [ANY] CACHE GROUP` privilege. `ANY` is required if the cache manager user creates cache groups owned by a user other than itself.

[8] `ANY` is required if any of the cache tables are owned by a user other than the cache manager user.

[9] Required if the cache group's autorefresh mode is incremental and initial autorefresh state is `OFF`, and the Oracle Database objects used to manage the caching of Oracle Database data are automatically created.

[10] Required if the TimesTen user accessing the cache group does not own the cache group.

[11] Required if the cache group's autorefresh mode is incremental.

[12] Required if the TimesTen user accessing the cache group does not own all its cache tables.

[13] The privilege must be granted to the Oracle Database user with the same name as the TimesTen cache manager user if the Oracle Database user is not the cache administration user.

[14] Required if the TimesTen user accessing the cache table does not own the table.

B

SQL*Plus Scripts for TimesTen Cache

This appendix lists the SQL*Plus scripts that are installed with TimesTen Cache used to perform various configuration, administrative and monitoring tasks, and provides links to more information including examples. All scripts are installed in the *`timesten_home`*`/install/oraclescripts` directory.

Installed SQL*Plus scripts

- `cacheCleanUp.sql`: Drops Oracle Database objects such as change log tables and triggers used to implement autorefresh operations. Script is used when a TimesTen database containing autorefresh cache groups is unavailable because the TimesTen Classic system is offline, or the database was destroyed without dropping its autorefresh cache groups. Run this script as the cache administration user. See "Dropping Oracle Database objects used by autorefresh cache groups" on page 6-13 for more information.
- `cacheInfo.sql`: Returns change log table information for all Oracle Database tables cached in an autorefresh cache group, and information about Oracle Database objects used to track DDL statements issued on cached Oracle Database tables. Script is used to monitor autorefresh operations on cache groups and DDL statements issued on cached Oracle Database tables. Run this script as the cache administration user. See "Monitoring autorefresh operations on cache groups" on page 6-4 and "Tracking DDL statements issued on cached Oracle Database tables" on page 6-5 for more information.
- `grantCacheAdminPrivileges.sql`: Grants privileges to the cache administration user that are required to automatically create Oracle Database objects used to manage the caching of Oracle Database data when particular cache group operations are performed. This includes the `TT_CACHE_ADMIN_ROLE` role that defines privileges on Oracle Database tables. Run this script as the `sys` user. See "Automatically create Oracle Database objects used to manage data caching" on page 3-4 for more information.
- `initCacheAdminSchema.sql`: Grants a minimal set of privileges to the cache administration user and manually creates Oracle Database objects used to manage the caching of Oracle Database data. This includes the `TT_CACHE_ADMIN_ROLE` role that defines privileges on Oracle Database tables. Run this script as the `sys` user. See "Manually create Oracle Database objects used to manage data caching" on page 3-5 for more information.

C

Compatibility Between TimesTen and Oracle Databases

The following sections list compatibility issues between TimesTen and Oracle Databases. The list is not complete, but it indicates areas that require special attention.

- Summary of compatibility issues
- Transaction semantics
- API compatibility
- SQL compatibility
- Mappings between Oracle Database and TimesTen data types

Summary of compatibility issues

Consider the following differences between TimesTen and Oracle databases:

- TimesTen and Oracle database metadata are stored differently. See "API compatibility" on page C-2 for more information.
- TimesTen and Oracle databases have different transaction isolation models. See "Transaction semantics" on page C-1 for more information.
- TimesTen and Oracle databases have different connection and statement properties. For example, TimesTen does not support catalog names, scrollable cursors or updateable cursors.
- Sequences are not cached and synchronized between the TimesTen database and the corresponding Oracle database. See "SQL expressions" on page C-10 for more information.
- Side effects of Oracle Database triggers and stored procedures are not reflected in the TimesTen database until after an automatic or manual refresh operation.

Transaction semantics

TimesTen and Oracle Database transaction semantics differ as follows:

- Oracle Database serializable transactions can fail at commit time because the transaction cannot be serialized. TimesTen uses locking to enforce serializability.
- Oracle Database can provide both statement-level and transaction-level consistency by using a multiversion consistency model. TimesTen does not provide statement-level consistency. TimesTen provides transaction-level consistency by using serializable isolation.

- Oracle Database users can lock tables explicitly through SQL. This locking feature is not supported in TimesTen.
- Oracle Database supports savepoints while TimesTen does not.
- In Oracle Database, a transaction can be set to be read-only or read/write. This is not supported in TimesTen.

For more information about TimesTen isolation levels and transaction semantics, see "Transaction Management" in *Oracle TimesTen In-Memory Database Operations Guide*.

API compatibility

The following sections list methods from the JDBC and ODBC APIs that have a compatibility issue with TimesTen Cache.

- JDBC API compatibility
- ODBC API compatibility

JDBC API compatibility

Compatibility issues that apply to JDBC include the following:

- JDBC database metadata functions return TimesTen metadata. If you want Oracle metadata, 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.
- Oracle extensions (`oracle.sql` and `oracle.jdbc` packages) are not supported.
- Java stored procedures are not supported in TimesTen.

java.sql.Connection

The following `Connection` methods have no compatibility issues:

```
close()
commit()
createStatement()
prepareCall()
prepareStatement()
rollback()
setAutoCommit()
```

The following methods are executed locally in TimesTen:

```
getCatalog()
getMetaData
get/setTransactionIsolation()
isReadOnly()
isClosed()
nativeSQL()
setCatalog()
setReadOnly()
```

Note: See "Transaction semantics" on page C-1 for restrictions for the `get/setTransactionIsolation()` methods.

The `isClosed()` method returns only the TimesTen connection status.

---

java.sql.Statement

The following `Statement` methods have no compatibility issues:

```
addBatch()
clearBatch()
close()
execute()
executeBatch()
executeQuery()
executeUpdate()
getResultSet()
getUpdateCount()
getWarnings()
```

The following methods are executed locally in TimesTen:

```
cancel()
get/setMaxFieldSize()
get/setMaxRows()
get/setQueryTimeout()
getMoreResults()
setEscapeProcessing()
setCursorName()
```

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:

```
addBatch()
close()
execute()
executeUpdate()
executeQuery()
getResultSet()
getUpdateCount()
setXXX()
setXXXStream()
```

The following methods are executed locally in TimesTen:

```
cancel()
get/setMaxFieldSize()
get/setMaxRows()
get/setQueryTimeout()
getMoreResults()
```

```
setEscapeProccessing()
setCursorName()
```

java.sql.CallableStatement

The same restrictions as shown for the `java.sql.Statement` and `java.sql.PreparedStatement` interfaces apply to `CallableStatement`.

- 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()` methods. The `getXXXStream()` entry points return a stream that reads data from this buffer.
- Oracle supports up to 2 GB of long or long raw data. When cached, TimesTen converts `LONG` data into `VARCHAR2` data. TimesTen converts `LONG RAW` data into `VARBINARY` data. Both `VARCHAR2` and `VARBINARY` data types can store up to a maximum 4,194,304 (2^{22}) bytes).
- Oracle always streams `LONG/LONG RAW` data even if the application does not call `getXXXStream()`.

- TimesTen does not support the `mark()`, `markSupported()`, and `reset()` methods.

ODBC API compatibility

Table C–1 describes the compatibility of ODBC functions.

Table C–1 ODBC function compatibility with TimesTen Cache

Function name	Compatibility
`SQLBindParameter`	Default TimesTen behavior matches Oracle Database behavior. See "Binding parameters and executing statements" in *Oracle TimesTen In-Memory Database C Developer's Guide* for information.
`SQLBrowseConnect`, `SQLColumnPrivileges`, `SQLExtendedFetch`, `SQLMoreResults`, `SQLSetPos`, `SQLSetScrollOptions`, `SQLTablePrivileges`	Not supported.
`SQLCancel`	There are some restrictions. In particular, `SQLCancel` cannot cancel TimesTen Cache administrative operations. See the `SQLCancel` entry in "ODBC 2.5 function support" in the *Oracle TimesTen In-Memory Database C Developer's Guide* for additional information.
`SQLGetCursorName`	There are some restrictions. See the `SQLGetCursorName` entry in "ODBC 2.5 function support" in the *Oracle TimesTen In-Memory Database C Developer's Guide* for additional information

SQL compatibility

This section compares TimesTen's SQL implementation with Oracle Database SQL. The purpose is to provide users with a list of Oracle Database SQL features not supported in TimesTen or supported with different semantics.

- Schema objects
- Nonschema objects
- Differences between Oracle Database and TimesTen tables
- Data type support
- SQL operators
- SELECT statements
- SQL subqueries
- SQL functions
- SQL expressions
- INSERT/DELETE/UPDATE/MERGE statements
- TimesTen-only SQL and built-in procedures
- PL/SQL constructs

Schema objects

TimesTen does not recognize some of the schema objects that are supported in Oracle Database. TimesTen returns a syntax error when a statement manipulates or uses these objects. TimesTen passes the statement to Oracle Database. The unsupported objects are:

Clusters
Objects created by the `CREATE DATABASE` statement
Objects created by the `CREATE JAVA` statement
Database links
Database triggers
Dimensions
Extended features
External procedure libraries
Index-organized tables
Mining models
Partitions
Object tables, types and views
Operators

TimesTen supports views and materialized views, but it cannot cache an Oracle Database view. TimesTen can cache an Oracle Database materialized view in a user-managed cache group without the `AUTOREFRESH` cache group attribute and `PROPAGATE` cache table attribute. The cache group must be manually loaded and flushed.

Caching Oracle Database partitioned tables

TimesTen can cache Oracle Database partitioned tables at the table level, but individual partitions cannot be cached. The following describes how operations on partitioned tables affect cache groups:

- DDL operations on a table that has partitions do not affect the cache group unless there is data loss. For example, if a partition with data is truncated, an `AUTOREFRESH` operation does not delete the data from the corresponding cached table.
- `WHERE` clauses in any cache group operations cannot reference individual partitions or subpartitions. Any attempt to define a single partition of a table returns an error.

Nonschema objects

TimesTen does not recognize some of the schema objects that are supported in Oracle Database. TimesTen returns a syntax error when a statement manipulates or uses these objects. TimesTen passes the statement to Oracle Database. The unsupported objects are:

Contexts
Directories
Editions
Restore points
Roles
Rollback segments
Tablespaces

Differences between Oracle Database and TimesTen tables

The Oracle Database table features that TimesTen does not support are:

- `ON DELETE SET NULL`

- Check constraints
- Foreign keys that reference the table on which they are defined

Data type support

The following Oracle Database data types are not supported by TimesTen:

`TIMESTAMP WITH TIME ZONE`
`TIMESTAMP WITH LOCAL TIME ZONE`
`INTERVAL YEAR TO MONTH`
`INTERVAL DAY TO SECOND`
`UROWID`
`BFILE`
Oracle Database-supplied types
User-defined types

The following TimesTen data types are not supported by Oracle Database:

`TT_CHAR`
`TT_VARCHAR`
`TT_NCHAR`
`TT_NVARCHAR`
`TT_BINARY`
`TT_VARBINARY`
`TINYINT` and `TT_TINYINT`
`TT_SMALLINT`
`TT_INTEGER`
`TT_BIGINT`
`TT_DECIMAL`
`TT_DATE`
`TIME` and `TT_TIME`
`TT_TIMESTAMP`

Note: TimesTen `NCHAR` and `NVARCHAR2` data types are encoded as `UTF-16`. Oracle Database `NCHAR` and `NVARCHAR2` data types are encoded as either `UTF-16` or `UTF-8`.

To cache an Oracle Database `NCHAR` or `NVARCHAR2` column, the Oracle Database `NLS_NCHAR_CHARACTERSET` encoding must be `AL16UTF16`, not `AL32UTF8`.

SQL operators

TimesTen supports these operators and predicates that are supported by Oracle Database:

unary -
`+, -, *, /`
`=, <, >, <=, >=, <>, !=`
`||`
`IS NULL`, `IS NOT NULL`
`LIKE` (Oracle Database `LIKE` operator ignores trailing spaces, but TimesTen does not)
`BETWEEN`
`IN`
`NOT IN` (list)
`AND`

`OR`
`+` (outer join)
`ANY`, `SOME`
`ALL` (list)
`EXISTS`
`UNION`
`MINUS`
`INTERSECT`

To perform a bitwise `AND` operation of two bit vector expressions, TimesTen uses the ampersand character (`&`) between the expressions while Oracle Database uses the `BITAND` function with the expressions as arguments.

SELECT statements

TimesTen supports these clauses of a `SELECT` statement that are supported by Oracle Database:

- `FOR UPDATE`
- `ORDER BY`, including `NULLS FIRST` and `NULLS LAST`
- `GROUP BY`, including `ROLLUP`, `GROUPING_SETS` and grouping expression lists
- Table alias
- Column alias
- Subquery factoring clause with constructor

Oracle Database 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.

TimesTen does not support the `CONNECT BY` clause.

SQL subqueries

TimesTen supports these subqueries that are supported by Oracle Database:

`IN` (subquery)
`>,<,= ANY` (subquery)
`>,=,< SOME` (subquery)
`EXISTS` (subquery)
`>,=,<` (scalar subquery)
Subqueries in `WHERE` clause of `DELETE/UPDATE`
Subqueries in `FROM` clause
Subquery factoring clause (`WITH` constructor)

Note: A nonverifiable scalar subquery is a scalar subquery whose 'single-row-result-set' property cannot be determined until execution time. TimesTen allows at most one nonverifiable scalar subquery in the entire query and the subquery cannot be specified in an `OR` expression.

SQL functions

TimesTen supports these functions that are supported by Oracle Database:

`ABS`

`ADD_MONTHS`
`ASCIISTR`
`AVG`
`CAST`
`CEIL`
`COALESCE`
`CONCAT`
`COUNT`
`CHR`
`DECODE`
`DENSE_RANK`
`EMPTY_BLOB`
`EMPTY_CLOB`
`EXTRACT`
`FIRST_VALUE`
`FLOOR`
`GREATEST`
`GROUP_ID`
`GROUPING`
`GROUPING_ID`
`INSTR`
`LAST_VALUE`
`LEAST`
`LENGTH`
`LOWER`
`LPAD`
`LTRIM`
`MAX`
`MIN`
`MOD`
`MONTHS_BETWEEN`
`NCHR`
`NLS_CHARSET`
`NLS_CHARSET_NAME`
`NLSSORT`
`NULLIF`
`NUMTOYMINTERVAL`
`NUMTODSINTERVAL`
`NVL`
`POWER`
`RANK`
`REPLACE`
`ROUND`
`ROW_NUMBER`
`RPAD`
`RTRIM`
`SIGN`
`SQRT`
`SUBSTR`
`SUM`
`SYS_CONTEXT`
`SYSDATE`
`TO_BLOB`
`TO_CLOB`
`TO_CHAR`

`TO_DATE`
`TO_LOB`
`TO_NCLOB`
`TO_NUMBER`
`TRIM`
`TRUNC`
`UID`
`UNISTR`
`UPPER`
`USER`

These TimesTen functions are not supported by Oracle Database:

`CURRENT_USER`
`GETDATE`
`ORA_SYSDATE`
`SESSION_USER`
`SYSTEM_USER`
`TIMESTAMPADD`
`TIMESTAMPDIFF`
`TT_HASH`
`TT_SYSDATE`

TimesTen and the Oracle Database interpret the literal `N'\UNNNN'` differently. In TimesTen, `N'\u`*`nnnn`*`'` (where *nnnn* is a number) is interpreted as the national character set character with the code *nnnn*. In the Oracle Database, `N'\u`*`nnnn`*`'` is interpreted as 6 literal characters. The `\u` is not treated as an escape. This difference causes unexpected behavior. For example, loading a cache group with a `WHERE` clause that contains a literal can fail. This can also affects dynamic loading. Applications should use the `UNISTR` SQL function instead of literals.

SQL expressions

TimesTen supports these expressions that are supported by Oracle Database:

Column Reference
Sequence
`NULL`
`()`
Binding parameters
`CASE` expression
`ROWID` pseudocolumn
`ROWNUM` pseudocolumn

TimesTen and Oracle Database treat literals differently. See the description of *`HexadecimalLiteral`* in "Constants" in *Oracle TimesTen In-Memory Database SQL Reference*.

INSERT/DELETE/UPDATE/MERGE statements

TimesTen supports these DML statements that are supported by Oracle Database:

- `INSERT INTO ... VALUES`
- `INSERT INTO ... SELECT`
- `UPDATE WHERE` expression (expression may contain a subquery)

- `DELETE WHERE` expression (expression may contain a subquery)

TimesTen does not support updating of primary key values except when the new value is the same as the old value.

TimesTen-only SQL and built-in procedures

This section lists TimesTen SQL statements and functions and built-in procedures that are not supported by Oracle Database. With `PassThrough`=3, these statements are passed to Oracle Database for execution and an error is generated.

- All TimesTen cache group DDL and DML statements, including `CREATE CACHE GROUP`, `DROP CACHE GROUP`, `ALTER CACHE GROUP`, `LOAD CACHE GROUP`, `UNLOAD CACHE GROUP`, `REFRESH CACHE GROUP` and `FLUSH CACHE GROUP`.
- All TimesTen replication management DDL statements, including `CREATE REPLICATION`, `DROP REPLICATION`, `ALTER REPLICATION`, `CREATE ACTIVE STANDBY PAIR`, `ALTER ACTIVE STANDBY PAIR` and `DROP ACTIVE STANDBY PAIR`.
- `FIRST` *n* clause.
- `ROWS` *m* `TO` *n* clause.
- All TimesTen built-in procedures. See "Built-In Procedures" in *Oracle TimesTen In-Memory Database Reference*.
- TimesTen specific syntax for character and unicode strings are not always converted to the Oracle Database syntax when using `PassThrough`=3.

> **Note:** For more details on TimesTen support for unicode strings, see the "Character and unicode strings" section in the *Oracle TimesTen In-Memory Database Reference*.

  - Supplying `\046` converts to the `&` symbol on TimesTen, but is not converted to this symbol when passed through to an Oracle database. The `\`*xyz* notation is not supported by the Oracle database. To send a character through to an Oracle database, pass it as an argument within the `CHR()` function with the decimal value of the character.
  - TimesTen enables depicting a unicode value (a four-digit hexadecimal number) within a character string with the `\u`*xyzw* syntax (for `NCHAR` and `NVARCHAR2` only) where you substitute the unicode value for *xyzw*, as in`\ufe4a`.

    The `\u`*xyzw* notation is not supported by the Oracle database. Thus, any unicode strings in `NCHAR` or `NVARCHAR2` columns passed through to an Oracle database must be passed as an argument within the `UNISTR()` function without the `u` character.

    The following example inserts the unicode values `'0063'` and `'0064'`, which are the `a` and `b` characters respectively. Since we are using `PassThrough`=3, this statement is executed on the Oracle database; thus, we do not provide the `u` character as we would if this was executed on TimesTen.

```
Command> INSERT INTO my_tab VALUES (UNISTR(n'\0063\0064'));
1 row inserted.
```

PL/SQL constructs

TimesTen supports a subset of stored procedure constructs, functions, data types, packages and package bodies that are supported by Oracle Database. See *Oracle TimesTen In-Memory Database PL/SQL Developer's Guide* for details.

Mappings between Oracle Database and TimesTen data types

When you choose data types for columns in the TimesTen cache tables, consider the data types of the columns in the Oracle Database tables and choose an equivalent or compatible data type for the columns in the cache tables.

Note: TimeTen cache, including passthrough, does not support the Oracle Database `ROWID` data type. However, you can cast a `ROWID` data type to a `CHAR(18)` when provided on the `SELECT` list in a SQL query.

The following example demonstrates the error that is returned when you do not cast the `ROWID` data type. Then, the example shows the correct casting of a `ROWID` data type to `CHAR(18)`:

```
Command> SET PASSTHROUGH 3;
Passthrough command has set autocommit off.
Command> SELECT ROWID FROM dual;
 5115: Unsupported type mapping for column ROWID
The command failed.
Command> SELECT CAST (ROWID AS CHAR(18)) FROM DUAL;
< AAAAB0AABAAAAEoAAA >
1 row found.
```

Primary and foreign key columns are distinguished from non-key columns. The data type mappings allowed for key columns in a cache table are shown in Table C–2.

Table C–2 Data type mappings allowed for key columns

Oracle Database 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`
	`NUMBER(`*p*`,0)`

`NUMBER`	`TT_TINYINT`
	`TT_SMALLINT`
	`TT_INTEGER`
	`TT_BIGINT`
	`NUMBER`

`CHAR(`*n*`)`	`CHAR(`*n*`)`
`VARCHAR2(`*n*`)`	`VARCHAR2(`*n*`)`
`RAW(`*n*`)`	`VARBINARY(`*n*`)`

Table C–2 (Cont.) Data type mappings allowed for key columns

Oracle Database data type	TimesTen data type
`DATE`	`DATE`
`TIMESTAMP(`*n*`)`	`TIMESTAMP(`*n*`)`
`NCHAR(`*n*`)`	`NCHAR(`*n*`)`
`NVARCHAR2(`*n*`)`	`NVARCHAR2(`*n*`)`

Table C–3 shows the data type mappings allowed for non-key columns in a cache table.

Table C–3 Data type mappings allowed for non-key columns

Oracle Database 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`
	`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(`*n*`)`	`CHAR(`*n*`)`
`VARCHAR2(`*n*`)`	`VARCHAR2(`*n*`)`
`RAW(`*n*`)`	`VARBINARY(`*n*`)`
`LONG`	`VARCHAR2(`*n*`)` Where *n* can be any valid value within the range defined for the `VARCHAR2` data type.

Table C–3 (Cont.) Data type mappings allowed for non-key columns

Oracle Database data type	TimesTen data type
`LONG RAW`	`VARBINARY(`*n*`)` Where *n* can be any valid value within the range defined for the `VARBINARY` data type.

`DATE`	`DATE`
	`TIMESTAMP(0)`

`TIMESTAMP(`*n*`)`	`TIMESTAMP(`*n*`)`

`FLOAT(`*n*`)`
Note: Includes `DOUBLE` 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`. `FLOAT(63)` can be declared as `REAL`.

`BINARY_FLOAT`	`BINARY_FLOAT`
`BINARY_DOUBLE`	`BINARY_DOUBLE`
`NCHAR(`*n*`)`	`NCHAR(`*n*`)`
`NVARCHAR2(`*n*`)`	`NVARCHAR2(`*n*`)`
`CLOB`	`VARCHAR2(`*n*`)` Where 1 <= *n* <= 4 MB.
`BLOB`	`VARBINARY(`*n*`)` Where 1 <= *n* <= 4 MB.
`NCLOB`	`NVARCHAR2(`*n*`)` Where 1 <= *n* <= 2 MB.

Index