Oracle TimesTen In-Memory Database Introduction

Release 7.0

B31687-02

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

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

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

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

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

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

April 2007

Printed in the United States of America

Contents

About this Guide

TimesTen documentation 1
Background reading 3
Conventions used in this guide 4
Technical Support 6

1 What is Oracle TimesTen In-Memory Database?

Why is Oracle TimesTen In-Memory Database fast? 8
TimesTen feature overview 10
- ODBC and JDBC interfaces 10
- SQL functionality 10
- Access Control 10
- Distributed transactions 11
- Database connectivity 11
- Durability 11
- Query optimization 12
- Concurrency 12
- Automatic data aging 13
- Globalization support 13
- Transaction log monitoring 13
- Administration and utilities 14
- Replication – TimesTen to TimesTen 14
- Cache Connect to Oracle 14

2 How is TimesTen Used?

General uses for TimesTen 18
TimesTen application scenarios 19
- Scenario 1: Caller usage metering application 19
- Scenario 2: Real-time quote service application 22
- Scenario 3: Call center application 24

3 TimesTen Components

TimesTen basic architecture 26
- Shared libraries 26
- Memory-resident data structures 27
- Database processes 27
- Administrative programs 27
- Checkpoint and log files 27

Replication architecture 28
Cache Connect architecture 30
TimesTen connection options 31
Direct driver connection 31
Client/server connection 31
Driver manager connection 32
TimesTen APIs 33
Transaction Log API 33
Distributed Transaction Processing APIs 33
TTClasses 33
For more information 33

4 Concurrent Operations

Transaction isolation 36
Read committed isolation 36
Serializable isolation 37
Locks 38
Database-level locking 38
Table-level locking 38
Row-level locking 39
For more information 39

5 Query Optimization

Optimization time and memory usage 42
Statistics 42
Optimizer hints 42
Indexing techniques 43
T-tree indexes 43
Hash indexes 44
Scan methods 44
Join methods 45
Nested loop join 45
Merge join 46
Optimizer plan. 47
For more information 48

6 Data Availability and Integrity

Logging 49
Writing the log buffer to disk 49
When are log files deleted? 50
Disabling logging 50
Checkpointing 50

Nonblocking checkpoints . . . 51
Blocking checkpoints . . . 51
Recovery from log and checkpoint files . . . 51
Replication . . . 51
Replication configurations . . . 52
Active standby pair . . . 55
Asynchronous and return service replication . . . 56
Replication failover and recovery . . . 58
For more information . . . 59

7 Event Notification

Transaction Log API . . . 61
How XLA works . . . 62
Log update records . . . 63
Materialized views and XLA . . . 63
SNMP traps . . . 64
For more information . . . 65

8 Cache Connect to Oracle

Cache groups . . . 67
Cache instances . . . 68
Loading and updating cache groups . . . 70
Oracle-to-TimesTen updates . . . 71
TimesTen-to-Oracle updates . . . 72
Aging feature . . . 73
Passthrough feature . . . 73
System-managed cache groups . . . 75
Usermanaged cache groups . . . 76
Replicating cache groups . . . 76
For more information . . . 76

9 TimesTen Administration

Installing TimesTen . . . 77
TimesTen Access Control . . . 77
Command line administration . . . 78
SQL administration . . . 79
Browser-based cache group administration . . . 79
ODBC Administrator . . . 79
Upgrading TimesTen . . . 79
In-place upgrades . . . 80
Offline upgrades . . . 80
Online upgrades . . . 80

For more information 80

Index

About this Guide

This guide provides an introduction to the Oracle TimesTen In-Memory Database. Readers of this guide will benefit most if they have a basic understanding of database systems.

TimesTen documentation

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

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

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

Oracle TimesTen In-Memory Database TTClasses Guide	Describes how to use the TTClasses C++ API to use the features available in TimesTen to develop and implement applications.
TimesTen to TimesTen Replication Guide	Provides information to help you understand how TimesTen Replication works and step-by-step instructions and examples that show how to perform the most commonly needed tasks. This guide is for application developers who use and administer TimesTen and for system administrators who configure and manage TimesTen Replication.
TimesTen Cache Connect to Oracle Guide	Describes how to use Cache Connect to cache Oracle data in TimesTen data stores. This guide is for developers who use and administer TimesTen for caching Oracle data.
Oracle TimesTen In-Memory Database Troubleshooting Procedures Guide	Provides information and solutions for handling problems that may arise while developing applications that work with TimesTen, or while configuring or managing TimesTen.

Background reading

For a Java reference, see:

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

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

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

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

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

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

For a review of SQL, see:

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

For information about Unicode, see:

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

Conventions used in this guide

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

TimesTen documentation uses these typographical conventions:

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

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

If you see...	It means...
`fixed width italics`	Variable; must be replaced with an appropriate value.
[ ]	Square brackets indicate that an item in a command line is optional.
{ }	Curly braces indicated that you must choose one of the items separated by a vertical bar (|) in a command line.
|	A vertical bar (or pipe) separates arguments that you may use more than one argument on a single command line.
. . .	An ellipsis (. . .) after an argument indicates that you may use more than one argument on a single command line.
%	The percent sign indicates the UNIX shell prompt.
#	The number (or pound) sign indicates the UNIX root prompt.

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

If you see...	It means...
install_dir	The path that represents the directory where the current release of TimesTen is installed.
TTinstance	The instance name for your specific installation of TimesTen. Each installation of TimesTen must be identified at install time with a unique alphanumeric instance name. This name appears in the install path. The instance name "giraffe" is used in examples in this guide.
bits or *bb*	Two digits, either 32 or 64, that represent either the 32-bit or 64-bit operating system.
release or *rr*	Two digits that represent the first two digits of the current TimesTen release number, with or without a dot. For example, 60 or 7.0 represents TimesTen Release 7.0.
jdk_version	Two digits that represent the version number of the major JDK release. Specifically, 14 represent JDK 1.4; 5 represents JDK 5.
timesten	A sample name for the TimesTen instance administrator. You can use any legal user name as the TimesTen administrator. On Windows, the TimesTen instance administrator must be a member of the Administrators group. Each TimesTen instance can have a unique instance administrator name.
DSN	The data source name.

Technical Support

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

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

1

What is Oracle TimesTen In-Memory Database?

Oracle TimesTen In-Memory Database is a memory-resident relational database that empowers applications with the instant responsiveness and very high throughput required by today's real-time enterprises and industries such as telecom, capital markets and defense. Deployed in the application tier as an embedded database, Oracle TimesTen In-Memory Database operates on databases that fit entirely in physical memory using standard SQL interfaces.

Oracle TimesTen In-Memory Database delivers real-time performance by changing the assumptions about where data resides at runtime. By managing data in memory and optimizing data structures and access algorithms accordingly, database operations execute with maximum efficiency, achieving dramatic gains in responsiveness and throughput, even compared with a fully cached, disk-based relational database management system (RDBMS). Oracle TimesTen In-Memory Database libraries are also embedded within applications, eliminating context switching and unnecessary network operations, further improving performance.

Following the standard relational data model, SQL, JDBC and ODBC are used to access Oracle TimesTen In-Memory Databases. The use of SQL to shield applications from system internals allows databases to be altered or extended without impacting existing applications. New services can be quickly added into a production environment simply by adding application modules, tables and columns. As with any mainstream RDBMS, a cost-based optimizer automatically determines the fastest way to process queries and transactions. Any developer familiar with Oracle Databases or SQL interfaces can be immediately productive developing real-time applications with Oracle TimesTen In-Memory Database.

Why is Oracle TimesTen In-Memory Database fast?

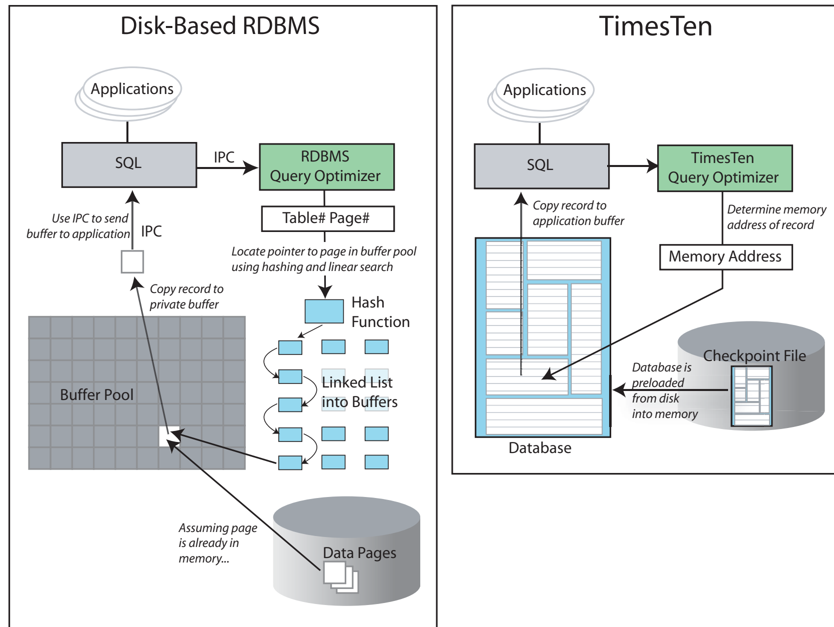
Much of the work that is done by a conventional, disk-optimized RDBMS is done under the assumption that data primarily resides on disk. Optimization algorithms, buffer pool management, and indexed retrieval techniques are designed based on this fundamental assumption.

Even when an RDBMS has been configured to hold all of its data in main memory, its performance is hobbled by assumptions of disk-based data residency. These assumptions cannot be easily reversed because they are hard-coded in processing logic, indexing schemes, and data access mechanisms.

Oracle TimesTen In-Memory Database (TimesTen) is designed with the knowledge that data resides in main memory and can take more direct routes to data, reducing the length of the code path and simplifying algorithms and structure.

When the assumption of disk-residency is removed, complexity is dramatically reduced. The number of machine instructions drops, buffer pool management disappears, extra data copies are not needed, index pages shrink, and their structure is simplified. The design becomes simple and more compact, and requests are executed faster. Figure 1.1 shows the simplicity of the TimesTen design.

Figure 1.1 Comparing a disk-based RDBMS to TimesTen



In a conventional RDBMS, client applications communicate with a database server process over some type of IPC connection, which adds performance overhead to all SQL operations. An application can link TimesTen directly into its address space to eliminate the IPC overhead and streamline query processing. This is accomplished through a *direct connection* to TimesTen. Traditional client-server access is also supported for functions such as reporting, or when a large number of application-tier platforms must share access to a common in-memory database. From an application's perspective, the TimesTen API is identical whether it is a direct connection or a client/server connection.

TimesTen feature overview

TimesTen has many familiar database features as well as some unique features. This section includes the following topics:

- ODBC and JDBC interfaces
- SQL functionality
- Access Control
- Distributed transactions
- Database connectivity
- Durability
- Query optimization
- Concurrency
- Automatic data aging
- Globalization support
- Transaction log monitoring
- Administration and utilities
- Replication – TimesTen to TimesTen
- Cache Connect to Oracle

ODBC and JDBC interfaces

TimesTen supports ODBC and JDBC. Unlike many other database systems, where ODBC or JDBC API support may be much slower than the proprietary interface, ODBC and JDBC are native TimesTen interfaces that operate directly with the database engine. TimesTen supports versions of these APIs that are both fully compliant with the standards and tuned for maximum performance in the TimesTen environment.

SQL functionality

TimesTen supports a wide range of SQL-92 functionality, including indexes and materialized views, as well as SQL extensions to simplify the configuration and management of special features, such as replication and Cache Connect to Oracle.

Access Control

TimesTen can be installed with an additional layer of user-access control by enabling the *Access Control* feature. TimesTen Access Control uses standard SQL operations to establish TimesTen user accounts with specific privilege levels.

See "TimesTen Access Control" on page 77 for more information on TimesTen Access Control.

Distributed transactions

TimesTen supports distributed transactions through the XA and JTA interfaces. These standard interfaces allow TimesTen to interoperate with transaction managers in distributed transaction processing (DTP) environments.

Database connectivity

TimesTen supports direct driver connections for higher performance, as well as connections through a driver manager. TimesTen also supports client/server connections.

These connection options allow users to choose the best performance/ functionality tradeoff for their applications. Direct driver connections are fastest; client/server connections may provide more flexibility; and driver manager connections can provide support for ODBC applications written for multiple ODBC versions.

See "TimesTen connection options" on page 31".

Durability

Durability is achieved through a combination of transaction logging and periodic refreshes of a disk-resident version of the database. Log records are written to disk asynchronous or synchronous to the completion of the transaction and controlled by the application at the transaction level. For systems where maximum throughput is paramount, such as non-monetary transactions within network systems, asynchronous logging allows extremely high throughput with minimal exposure. In cases where data integrity must be preserved, such as securities trading, Oracle TimesTen In-Memory Database ensures complete durability, with no loss of data.

TimesTen transaction logging is used in the following situations:

- Recover transactions if the application or database fails
- Undo transactions that are rolled back
- Replicate changes to other TimesTen databases
- Replicate TimesTen changes to Oracle tables
- Enable applications to detect changes to tables (using the XLA API)

TimesTen has a checkpoint operation that takes place in the background and has very little impact on database applications. This is called a "fuzzy" checkpoint. TimesTen also has a blocking checkpoint that does not require log files for recovery. Checkpoints are automatic. TimesTen maintains two checkpoint files in case a failure occurs in mid-checkpoint. Checkpoint files should reside on disks

separate from the transaction logs to minimize the impact of checkpointing on regular application activity.

See the following sections for more information:

- "Logging" on page 49
- "When are log files deleted?" on page 50
- "Checkpointing" on page 50

Query optimization

TimesTen has a cost-based query optimizer that chooses the best query plan based on factors such as the ordering of tables and choice of access method.

Optimizer cost sensitivity is somewhat higher in TimesTen than in disk-based systems because the cost structure of a main-memory system differs from that of disk-based systems in which disk access is a dominant cost factor. Because disk access is not a factor in TimesTen, the optimization cost model includes factors not usually considered by optimizers for disk-based systems, such as the cost of evaluating predicates.

TimesTen provides T-tree and hash indexes and supports two types of join methods (nested-loop and merge-join). The optimizer can create temporary indexes on the fly.

The TimesTen optimizer also accepts hints that give applications the flexibility to make tradeoffs between such factors as temporary space usage and performance.

See Chapter 5, "Query Optimization" for more information about TimesTen's query optimizer and indexing techniques.

Concurrency

TimesTen provides full support for shared databases. Unlike most database systems, TimesTen provides options to allow users to determine the optimum balance among response time, throughput, and transaction semantics for their system.

Read-committed isolation provides non-blocking operations and is the default isolation level. For databases with extremely strict transaction semantics, serializable isolation is available. These isolation levels conform to ODBC standards and are implemented with optimal performance in mind. As defined by the ODBC standard, a default isolation level can be set for a TimesTen database, which can be dynamically modified for each connection at runtime.

Most algorithms in the TimesTen database are multithreaded.

For more information about managing concurrent operations in TimesTen, see Chapter 4, "Concurrent Operations."

Automatic data aging

Data aging is an operation to remove data that is no longer needed. There are two general types of data aging: removing old data based on some time value or removing data that has been least recently used (LRU). For example, you can remove yesterday's price list, remove profiles and preferences of users who have logged out from the system, or remove detailed records that are more than 2 days old.

Two types of automatic data aging capability for TimesTen database tables and cache groups are available:

- Time-based data aging based on timestamp values
- Usage-based data aging based on the LRU algorithm

For more information, see "Implementing aging in your tables" in *Oracle TimesTen In-Memory Database Operations Guide* and "Implementing aging in a cache group" in *TimesTen Cache Connect to Oracle Guide*.

Globalization support

TimesTen provides globalization support for storing, retrieving, and processing data in native languages. Over 50 different national, multinational, and vendor-specific character sets including the most popular single-byte and multibyte encodings, plus Unicode, are supported as the database storage character set. The connection character set can be defined to enable an application running in a different encoding to communicate to the TimesTen database; character set conversion between the application and the database occurs automatically and transparently.

TimesTen offers linguistic sorting capabilities that handle the complex sorting requirements of different languages and cultures. More than 80 linguistic sorts are provided. They can be extended to enable the application to perform case-insensitive and accent-insensitive sorting and searches.

For more information, see Chapter 4, "Globalization Support" in *Oracle TimesTen In-Memory Database Operations Guide*.

Transaction log monitoring

Like several other database systems, TimesTen has an API that allows applications to monitor update activities in order to generate actions outside the database. In TimesTen, this capability is provided by the Transaction Log API (XLA), which allows applications to monitor update records as they are written to the transaction log and take various actions based on the detected updates. For example, an XLA application can apply the detected updates to another database, which could be TimesTen or a disk-based RDBMS. Another type of XLA application can notify subscribers that an update of interest has taken place.

TimesTen provides materialized views that can be used in conjunction with the XLA logging API to enable notification of events described by SQL queries.

See Chapter 7, "Event Notification" for more information about the logging API and materialized views.

Administration and utilities

TimesTen supports typical database utilities such as interactive SQL, backup and restore, copy (copies data between different database systems), and migrate (a high speed copy for moving data between different versions of TimesTen).

Similar to most other database systems, SQL configuration is available for many other administrative activities, such as creating indexes and altering tables. TimesTen also uses SQL extensions to set up replication, Cache Connect to Oracle, and materialized views. A web-based administrator is also available for setting up Cache Connect to Oracle.

TimesTen built-in procedures and C language functions enable programmatic control over TimesTen operations and settings. TimesTen command-line utilities allow users to monitor the status of connections, locks, replication, and so on. Status can also be obtained using SQL SELECT queries on the system tables in the TimesTen schema. For example, the TimesTen SYS.MONITOR table records many statistics that are useful for analyzing or debugging a TimesTen application.

For more information on TimesTen administration, see Chapter 9, "TimesTen Administration."

Replication – TimesTen to TimesTen

Replication – TimesTen to TimesTen is an option to the Oracle TimesTen In-Memory Database that enables real-time data replication between servers for high availability and load sharing. Data replication configurations can be active-standby or active-active, using asynchronous or synchronous transmission, with conflict detection and resolution and automatic resynchronization after a failed server is restored. Data replication is fully compatible with the Cache Connect to Oracle option.

See "Replication" on page 51.

Cache Connect to Oracle

Cache Connect to Oracle is an option to the Oracle TimesTen In-Memory Database that creates a real-time, updatable cache for Oracle data. It offloads computing cycles from Oracle databases and enables responsive and scalable real-time applications. Cache Connect to Oracle loads a subset of Oracle tables into TimesTen. It can be configured to propagate updates in both directions and to automate passthrough of SQL requests for non-cached data. It automatically

resynchronizes data after failures. Cache Connect to Oracle is compatible with the Replication – TimesTen to TimesTen option.

See Chapter 8, “Cache Connect to Oracle.”

2

How is TimesTen Used?

This chapter describes how TimesTen can used to enable applications that require real-time access to data. It includes the following sections:

- General uses for TimesTen
- TimesTen application scenarios

General uses for TimesTen

In general, TimesTen can be used as:

- The *primary database* for real-time applications. In this case all data needed by the applications resides in TimesTen.
- A *real-time data manager* for specific tasks in an overall workflow in collaboration with a disk-based RDBMS. For example, a phone billing application may capture and store recent call records in TimesTen while storing information about customers, their billing addresses, credit information, and so on in a disk-based RDBMS. It may also age and keep archives of all call records in the disk-based database. Thus, the information that requires real-time access is stored in TimesTen while the information needed for longer-term analysis, auditing, and archival is stored in the disk-based RDBMS.
- A *data utility* for accelerating performance-critical points in an architecture. For example, providing persistence and transactional capabilities to a message queuing system might be achieved by using TimesTen as the repository for the messages.
- A *data integration point* for multiple data sources on top of which new applications can be built. For example, an organization may have large amounts of information stored in several data sources, but only subsets of this information may be relevant to running its daily business. A suitable architecture would be to pull the relevant information from the different data sources into one TimesTen operational database to provide a central repository for the data of immediate interest to the applications.
- A *read-only cache*. Oracle data can be cached in a TimesTen read-only cache group. TimesTen read-only cache groups are automatically refreshed when Oracle tables are updated. A read-only cache group provides fast access to reference data such as look-up tables and subscriber profiles.
- An *updatable cache*. Oracle data can be cached in TimesTen updatable cache groups. Transactions on the TimesTen cache groups can be committed synchronously or asynchronously to the associated Oracle tables.
- A *distributed cache*. Oracle data can be cached in multiple instances of TimesTen running on different machines to provide scalability. You can configure a dynamic distributed cache in which records are loaded automatically and aged automatically.

TimesTen application scenarios

This section describes some application scenarios to illustrate how TimesTen might be integrated as part of an overall data management solution.

The application scenarios are:

- “Scenario 1: Caller usage metering application”: Uses TimesTen to store metering data on the activities of cellular callers. The metering data is collected from multiple TimesTen nodes distributed throughout a service area and archived in a central disk-based database for use by a central billing application.
- “Scenario 2: Real-time quote service application”: Uses TimesTen to store stock quotes from a data feed for access by program trading applications. Quote data is collected from the data feed and published on a real-time message bus. The data is read from the message bus and stored in TimesTen, where it is accessed by the program trading applications.
- “Scenario 3: Call center application”: Uses TimesTen as an application-tier cache to hold customer profiles maintained in a central Oracle database.

Scenario 1: Caller usage metering application

NoWires Communications, a cellular service provider, has a *usage metering* application that keeps track of the duration of each cellular call and the services used. For example, if a caller makes a regular call, a base rate is applied for the duration of the call. If a caller uses special features, such as roaming, Web browsing, or 3-way calling, extra charges are applied.

The usage metering application must efficiently monitor up to 100,000 concurrent calls, gather usage data on each call, and store the data in a central database for use by other applications that generate bills, reports, audits, and so on.

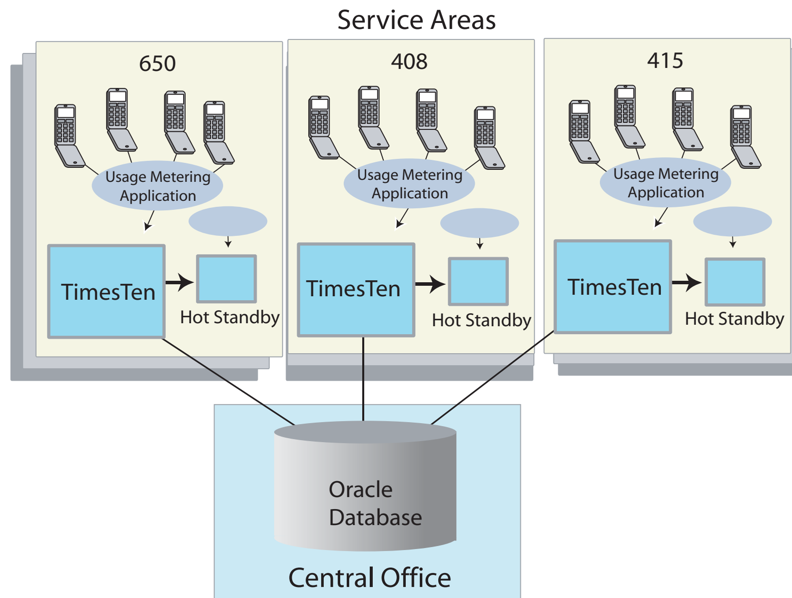
In the current configuration, all of the application data is stored in a *Calls* table in a central disk-based RDBMS. However, volume has been increasing and the company’s infrastructure needs modernization. The company has determined that scaling the central RDBMS would result in too many connections and too many records in the *Calls* table for acceptable performance. In addition, the RDBMS database is not capable of real time updates, which are needed by the usage metering application for maximum performance.

The company determines that performance and scaling could be improved by using TimesTen to store the caller data that is of immediate interest to the usage metering application and to warehouse all of the other data in the central RDBMS. The company’s solution is to decentralize the data gathering operation by distributing multiple instances of the usage metering application and TimesTen on individual nodes throughout its service areas. For maximum

performance, each usage metering application connects to its local TimesTen database by means of an ODBC direct driver connection.

Figure 2.1shows the decentralized, scalable configuration.

Figure 2.1 Distributed collection of usage data



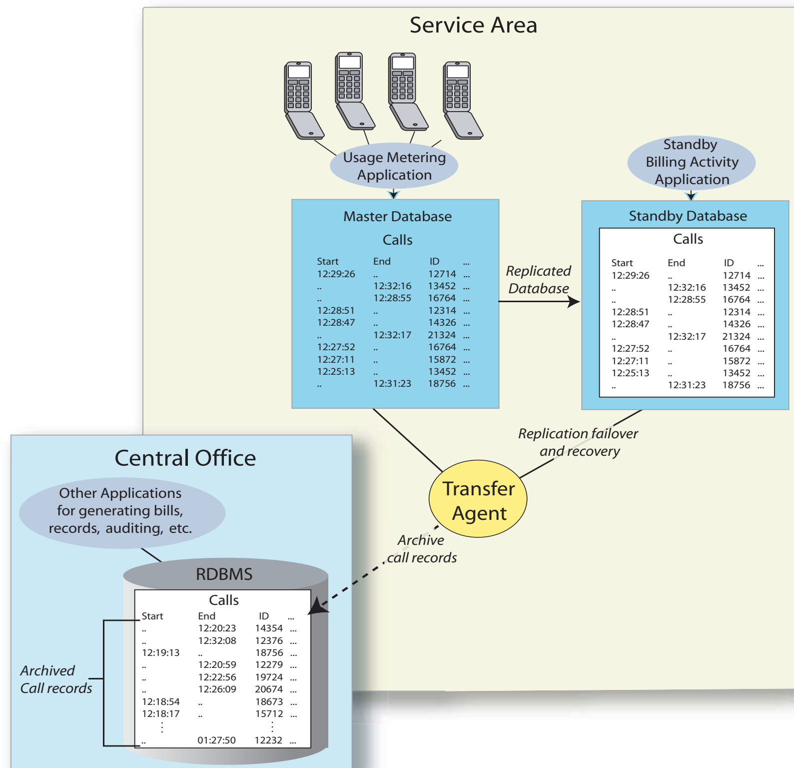
A separate usage metering application and TimesTen node combination is deployed to handle the real-time processing for calls beginning and terminating at different geographical locations delineated by area code. For each call, the local node stores a separate record for the beginning and the termination of a call. This is because the beginning of a cellular call might be detected by one node and its termination by another node.

In this scenario, TimesTen gains a performance edge over the central RDBMS database because it can manage millions of rows per table with relatively small performance degradation. So each TimesTen node scales well “vertically,” and the overall environment scales “horizontally” by adding more TimesTen nodes.

Revenue impacting transactions (inserts/updates) against TimesTen must be durable, so logging is used. To ensure data availability, each TimesTen node is replicated to a hot standby node.

Figure 2.2 shows that the company implements a special background component, called a *Transfer Agent*, that runs on each TimesTen node.

Figure 2.2 Transfer Agent archiving call records



The Transfer Agent connects locally to TimesTen by the ODBC direct driver and to the remote RDBMS database by IPC. The purpose of the Transfer Agent is to periodically archive the TimesTen records to the central RDBMS database, as well as to handle replication failover and recovery and update statistical information. In the event of a failure of the main node, the Transfer Agent shifts the callers to a redundant usage metering application on the TimesTen standby node.

Each time a customer makes, receives, or terminates a cellular call, the application inserts a record of the activity into its TimesTen *Calls* table. Each call record includes a timestamp, unique identifier, originating host's IP address, and information on the services used.

The Transfer Agent periodically selects the first 5000 rows in the *Calls* table and uses fast batch operations to archive them to the central RDBMS. (The result set of the SQL SELECT statement is limited to a 'first N' rows value to ensure low impact on the real-time processing of calls and to keep the data being archived to a manageable size.) After the Transfer Agent confirms successful archival of the

call records in the central RDBMS, it deletes them from TimesTen. The interval at which the Transfer Agent archives calls changes dynamically, based on the call load and the current size of the database.

Scenario 2: Real-time quote service application

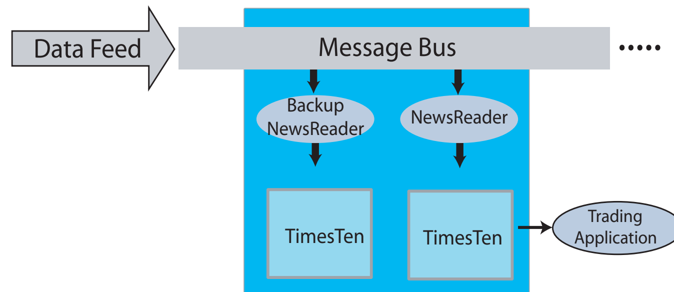
Beeman&Shuster, a financial services company, is adding a real-time quote and news service to their online trading facility. The immediate requirement of the real-time quote service is to read an incoming news wire from one of the major market data vendors and to make a select subset of the data available to trading applications that manage the automated trading operations for the company. The company's plan is to build an infrastructure that is extensible and flexible enough to accommodate future expansion to provide real-time quotes, news, and other trading services to retail subscribers.

The real-time quote service includes a *NewsReader* process that reads incoming data from a real-time message bus that is constantly fed data from a news wire. Each NewsReader is paired with a backup NewsReader that independently reads the data from the bus and inserts it into a separate TimesTen database. In this way, the message bus is used to fork incoming data to two TimesTen databases for redundancy. In this scenario, forking the data from the message bus is more efficient than using TimesTen replication.

Of each pair, one NewsReader makes the stock data available to a trading application, while the other serves as a hot standby backup to be used by the application in the event of a failure. The current load requires four NewsReader pairs, but more NewsReader pairs can be added in the future to scale the service to deliver real-time quotes to other types of clients over the Web, pager, cellular phone, and so on.

Figure 2.3 shows the configuration for capturing data from a message bus and feeding it to NewsReaders.

Figure 2.3 Capturing feed data from a message bus



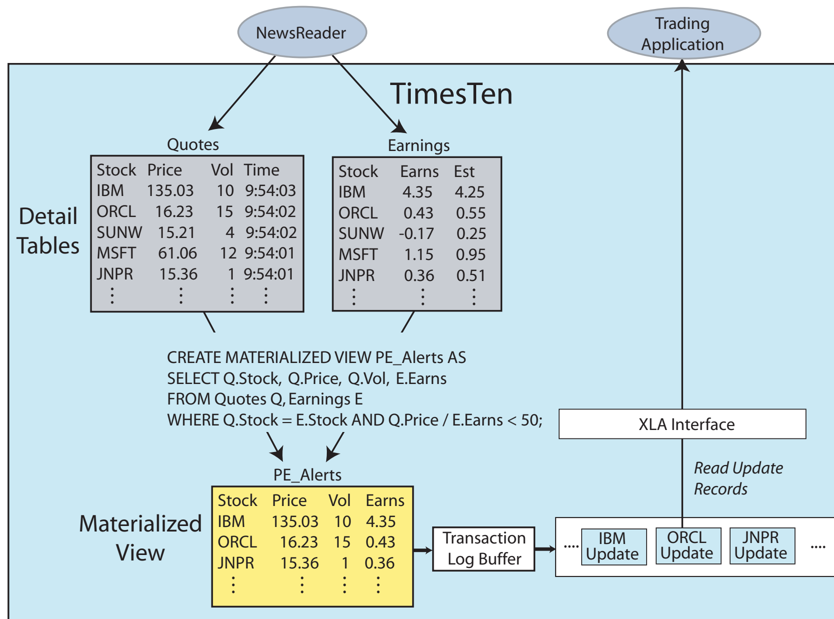
As shown in Figure 2.4, the NewsReader updates stock price data in a *Quotes* table in the TimesTen database. Less dynamic earnings data is updated in an

Earnings table. The *Stock* columns in the *Quotes* and *Earnings* tables are linked through a foreign key relationship.

The purpose of the trading application is to track only those stocks with PE ratios below 50, then use some internal logic to analyze the current stock price and trading volume to determine whether to place a trade using another part of the trading facility. For maximum performance, the trading application implements an event facility that uses the TimesTen Transaction Log API (XLA) to monitor the TimesTen transaction log for updates to the stocks of interest.

To provide the fastest possible access to such updates, the company creates a materialized view, named *PE_Alerts*, with a WHERE clause that calculates the PE ratio from the *Price* column in the *Quotes* table and the *Earns* column in the *Earnings* table. By using the XLA event facility to monitor the transaction log for price updates in the materialized view, the trading application only receives alerts for those stocks that meet its trading criteria.

Figure 2.4 Using materialized views and XLA

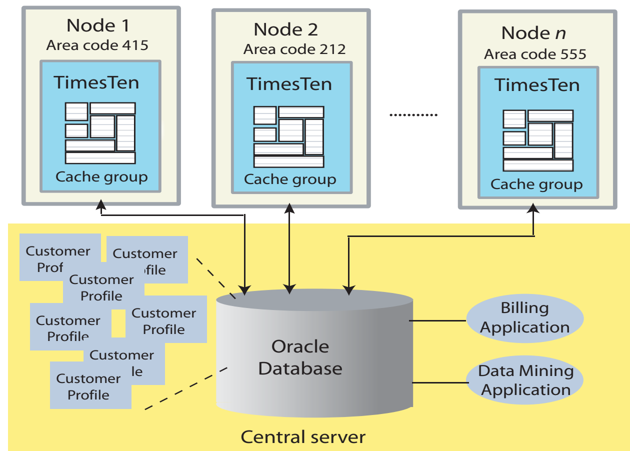


Scenario 3: Call center application

Advance Call Center provides customer service for *NoWires Communications*.

Figure 2.5 shows that the call center system includes a central server that hosts back-end applications and an Oracle database that stores the customer profiles.

Figure 2.5 Using Cache Connect to cache Oracle data



To manage a large volume of concurrent customer sessions, the call center designates application server nodes for specific U.S. area codes or groups of area codes. Each node contains a TimesTen database. When a customer contacts the call center, the appropriate customer profile is transparently loaded from the Oracle database into a cache group in the TimesTen database, using the TimesTen "Cache Connect to Oracle" option.

When a customer completes a call, changes to the customer profile are flushed from TimesTen to Oracle. Least recently used (LRU) aging is configured to remove inactive customer profiles from TimesTen.

All of the customer data is stored in the Oracle database. The Oracle database is much larger than each TimesTen database and is best accessed by applications that do not require the real-time performance of TimesTen but do require access to large amounts of data. Such applications include a *billing application* and a *data mining application.*

3

TimesTen Components

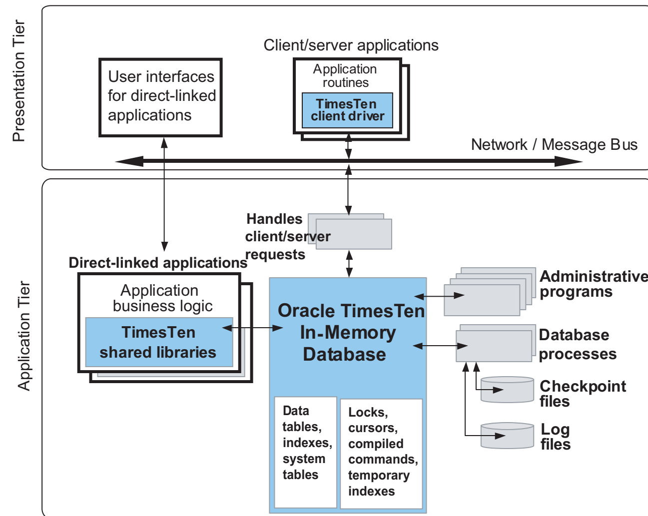
This chapter includes the following topics:

- TimesTen basic architecture
- Replication architecture
- Cache Connect architecture
- TimesTen connection options
- TimesTen APIs

TimesTen basic architecture

This section describes the system components of the Oracle TimesTen In-Memory Database from the perspective of installation, operation, and computing resources. Figure 3.1 shows the TimesTen components.

Figure 3.1 TimesTen components



TimesTen products consist of combinations of the following components:

- Shared libraries
- Memory-resident data structures
- Database processes
- Administrative programs
- Checkpoint and log files

Shared libraries

The routines that implement the TimesTen functionality are embodied in a set of shared libraries that developers link with their applications and execute as a part of the application's process. This shared library approach is in contrast to a more conventional RDBMS, which is implemented as a collection of executable programs to which applications connect, typically over a client/server network. Applications can also use a client/server connection to access an Oracle TimesTen database, though in most cases the best performance will be realized with a directly linked application. See "TimesTen connection options" on page 31.

Memory-resident data structures

The TimesTen database resides entirely in main memory at runtime. It is maintained in shared memory segments in the operating system and contains all user data, indexes, system catalogs, log buffers, lock tables and temp space. Multiple applications can share one database, and a single application can access multiple databases on the same system.

Database processes

TimesTen assigns a separate process to each database to perform operations including the following tasks:

- Loading the database into memory from a checkpoint file on disk
- Recovering the database if it needs to be recovered after loading it into memory
- Performing periodic checkpoints in the background against the active database
- Detecting and handling deadlocks
- Performing data aging

Administrative programs

Utility programs are explicitly invoked by users, scripts, or applications to perform services such as interactive SQL, bulk copy, backup and restore, database migration and system monitoring.

Checkpoint and log files

Checkpoint files contain an image of the database on disk. TimesTen uses dual checkpoint files for additional safety, in case the system fails while a checkpoint operation is in progress. Changes to databases are captured in transaction logs that are written to disk periodically. If a database needs to be recovered, TimesTen merges the database checkpoint on disk with the completed transactions that are still in the log files. Normal disk file systems are used for checkpoints and log files.

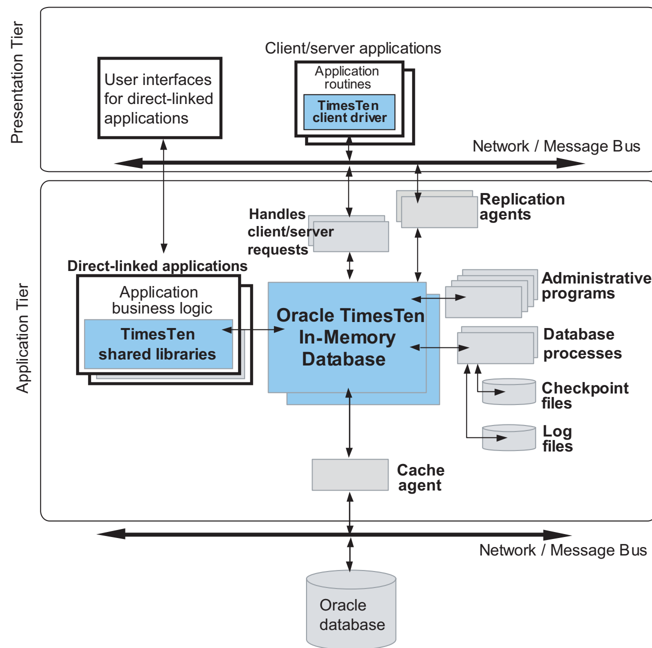
See Chapter 6, “Data Availability and Integrity” for more information.

Replication architecture

The TimesTen replication feature enables you to achieve near-continuous availability or workload distribution by sending updates between two or more servers. A master server is configured to send updates and a subscriber server is configured to receive them. A server can be both a master and a subscriber in a bidirectional replication scheme. Time-based conflict detection and resolution are used to establish precedence in case the same data is updated in multiple locations at the same time.

Figure 3.2 shows the basic TimesTen architecture with the addition of the replication and Cache Connect features.

Figure 3.2 TimesTen architecture including replication and Cache Connect



When replication is configured, a *replication agent* is started for each database. If multiple databases on the same server are configured for replication, each database has a separate replication agent. Each replication agent has the ability to send updates to one or more subscribers and to receive updates from one or more masters. Each of these connections is implemented as a separate thread of execution inside the replication agent process. Replication agents communicate through TCP/IP stream sockets.

For maximum performance, the replication agent detects updates to a database by monitoring the existing transaction log. It sends updates to the subscribers in

batches, if possible. Only committed transactions are replicated. On the subscriber node, the replication agent updates the database through an efficient low-level interface, avoiding the overhead of the SQL layer.

See “Replication” on page 51 for more information.

Cache Connect architecture

When TimesTen is used to cache portions of an Oracle database in a TimesTen in-memory database, a *cache group* is created to hold the cached data. The *cache agent* performs all asynchronous data transfers between the cache and Oracle database.

A cache group is a collection of one or more tables arranged in a logical hierarchy by using primary key and foreign key relationships. Each table in a cache group is related to an Oracle database table. A cache group table can contain all or a subset of the rows and columns in the related Oracle table. Cache groups can be created and modified by using the browser-based cache administrator or by using SQL statements. Cache groups support the following features:

- Applications can read from and write to cache groups.
- Cache groups can be refreshed from Oracle data automatically or manually.
- Updates to cache groups can be propagated to Oracle tables automatically or manually.
- Changes to either Oracle tables or the cache group can be tracked automatically.

When rows in a cache group are updated by applications, the corresponding rows in Oracle tables can be updated synchronously as part of the same transaction or asynchronously immediately afterward depending on the type of cache group that was created. The asynchronous configuration produces significantly higher throughput and much faster application response times.

Changes that originate in the Oracle tables are refreshed into the cache by the cache agent. The cache agent also performs user-defined aging of data out of the cache.

See Chapter 8, “Cache Connect to Oracle” for more information.

TimesTen connection options

Applications can connect to a TimesTen database in one of the following ways:

- Direct driver connection
- Client/server connection
- Driver manager connection

Direct driver connection

In a traditional database system, TCP/IP or another IPC mechanism is used by client applications to communicate with a database server process. All exchanges between client and server are sent over a TCP/IP connection. This IPC overhead adds substantial cost to all SQL operations and can be avoided in TimesTen by connecting the application directly to the TimesTen ODBC *direct driver.*

The ODBC direct driver is a library of ODBC and TimesTen routines that implement the database engine used to manage the databases. Java applications access the ODBC direct driver through the JDBC library.

An application can create a direct driver connection when it runs on the same machine as the TimesTen database. In a direct driver connection, the ODBC driver directly loads the TimesTen database into either the application's heap space or a shared memory segment. The application then uses the direct driver to access the memory image of the database. Because no inter-process communication (IPC) of any kind is required, a direct-driver connection provides extremely fast performance and is the preferred way for applications to access TimesTen databases.

Client/server connection

The TimesTen *client driver* and *server daemon* processes accommodate connections from remote client machines to databases across a network. The server daemon spawns a separate *server child process* for each client connection to the database.

Applications on a client machine issue ODBC calls to a local ODBC *client driver* that communicates with a server child process on the TimesTen server machine. The server child process, in turn, issues native ODBC requests to the ODBC direct driver to access the TimesTen database.

If the client and server reside on separate nodes in a network, they communicate by using sockets and TCP/IP. If both the client and server reside on the same machine, they can communicate more efficiently by means of a shared memory segment as IPC.

Traditional database systems are typically structured in this client/server model, even when the application and the database are on the same system. Client/server communication adds extra cost to all database operations.

Driver manager connection

Applications can connect to TimesTen through an ODBC *driver manager*, which is a database-independent interface that adds a layer of abstraction between the applications and the TimesTen database. In this way, the driver manager allows applications to be written to operate independently of the database and to use interfaces that are not directly supported by TimesTen. The use of a driver manager also enables a single process to have both direct and client connections to the database.

On Microsoft Windows systems, applications can connect to the MS ODBC driver manager to make use of a TimesTen database along with data sources from other vendors, such as Oracle. Driver managers for UNIX systems are available as open-source software as well as from third-party vendors.

TimesTen APIs

The runtime architecture of TimesTen supports connectivity through the ODBC and JDBC APIs, which allow applications to access data using SQL. TimesTen also provides built-in procedures and utilities that extend ODBC and JDBC functionality for TimesTen-specific operations.

TimesTen provides the following additional APIs:

- Transaction Log API
- Distributed Transaction Processing APIs
- TTClasses

Transaction Log API

The Transaction Log API (XLA) allows applications to detect changes made to specified tables in a local database. XLA also provides functions that can be used by applications to apply changes detected in one database to another database. See "Transaction Log API" on page 61 for more information.

Distributed Transaction Processing APIs

TimesTen implements the X/Open XA Specification and its Java derivative, the Java Transaction API (JTA).

The TimesTen implementation of the XA interfaces is intended for use by transaction managers in distributed transaction processing (DTP) environments. These interfaces can be used to write a new transaction manager or to adapt an existing transaction manager to operate with TimesTen resource managers.

The TimesTen implementation of the JTA interfaces is intended to enable Java applications, application servers, and transaction managers to use TimesTen resource managers in DTP environments.

TTClasses

TimesTen C++ Interface Classes (TTClasses) is easier to use than ODBC while maintaining fast performance. This C++ class library provides wrappers around the most common ODBC functionality. The TTClasses library is also intended to promote best practices when writing application software.

For more information

For more information about the TimesTen database, see Chapter 1, "Working with TimesTen Data Stores" in *Oracle TimesTen In-Memory Database C Developer's and Reference Guide* and "Chapter 6, "Working with Data in a TimesTen Data Store" in *Oracle TimesTen In-Memory Database Operations Guide*.

For more information about connecting to databases, see Chapter 1, "Creating TimesTen Data Stores" and Chapter 2, "Working with the TimesTen Client and Server" in *Oracle TimesTen In-Memory Database Operations Guide*.

For more information about TimesTen APIs, see:

- *Oracle TimesTen In-Memory Database C Developer's and Reference Guide*
- *Oracle TimesTen In-Memory Database Java Developer's and Reference Guide*
- *Oracle TimesTen In-Memory Database API Reference Guide*
- *Oracle TimesTen In-Memory Database SQL Reference Guide*
- *Oracle TimesTen In-Memory Database TTClasses Guide*

4

Concurrent Operations

A database can be accessed in shared mode. When a shared database is accessed by multiple transactions, there must be a way to coordinate concurrent changes to data with scans of the same data in the database.TimesTen uses transaction isolation and locks to coordinate concurrent access to data.

This chapter includes the following topics:

- Transaction isolation
- Locks

Transaction isolation

Transaction isolation provides an application with the appearance that the system performs one transaction at a time, even though there are concurrent connections to the database. Applications can use the **Isolation** connection attribute to set the isolation level for a connection. Concurrent connections can use different isolation levels.

Isolation level and concurrency are inversely related. A lower isolation level enables greater concurrency, but with greater risk of data inconsistencies. A higher isolation level provides a higher degree of data consistency, but at the expense of concurrency.

TimesTen has two isolation levels:

- Read committed isolation
- Serializable isolation

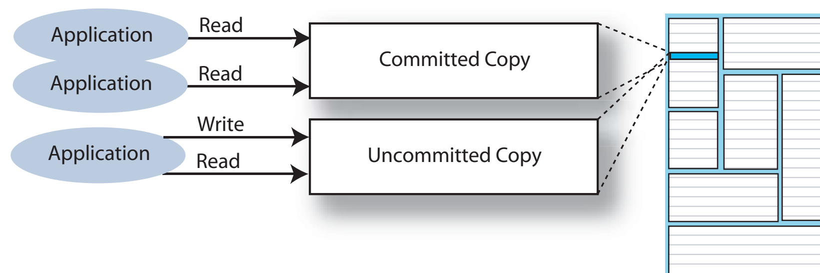
Read committed isolation

When an application uses read committed isolation, readers use a separate copy of the data from writers, so read locks are not needed. Read committed isolation is non-blocking for queries and can work with Serializable isolation or read committed isolation. Under read committed isolation, writers block only other writers and readers using serializable isolation; writers do not block readers using read committed isolation. Read committed isolation is the default isolation level.

TimesTen uses *versioning* to implement read committed isolation. TimesTen update operations create new copies of the rows they update to allow nonserializable reads of those rows to proceed without waiting.

Figure 4.1 shows that applications use a committed copy of the data to read while another application writes and reads on an uncommitted copy.

Figure 4.1 Read committed isolation



Read committed isolation provides increased concurrency because readers do not block writers and writers do not block readers. This isolation level is useful for applications that have long-running scans that may conflict with other operations

needing access to a scanned row. However, the disadvantage when using this isolation level is that non-repeatable reads are possible within a transaction or even a single statement (for example, the inner loop of a nested join).

When using this isolation level, DDL statements that operate on a table can block readers and writers of that table. For example, an application cannot read a row from a table if another application has an uncommitted DROP TABLE, CREATE INDEX, or ALTER TABLE operation on that table. In addition, blocking checkpoints will block readers and writers.

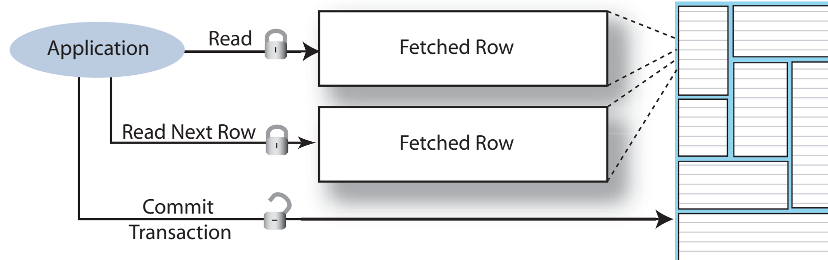
Read committed isolation does acquire read locks as needed during materialized view maintenance to ensure that views are consistent with their detail tables. These locks are not held until the end of the transaction but are instead released when maintenance has been completed.

Serializable isolation

When an application uses serializable isolation, locks are acquired within a transaction and are held until the transaction commits or rolls back for both reads and writes. This level of isolation provides for repeatable reads and increased isolation within a transaction at the expense of decreased concurrency. Transactions use serializable isolation when database-level locking is chosen.

Figure 4.2 shows that locks are held until the transaction is committed.

Figure 4.2 Serializable isolation



Serializable isolation level is useful for transactions that require the strongest level of isolation. Concurrent applications that need to modify the data that is read by a transaction may encounter lock timeouts because read locks are held until the transaction commits.

Locks

Locks are used to serialize access to resources to prevent one user from changing an element that is being read or changed by another user. TimesTen automatically performs locking if a database is accessed in shared mode.

Serializable transactions acquire share locks on the items they read and exclusive locks on the items they write. These locks are held until the transaction commits or rolls back. Read-committed transactions acquire exclusive locks on the items they write and hold these locks until the transactions are committed. Read-committed transactions do not acquire locks on the items they read. Committing or rolling back a transaction closes all cursors and releases all locks held by the transaction.

TimesTen performs deadlock detection to report and eliminate deadlock situations. If an application is denied a lock because of a deadlock error, it should roll back the entire transaction and retry it.

Applications can select from three lock levels:

- Database-level locking
- Table-level locking
- Row-level locking

Database-level locking

Locking at the database level locks an entire database when it is accessed by a transaction. All database-level locks are exclusive. A transaction that requires a database-level lock cannot start until there are no active transactions on the database. After a transaction has obtained a database-level lock, all other transactions are blocked until the transaction commits or rolls back.

Database-level locking restricts concurrency more than table-level locking and is useful only for initialization operations such as bulkloading, when no concurrency is necessary. Database-level locking has better response time than row-level or table-level locking at the cost of diminished concurrency and diminished throughput.

Different transactions can coexist with different levels of locking, but the presence of even one transaction that uses database-level locking leads to reduced concurrency.

Use the **LockLevel** connection attribute or the **ttLockLevel** built-in procedure to implement database-level locking.

Table-level locking

Table-level locking locks a table when it is accessed by a transaction. It is useful when a statement accesses most of the rows in a table. Applications can call the

ttOptSetFlag built-in procedure to request that the optimizer use table locks. The optimizer determines when a table lock should be used.

Table locks can reduce throughput, so they should be used only when a substantial portion of the table needs to be locked or when high concurrency is not needed.For example, tables can be locked for operations such as bulk updates. In read-committed isolation, TimesTen does not use table-level locking for read operations unless it is explicitly requested by the application.

Row-level locking

Row-level locking locks only the rows that are accessed by a transaction. It provides the best concurrency by allowing concurrent transactions to access rows in the same table. Row-level locking is preferable when there are many concurrent transactions, each operating on different rows of the same tables.

By default, TimesTen enables row-level locking. Use the **LockLevel** connection attribute or the **ttLockLevel** built-in procedure to set it explicitly. Use the **ttOptSetFlag** built-in procedure to set it for the optimizer to use for a specific transaction.

For more information

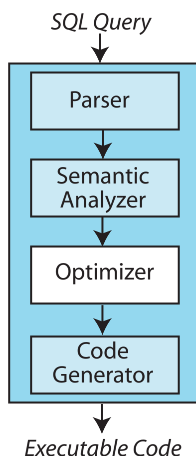
For more information about locks and transaction isolation, see Chapter 7, "Transaction Management and Recovery" in *Oracle TimesTen In-Memory Database Operations Guide*.

5

Query Optimization

TimesTen has a cost-based query optimizer that speeds up data access by automatically searching for the best way to answer queries. Optimization is performed in the third stage of the compilation process. The four stages of compilation are shown in Figure 5.1.

Figure 5.1 Compilation stages



The role of the optimizer is to determine the lowest cost plan for executing queries. By "lowest cost plan" we mean an access path to the data that will take the least amount of time. The optimizer determines the cost of a plan based on:

- Table and column statistics
- Metadata information (such as referential integrity, primary key)
- Index choices (including automatic creation of temporary indexes)
- Scan methods (full table scan, Rowid lookup, T-tree or hash index scan)
- Join algorithm choice (nested loop joins, nested loop joins with indexes, or merge join)

This chapter includes the following topics:

- Optimization time and memory usage
- Statistics
- Optimizer hints

- Indexing techniques
- Scan methods
- Join methods
- Optimizer plan

Optimization time and memory usage

The TimesTen optimizer is designed to generate the best possible plan within reasonable time and memory constraints. No optimizer always chooses the optimal plan for every query. Instead, the goal of the TimesTen optimizer is to choose the best plan from among a set of plans generated by using strategies for finding the most promising areas within the search-space of plans. Because optimization usually happens only once for each query while the query itself may be executed many times, the optimizer is designed to give precedence to execution time over optimization time.

The plans generated by the optimizer emphasize performance over memory usage. The optimizer may designate the use of significant amounts of temporary memory space in order to speed up execution time. In memory-constrained environments, applications can use the optimizer hints described in "Optimizer hints" on page 42 to disable the use of temporary indexes and tables in order to create plans that trade maximum performance for less memory usage.

Statistics

When determining the execution path for a query, the optimizer examines statistics about the data referenced by the query, such as the number of rows in the tables, the minimum and maximum values and the number of unique values in interval statistics of columns used in predicates, the existence of primary keys within a table, the size and configuration of any existing indexes. These statistics are stored in the SYS.TBL_STATS and SYS.COL_STATS tables, which are populated by TimesTen when an applications calls the **ttOptUpdateStats** built-in procedure.

The optimizer uses the statistics for each table to calculate the *selectivity* of predicates, such as T1.A = 4, or a combination of predicates, such as T1.A = 4 AND T1.B < 10. *Selectivity* is an estimate of the number of rows in a table. If a predicate selects a small percentage of rows, it is said to have *high* selectivity, while a predicate that selects a large percentage of rows has *low* selectivity.

Optimizer hints

The optimizer allows applications to provide *hints* to adjust the way that plans are generated. For example, applications can use the **ttOptSetFlag** procedure to provide the TimesTen optimizer with hints about how to best optimize any

particular query. This takes the form of directives that restrict the use of particular join algorithms, use of temporary indexes and types of index (T-tree or hash), use of locks, whether to optimize for all the rows or only the first '*n*' number of rows in a table, and whether to materialize intermediate results. The existing hints set for a database can be viewed using the **ttOptGetFlag** procedure.

Applications can also use the **ttOptSetOrder** procedure to specify the order in which tables are to be joined by the optimizer, as well as the **ttOptUseIndex** procedure to specify which indexes should be considered for each correlation in a query.

Indexing techniques

The TimesTen query optimizer uses indexes to speed up the execution of a query. The optimizer uses existing indexes or creates temporary indexes to generate an execution plan when preparing a SELECT, INSERT SELECT, UPDATE, or DELETE statement.

An index is a map of keys to row locations in a table. Strategic use of indexes is essential in order to obtain maximum performance from a TimesTen system.

TimesTen uses two types of indexes:

- T-tree indexes
- Hash indexes

T-tree indexes

Main memory data management systems are designed to reduce memory requirements, as well as to shrink code path and footprint. A typical disk-based RDBMS uses B-tree indexes to reduce the amount of disk I/O needed to accomplish indexed look-up of data files. TimesTen uses *T-tree* indexes, which are optimized for main memory access. T-tree indexes are more economical than B-trees because they require less memory space and fewer CPU cycles.

TimesTen creates a T-tree index automatically:

- When a primary key is defined on a table and the UNIQUE HASH ON clause is *not* specified
- To enforce unique column constraints and foreign key constraints

T-tree indexes can also be created with the CREATE INDEX statement.

T-tree indexes are used for lookups involving equality and inequality ranges such as "greater than/equal to". T-tree indexes can be designated as unique or not unique. Multiple NULL values are permitted in a unique T-tree index.

Hash indexes

Hash indexes are created for tables with a primary key when the UNIQUE HASH INDEX clause is specified in the CREATE TABLE statement. There can be only one hash index for each table.

In general, hash indexes are faster than T-tree indexes for exact match lookups and equijoins. However, hash indexes cannot be used for lookups involving ranges or the prefix of a key and can require more space than T-tree indexes.

Scan methods

The optimizer can select from multiple types of scan methods. The most common scan methods are:

- Full table scan
- Rowid lookup
- T-tree index scan (on either a permanent or temporary index)
- Hash index lookup (on either a permanent or temporary index)

TimesTen performs high-performance exact matches through hash indexes and rowid lookups, and performs range matches through T-tree indexes. The **ttOptSetFlag** built-in procedure can be used to allow or disallow the optimizer from considering certain scan methods when choosing a query plan.

A *full table scan* examines every row in a table. Because it is the least efficient way to evaluate a query predicate, a full scan is only used when no other method is available.

TimesTen assigns a unique ID, called ROWID, to each row stored in a table. A *Rowid lookup* is applicable if, for example, an application has previously selected a ROWID and then uses a 'WHERE ROWID =' clause to fetch that same row. Rowid lookups are faster than both types of index scans.

A *hash index lookup* involves using a hash index to find rows based on their primary keys. Such lookups are applicable if the table has a primary key that has a hash index and the predicate specifies an exact match over the primary key columns.

A *T-tree index scan* involves using a T-tree index to access a table. Such a scan is applicable as long as the table has one or more T-tree indexes. The optimizer attempts to match as long a prefix of the key columns as possible. If no match is found for a particular key column, then the index scan returns all of the values in that column. If the predicate specifies a single value (such as T1.A = 2) for each key column and if the T-tree index is unique, the optimizer locates the row by means of a lookup, rather than a scan.

Join methods

The optimizer can select from multiple types of join methods. When the rows from two tables are joined, one table is designated the *outer table* and the other the *inner table*. During a join, the optimizer scans the rows in the outer and inner tables to locate the rows that match the join condition.

The optimizer analyzes the statistics for each table and, for example, might identify the smallest table or the table with the best selectivity for the query as outer table. If indexes exist for one or more of the tables to be joined, the optimizer takes them into account when selecting the outer and inner tables.

If more than two tables are to be joined, the optimizer analyzes the various combinations of joins on table pairs to determine which pair to join first, which table to join with the result of the join, and so on for the optimum sequence of joins.

The cost of a join is largely influenced by the method in which the inner and outer tables are accessed to locate the rows that match the join condition. The optimizer can select from two join methods:

- Nested loop join
- Merge join

Nested loop join

In a nested loop join with no indexes, a row in the outer table is selected one at a time and matched against every row in the inner table. All of the rows in the inner table are scanned as many times as the number of rows in the outer table. If the inner table has an index on the join column, that index is used to select the rows that meet the join condition. The rows from each table that satisfy the join condition are returned. Indexes may be created on the fly for inner tables in nested loops, and the results from inner scans may be materialized before the join.

Figure 5.2 shows an example of a nested loop join. The join condition is:

```
WHERE T1.A=T2.A
```

T1 is the outer table and T2 is the inner table. Values in column A in table T1 that match values in column A in table T2 are 1 and 7. The join results concatenate the rows from T1 and T2. For example, the first join result is the following row:

```
7 50 43.54 21 13.69
```

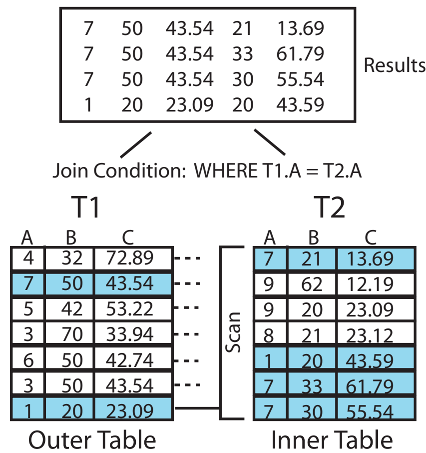
It concatenates a row from T1:

```
7 50 43.54
```

with the first row from T2 in which the values in column A match:

```
7 21 13.69
```

Figure 5.2 Nested loop join



Merge join

A merge join is used only when the join columns are sorted by T-tree indexes. In a merge join, a cursor advances through each index one row at a time. Because the rows are already sorted on the join columns in each index, a simple formula is applied to efficiently advance the cursors through each row in a single scan. The formula looks something like:

- If Inner.JoinColumn < Outer.JoinColumn -- advance inner cursor
- If Inner.JoinColumn = Outer.JoinColumn -- read match
- If Inner.JoinColumn > Outer.JoinColumn -- advance outer cursor

Unlike a nested loop join, there is no need to scan the entire inner table for each row in the outer table. A merge join can be used when T-tree indexes have been created for the tables before preparing the query. If no T-tree indexes exist for the tables being joined before preparing the query, the optimizer may in some situations create temporary T-tree indexes in order to use a merge join.

Figure 5.3 shows an example of a merge join. The join condition is:

```
WHERE T1.A=T2.A
```

X1 is the index on table T1, sorting on column A. X2 is the index on table T2, sorting on column A. The merge join results concatenate the rows in X1 with rows in X2 in which the values in column A match. For example, the first merge join result is:

```
1 20 23.09 20 43.59
```

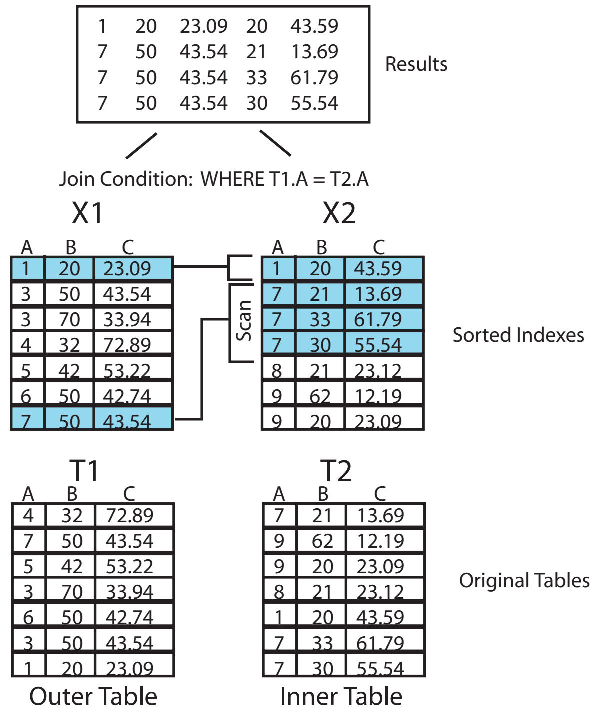
It concatenates a row in X1:

```
1 20 23.09
```

with the first row in X2 in which the values in column A match:

```
1 20 43.59
```

Figure 5.3 Merge join



Optimizer plan

Like most database optimizers, the TimesTen query optimizer stores the details on how to most efficiently perform SQL operations in an *execution plan*, which can be examined and customized by application developers and administrators.

The execution plan data is stored in the TimesTen SYS.PLAN table and includes information about which tables are to be accessed and in what order, which tables are to be joined, and which indexes are to be used. Users can direct the TimesTen query optimizer to enable or disable the creation of an execution plan in the SYS.PLAN table by setting the *GenPlan* optimizer flag in the **ttOptSetFlag** procedure.

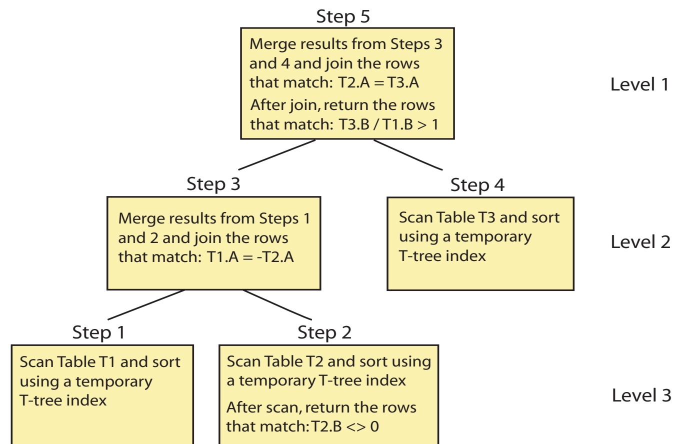
The execution plan designates a separate *step* for each database operation to be performed to execute the query. The steps in the plan are organized into *levels* that designate which steps must be completed to generate the results required by the step or steps at the next level.

For example, the optimizer prepares an execution plan for the following query:

```
SELECT COUNT(*)
FROM T1, T2, T3
WHERE T3.B/T1.B > 1
AND T2.B <> 0
AND T1.A = -T2.A
AND T2.A = T3.A;
```

In this example, the optimizer dissects the query into its individual operations and generates a 5-step execution plan to be performed at three levels, as shown in Figure 5.4.

Figure 5.4 Example execution plan



For more information

For more information about the query optimizer, see Chapter 9, "The TimesTen Query Optimizer" in *Oracle TimesTen In-Memory Database Operations Guide*.

For more information about indexing, see "Understanding indexes" and "Working with indexes" in Chapter 6, "Working with Data in a TimesTen Data Store" in *Oracle TimesTen In-Memory Database Operations Guide*.

Also see descriptions for the CREATE TABLE and CREATE INDEX statements in *Oracle TimesTen In-Memory Database SQL Reference Guide*.

6

Data Availability and Integrity

TimesTen ensures the availability, durability, and integrity of data through the following mechanisms:

- Logging
- Checkpointing
- Replication

Logging

The TimesTen log is used for the following purposes:

- Redo transactions if a system failure occurs
- Undo transactions that are rolled back
- Replicate changes to other TimesTen databases
- Replicate changes to an Oracle database
- Enable applications to monitor changes to tables through the XLA interface

By default, TimesTen keeps logs in an in-memory *transaction log buffer* and also stores them on disk.

Writing the log buffer to disk

When logging is enabled, applications can control when the data in the transaction log buffer is written to disk. TimesTen provides a **DurableCommits** connection attribute that specifies whether the log buffer is automatically written to disk at transaction commit or is deferred until the application calls the **ttDurableCommit** procedure.

Setting **DurableCommits** so that the log is not automatically posted to disk at transaction commit reduces the transaction execution time at the risk of losing the results of some committed transactions in the event of a failure. However, regardless of the setting of **DurableCommits**, the log is saved to disk when the log buffer in memory fills up.

ODBC provides an autocommit mode that forces a commit after each statement. By default, autocommit is enabled so that an implicit commit is issued immediately after a statement executes successfully. TimesTen recommends that you turn autocommit off so that commits are intentional.

When are log files deleted?

Log files are kept until TimesTen declares them to be *purgeable*. A log file cannot be purged until all of the following actions has been completed:

- All transactions writing log records to the log file (or a previous log file) have committed or rolled back
- All changes recorded in the log file have been written to the checkpoint files on disk
- All changes recorded in the log file have been replicated (if replication is used)
- All changes recorded in the log file have been propagated to Oracle (if Cache Connect is used)
- All changes recorded in log files have been reported to the XLA applications (if XLA is used)

Disabling logging

When logging is disabled, transactions cannot be rolled back and only database-level locking is allowed. Disable logging only for operations like bulk loads of tables and only if the following conditions are true:

- The operation can be restarted from the beginning if a failure occurs.
- The application performing the bulk load is the only application connected to the database.

Use the **Logging** connection attribute to disable logging.

Checkpointing

Checkpoints are used to keep a snapshot of the database. In the event of a system failure, TimesTen can use a checkpoint file together with log files to restore a database to its last transactionally consistent state.

Only the data that has changed since the last checkpoint operation is written to the checkpoint file. The checkpoint operation scans the database for blocks that have changed since the last checkpoint. It then updates the checkpoint file with the changes and removes any log files that are no longer needed.

TimesTen provides two kinds of checkpoints:

- Nonblocking checkpoints
- Blocking checkpoints

TimesTen creates nonblocking checkpoints automatically.

Nonblocking checkpoints

TimesTen initiates nonblocking checkpoints in the background automatically. Nonblocking checkpoints are also known as *fuzzy* checkpoints. The frequency of these checkpoints can be adjusted by the application. Nonblocking checkpoints do not require any locks on the database, so multiple applications can asynchronously commit or roll back transactions on the same database while the checkpoint operation is in progress.

Blocking checkpoints

An application can call the **ttCkptBlocking** procedure to initiate a blocking checkpoint in order to construct a transaction-consistent checkpoint. While a blocking checkpoint operation is in progress, any other new transactions are put in a queue behind the checkpointing transaction. If any transaction is long-running, it may cause many other transactions to be held up. No log is needed to recover from a blocking checkpoint because the checkpoint record contains the information needed to recover.

Recovery from log and checkpoint files

If a database becomes invalid or corrupted by a system or process failure, every connection to the database is invalidated. When an application reconnects to a failed database, the subdaemon allocates a new memory segment for the database and recovers its data from the checkpoint and log files.

During recovery, the latest checkpoint file is read into memory and all durably committed transactions are rolled forward from the appropriate log files. Uncommitted or rolled-back transactions are not recovered.

For applications that require uninterrupted access to TimesTen data in the event of failures, see "Replication" on page 51.

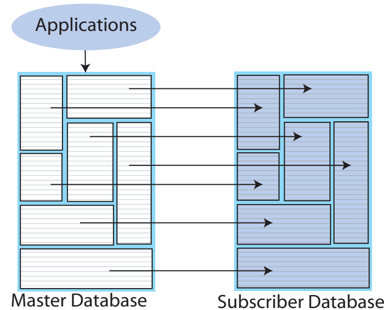
Replication

Replication is the process of copying data between databases. The fundamental motivation behind TimesTen replication is to make data highly available to mission-critical applications with minimal performance impact.

In addition to its role in failure recovery, replication is also useful for distributing user loads across multiple databases for maximum performance and for facilitating online upgrades and maintenance, as described in "Upgrading TimesTen" on page 79.

Figure 6.1 shows that TimesTen replication copies updates made in a *master* database to a *subscriber* database.

Figure 6.1 Master and subscriber databases



To enable high efficiency and low overhead, TimesTen uses replication scheme based on transaction log.

Replication at each master and subscriber database is controlled by *replication agents* that communicate through TCP/IP stream sockets. The replication agent on the master database reads the records from its transaction log. It forwards changes to replicated elements to the replication agent on the subscriber database. The replication agent on the subscriber then applies the updates to its database. If the subscriber agent is not running when the updates are forwarded by the master, the master retains the updates in its log until they can be applied at the subscriber.

Replication configurations

TimesTen replication can be configured in a variety of ways for the best balance between performance and availability.

Replication is configured through SQL statements. In general, replication can be configured to be unidirectional ("one way") from a master to one or more subscribers, or bidirectional ("two way") between two or more databases that serve as both master and subscriber. More than two databases configured for bidirectional replication is often referred to as "N-way" or "update anywhere" replication.

Figure 6.2 shows the following types of replication configurations:

- Unidirectional replication to one subscriber
- Unidirectional replicationto multiple subscribers
- Bidirectional replication
- N-way bidirectional replication

Figure 6.2 Replication configurations

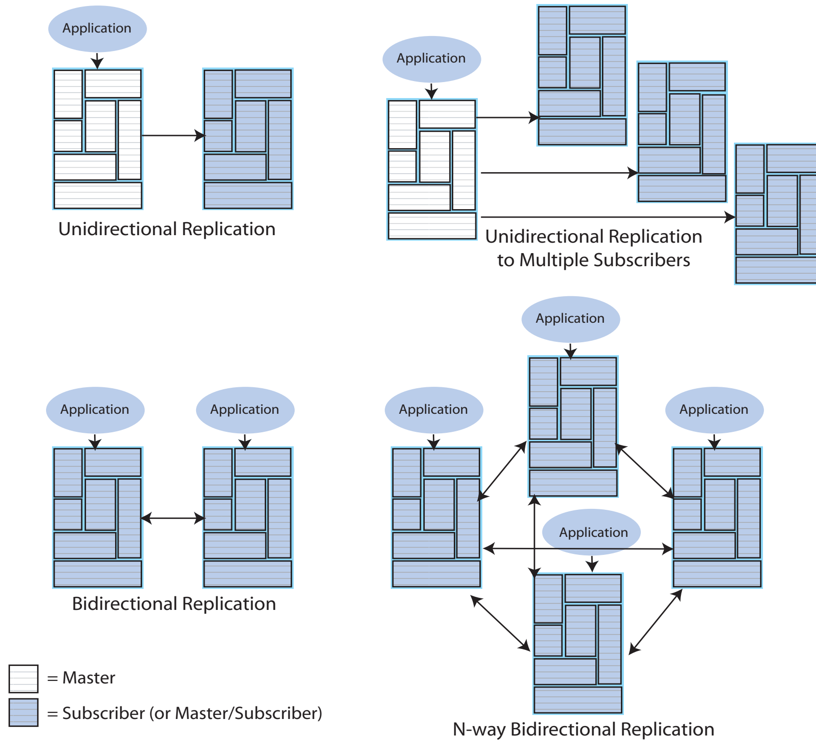
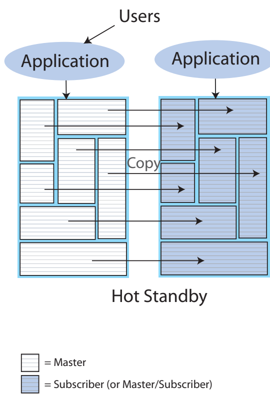


Figure 6.3 shows how replication can be configured to provide a hot standby backup database. A single master database is replicated to a subscriber database with appropriate failover mechanisms built into the application itself.

Figure 6.3 Hot standby configuration

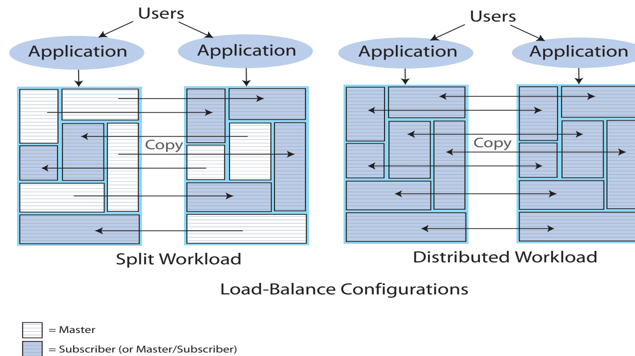


See “Scenario 1: Caller usage metering application” on page 19 for an example application that use hot standby databases.

Figure 6.4 shows two configurations for load balancing. The workload can be split between two bidirectionally replicated databases. There are two basic types of load-balancing configurations:

- *Split workload* where each database bidirectionally replicates a portion of its data to the other database.
- *Distributed workload* where user access is distributed across duplicate application/database combinations that bidirectionally replicate updates.

Figure 6.4 Load-balancing configurations

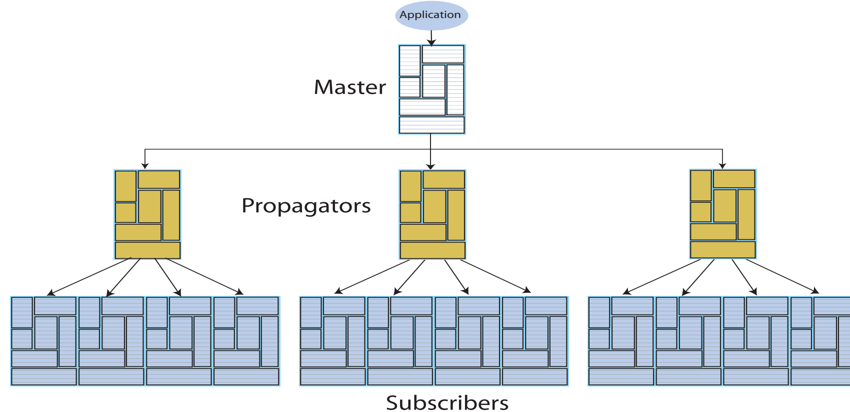


In a distributed workload configuration, the application has the responsibility to divide the work between the two systems so that replication collisions do not occur. If collisions do occur, TimesTen has a timestamp-based collision detection and resolution capability.

Replication can also be used to propagate updates from one or more databases to many databases. This can be useful when maintaining duplicate databases over lower-bandwidth connections and for distributing (or *fanning out*) replication loads in configurations in which a master database must replicate to a large number of subscribers.

Figure 6.5 shows a propagation configuration. One master propagates updates to three subscribers. The subscribers are also masters that propagate updates to additional subscribers.

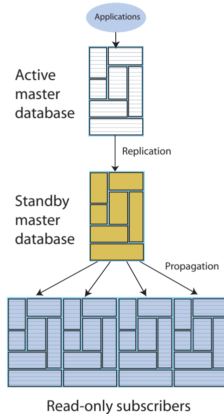
Figure 6.5 Propagation configuration



Active standby pair

Figure 6.6 shows an active standby pair.

Figure 6.6 Active standby pair



An active standby pair includes an active master database, a standby master database, subscriber databases, and the tables and cache groups that comprise the databases.

In an active standby pair, two databases are defined as masters. One is an active master database, and the other is a standby master database. The active master database is updated directly. The standby master database cannot be updated directly. It receives the updates from the active master database and propagates the changes to read-only subscriber databases. This arrangement ensures that the standby master database is always ahead of the subscriber databases and enables rapid failover to the standby database if the active master database fails.

Only one of the master databases can function as an active master database at a specific time. The active role is specified through the **ttRepStateSet** procedure. If the active master database fails, then the user can use the **ttRepStateSet** procedure to change the role of the standby master database to active before recovering the failed database as a standby database. The replication agent must be started on the new standby master database.

If the standby master database fails, the active master database can replicate changes directly to the subscribers. After the standby master database has recovered, it contacts the active standby database to receive any updates that have been sent to the subscribers while the standby was down or was recovering. When the active and the standby master databases have been synchronized, then the standby resumes propagating changes to the subscribers.

You can achieve high availability of cache instances by configuring an active standby pair for the following types of cache groups:

- A READONLY cache group with the AUTOREFRESH attribute
- An ASYNCHRONOUS WRITETHROUGH cache group

An active standby pair that replicates one of these types of cache groups can change the role of a cache group automatically as part of failover and recovery with minimal chance of data loss. See "Active standby pairs with cache groups" in *TimesTen to TimesTen Replication Guide*.

Asynchronous and return service replication

TimesTen replication is by default an asynchronous mechanism. When using asynchronous replication, an application updates the master database and continues working without waiting for the updates to be received by the subscribers. The master and subscriber databases have internal mechanisms to confirm that the updates have been successfully received and committed by the subscriber. These mechanisms ensure that updates are applied at a subscriber only once, but they are invisible to the application.

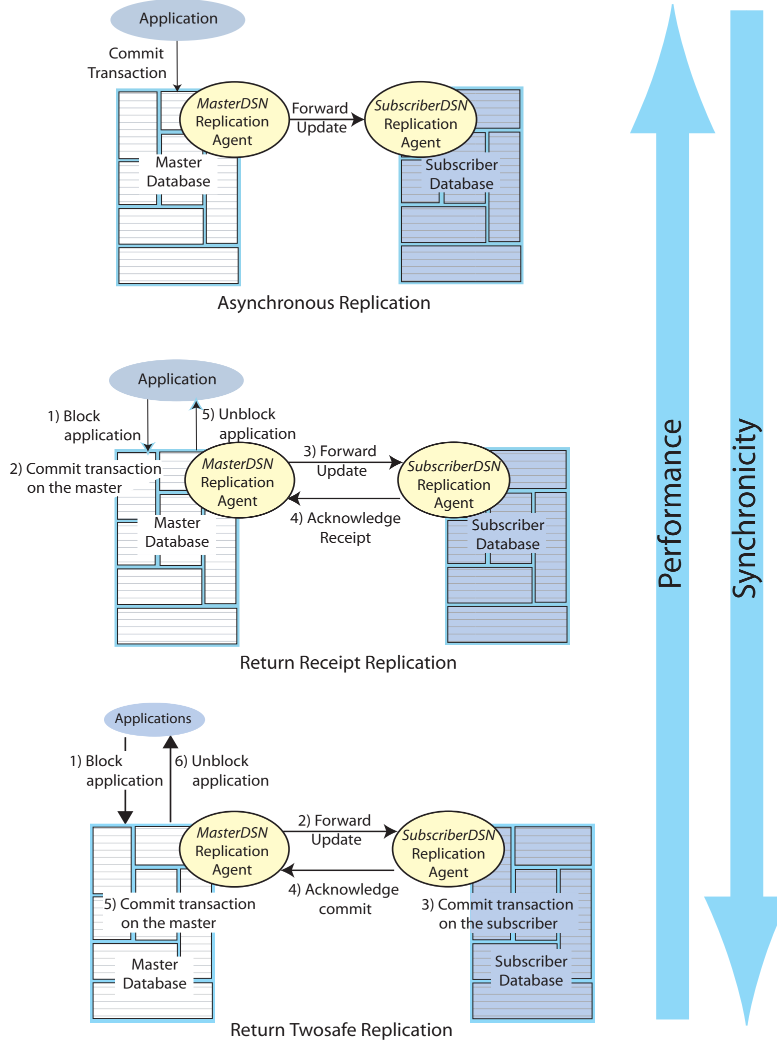
Asynchronous replication provides maximum performance, but the application is completely decoupled from the receipt process of the replicated elements on the subscriber. TimesTen also provides two *return service* options for applications that need higher levels of confidence that the replicated data is consistent between the master and subscriber databases:

- The *return receipt service* loosely couples or "synchronizes" the application with the replication mechanism by blocking the application until replication confirms that the update has been received by the subscriber replication agent.
- The *return twosafe service* enables fully synchronous replication by blocking the application until replication confirms that the update has been both received *and committed* on the subscriber.

Applications that use the return services trade some performance to ensure higher levels of data integrity and consistency between the master and subscriber databases. In the event of a master failure, the application has a higher degree of confidence that a transaction committed at the master persists in the subscribing database. Return receipt replication has less performance impact than return twosafe at the expense of less synchronization.

Figure 6.7 shows the tradeoff between performance and synchronicity among asynchronous replication, return receipt replication, and return twosafe replication.

Figure 6.7 Asynchronous and return service replication



Replication failover and recovery

In order for replication to make data highly available to applications with minimal performance impact, there must be a means to shift users from the failed database to its surviving backup as seamlessly as possible.

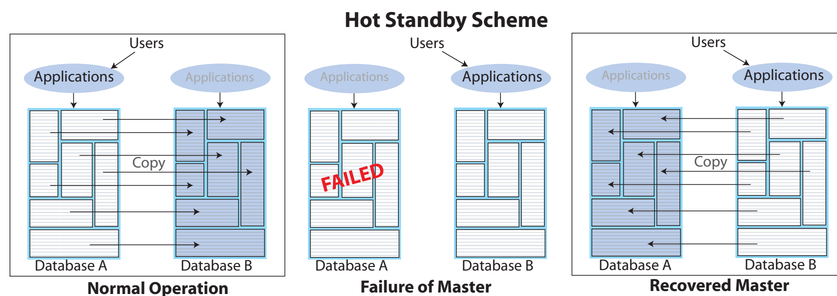
Management of failover and recovery operations are outside the scope of TimesTen. Instead, TimesTen replication has been designed and tested to operate with both custom and third-party cluster managers that detect failures, redirect users or applications from the failed database to one of its subscribers, and manage recovery of the failed database. The cluster manager or administrator can use the TimesTen **ttRepAdmin** `-duplicate` utility or **ttRepDuplicateEx** C function to duplicate the surviving database and recover the failed database.

Subscriber failures generally have no impact on the applications connected to the master databases and can be recovered without disrupting user service. If a failure occurs on a master database, the cluster manager must redirect the application load to a subscriber in order to continue service with no or minimal interruption.

Failover and recovery are more efficient when the databases are configured in a bidirectional general-workload scheme, such as the hot standby scheme shown in Figure 6.8 and the distributed workload scheme shown in Figure 6.9.

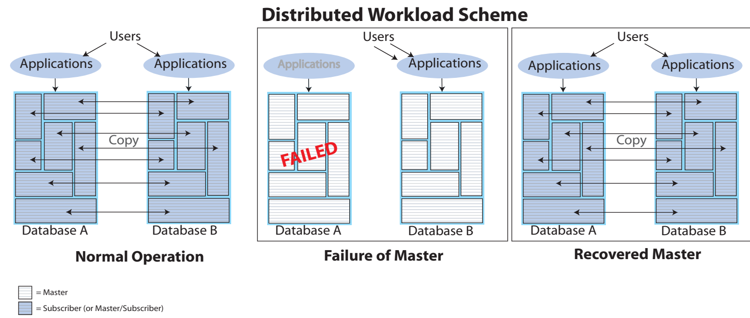
In the hot-standby scheme, if the master database fails, the cluster manager shifts the user load to the "hot standby" application on the subscriber database. Upon recovering the failed database, replication can be resumed with minimal interruption to service by reversing the roles of the master and subscriber databases.

Figure 6.8 Failover scenario for hot standby scheme



The failover procedure for databases configured using a distributed workload scheme is similar to that used for the hot standby except that failover involves shifting the users affected by the failed database to join the other users of an application on a surviving database. Upon recovery, the workload can be redistributed to the application on the recovered database.

Figure 6.9 Failover scenario for distributed workload scheme



For more information

For more information about logging and checkpointing, see Chapter 7, "Transaction Management and Recovery" in *Oracle TimesTen In-Memory Database Operations Guide*.

For more information about replication, see *TimesTen to TimesTen Replication Guide*.

7

Event Notification

TimesTen event notification is done through the Transaction Log API (XLA), which provides functions to detect changes to databases. XLA monitors log records. A log record describes an insert, update or delete on a row in a table. XLA can be used in conjunction with materialized views to focus the scope of notification on changes made to specific rows across multiple tables.

TimesTen can also use SNMP traps to send asynchronous alerts of events.

This chapter includes the following topics:

- Transaction Log API
- Materialized views and XLA
- SNMP traps

Transaction Log API

TimesTen provides a Transaction Log API (XLA) that enables applications to monitor the transaction log of a local database to detect changes made by other applications. XLA also provides functions that enable XLA applications to apply the detected changes to another database. XLA is a C language API. TimesTen provides a C++ wrapper interface for XLA as part of TTClasses, as well as a separate Java wrapper interface.

Applications use XLA to implement a change notification scheme. In this scheme, XLA applications can monitor a database for changes and then take actions based on those changes. For example, a TimesTen database in a stock trading environment might be constantly updated from a data stream of stock quotes. Automated trading applications might use XLA to “watch” the database for updates on certain stock prices and use that information to determine whether to execute orders. See “Scenario 2: Real-time quote service application” on page 22 for a complete example.

XLA can also be used to build a custom data replication solution in place of the TimesTen replication service described in the *TimesTen to TimesTen Replication Guide*. Such XLA-enabled replication solutions might include replication with a non-TimesTen database or *pushing* updates to another TimesTen database.

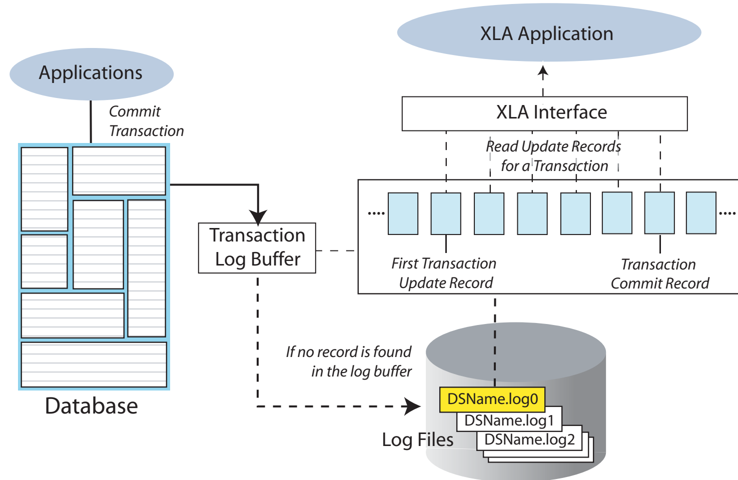
How XLA works

XLA operates in one of two modes:

- Persistent mode
- Nonpersistent mode

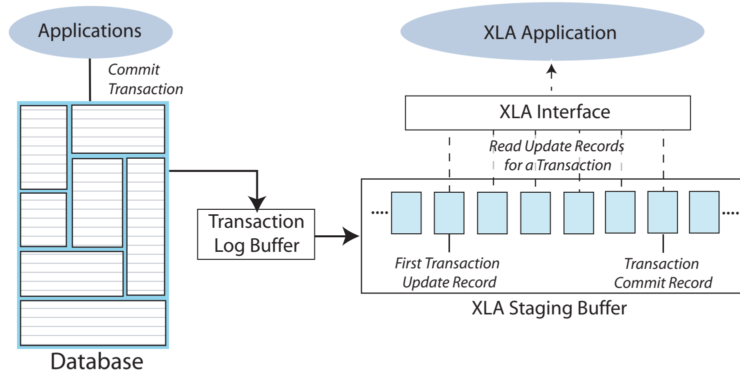
In persistent mode, XLA obtains update records for transactions directly from the transaction log buffer. If the records are not present in the buffer, XLA obtains the update records from the log files on disk, as shown in Figure 7.1. Records are available as long as the log files are available.

Figure 7.1 XLA in persistent mode



In nonpersistent mode, XLA obtains update records from the transaction log and stages them in an *XLA staging buffer*, as shown in Figure 7.2. After records are read from the buffer, they are removed and are no longer available.

Figure 7.2 XLA in nonpersistent mode



Nonpersistent mode is generally faster than persistent mode. However, the staging buffer can only be accessed by one reader at a time and are removed from the buffer after they are read. In addition, all of the update records in the buffer are lost when the database is shut down.When using persistent mode, multiple readers can simultaneously read transaction log updates and the log records are available for as long as the log files are available.

When working in persistent mode, readers use *bookmarks* to maintain their position in the log update stream. Bookmarks are stored in the database, so they are persistent across database connections, shutdowns, and failures.

Log update records

Update records are available to be read from the log as soon as the transaction that created them commits. A "log sniffer" application can obtain groups of update records written to the log.

Each returned record contains a fixed-length update header and one or two rows of data stored in an internal format. The update header describes:

- The table to which the updated row applies
- Whether the record is the first or last commit record in the transaction
- The type of transaction it represents
- The length of the returned row data
- Which columns in the row were updated

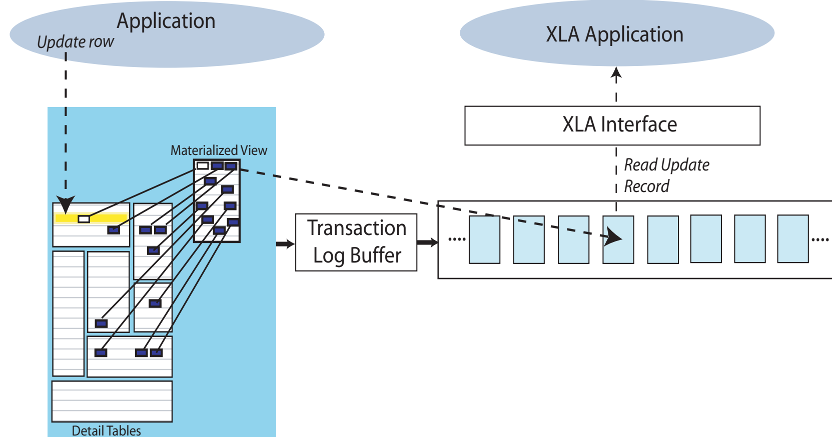
Materialized views and XLA

In most database systems, materialized views are used to simplify and enhance the performance of SELECT queries that involve multiple tables. Though they offer this same capability in TimesTen, another purpose of materialized views in TimesTen is their role in working with XLA to keep track of select rows and columns in multiple tables.

When a materialized view is present, an XLA application only needs to monitor update records that are of interest from a single materialized view. Without a materialized view, the XLA application would have to monitor all of the update records from all of the detail tables, including records reflecting updates to rows and columns of no interest to the application.

Figure 7.3 shows an update made to a column in a detail table that is part of the materialized view result set. The XLA application monitoring updates to the materialized view captures the updated record. Updates to other columns and rows in the same detail table that are not part of the materialized view result set are not seen by the XLA application.

Figure 7.3   Using XLA to detect updates on a materialized view table



See "Scenario 2: Real-time quote service application" on page 22 for an example of a trading application that uses XLA and a materialized view to detect updates to select stocks.

The TimesTen implementation of materialized views emphasizes performance and their function in detecting updates across multiple tables. Readers familiar with other implementations of materialized views should note that the following tradeoffs have been made:

- Materialized views must be explicitly created by the application. The TimesTen query optimizer has no facility to automatically create materialized views.
- The TimesTen query optimizer does not rewrite queries on the detail tables to reference materialized views. Application queries must directly reference views.
- There is no deferred maintenance option for materialized views. A materialized view is refreshed automatically when changes are made to its detail tables and there is no facility for manually refreshing a view.
- There are some restrictions to the SQL used to create materialized views. See CREATE MATERIALIZED VIEW in *Oracle TimesTen In-Memory Database SQL Reference Guide* for details.

SNMP traps

Simple Network Management Protocol (SNMP) is a protocol for network management services. Network management software typically uses SNMP to query or control the state of network devices like routers and switches. These devices sometimes also generate asynchronous alerts in the form of UDP/IP packets, called *SNMP traps*, to inform the management systems of problems.

TimesTen cannot be queried or controlled through SNMP. However, TimesTen sends SNMP traps for certain critical events to facilitate user recovery mechanisms. TimesTen sends traps for the following events:

- Cache Connect to Oracle autorefresh failure
- Database out of space
- Replicated transaction failure
- Death of daemons
- Database invalidation
- Assertion failure

These events also cause log entries to be written by the TimesTen daemon, but exposing them through SNMP traps allows properly configured network management software to take immediate action.

For more information

For more information about XLA, see Chapter 3, "XLA and TimesTen Event Management" in the *Oracle TimesTen In-Memory Database C Developer's and Reference Guide* and Chapter 3, "Using JMS/XLA for Event Management" in *Oracle TimesTen In-Memory Database Java Developer's and Reference Guide*.

For more information about TTClasses, see *Oracle TimesTen In-Memory Database TTClasses Guide*.

For more information about materialized views, see "Understanding materialized views" in Chapter 6, "Working with Data in a TimesTen Data Store of the *Oracle TimesTen In-Memory Database Operations Guide*. Also see CREATE MATERIALIZED VIEW in *Oracle TimesTen In-Memory Database SQL Reference Guide*.

For more information about SNMP traps, see "Diagnostics through SNMP Traps" in *Oracle TimesTen In-Memory Database Error Messages and SNMP Traps*.

8

Cache Connect to Oracle

The Cache Connect to Oracle feature allows TimesTen to cache data stored in one or more tables in an Oracle database.

This chapter includes the following topics:

- Cache groups
- Loading and updating cache groups
- System-managed cache groups
- Usermanaged cache groups
- Replicating cache groups

Cache groups

Cache Connect allows you to cache Oracle data by creating a *cache group* in a TimesTen database. A cache group can be created to cache a single Oracle table or a set of related Oracle tables. The Oracle data cached on TimesTen can consist of all of the rows and columns or a subset of the rows and columns in the Oracle tables.

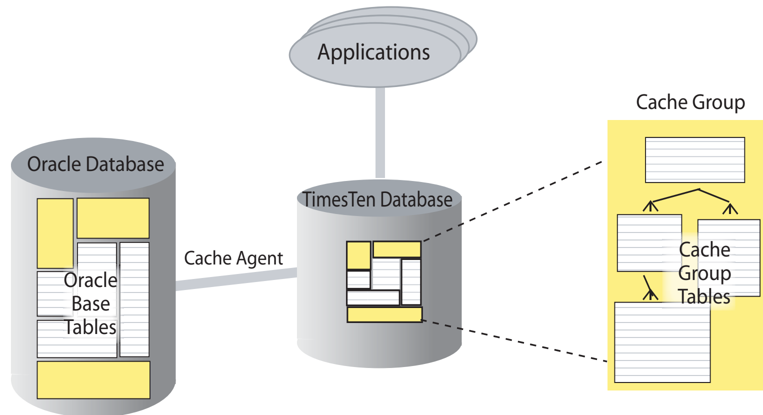
Cache Connect supports the following features:

- Applications can both read from and write to cache groups.
- Cache groups can be refreshed (bring Oracle data into the cache group) automatically or manually.
- Cache updates can be sent to the Oracle database automatically or manually. The updates can be sent synchronously or asynchronously.

The Oracle tables cached in a TimesTen cache group are referred to as *base tables*. The TimesTen Data Manager interacts with Oracle to perform all of the synchronous cache group operations, such as create a cache group and propagate updates between the cache group and the Oracle database. A TimesTen process called the *cache agent* performs asynchronous cache operations, such as loading data into the cache group, manual refreshing of the data from the Oracle database to the cache group, and autorefreshing of the data from the Oracle database to the cache group.

Figure 8.1 illustrates the Cache Connect features and processes.

Figure 8.1 Cache Connect cache group



Cache groups can be created and modified by SQL statements or by the browser-based *Cache Administrator*, as described in the *TimesTen Cache Connect to Oracle Guide*.

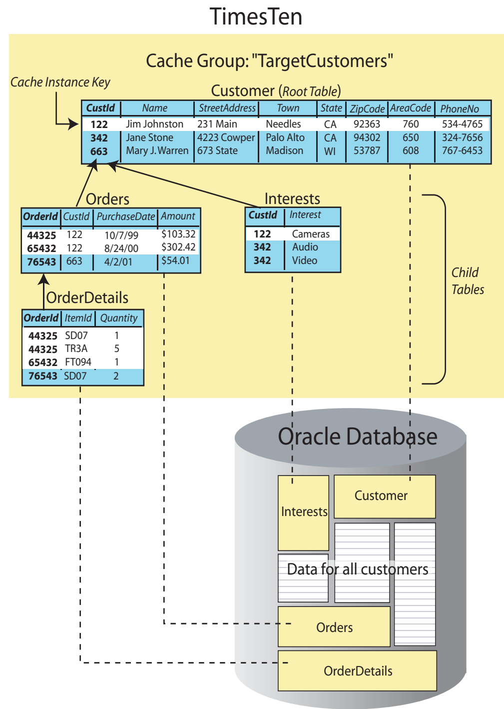
Cache instances

Each cache group has a *root table* that contains the primary key for the cache group. Rows in the root table may have one-to-many relationships with rows in *child tables*, each of which may have one-to-many relationships with rows in other child tables.

A *cache instance* is the set of rows that are associated by foreign key relationships with a particular row in the cache group root table. Each primary key value in the root table specifies a cache instance. Cache instances form the unit of cache loading and cache aging. No table in the cache group can be a child to more than one parent in the cache group. Each TimesTen record belongs to only one cache instance and has only one parent in its cache group.

In the example shown in Figure 8.2, all records in the *Customer*, *Orders*, and *Interests* tables that belong to a given customer ID (*CustId*) are related to each other through foreign key constraints, so they belong to the same cache instance. Because *CustId* uniquely identifies the cache instance, it is referred to as the *cache instance key.*

Figure 8.2 TargetCustomers cache group

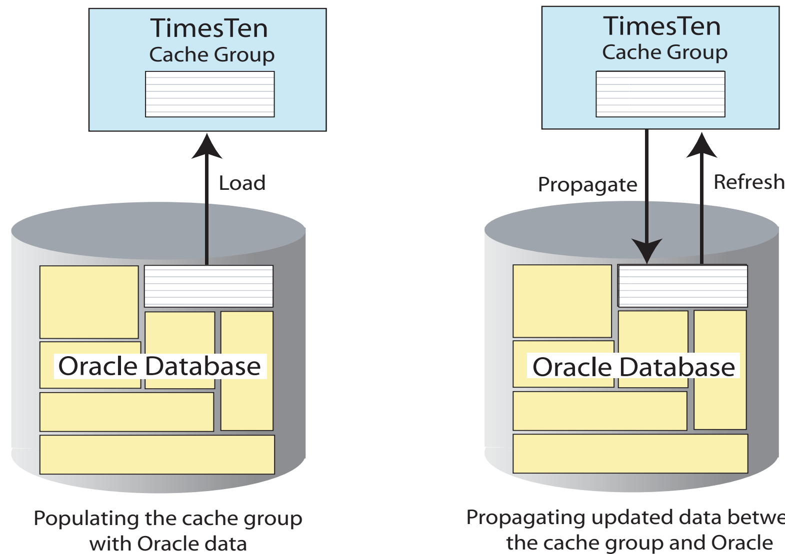


Loading and updating cache groups

The data from Oracle is initially *loaded* into TimesTen to populate the cache group. After loading the cache group, the cached data can be updated in either the TimesTen cache group or the Oracle database. Cache Connect can automatically *propagate* updates from the cache group to Oracle, as well as *refresh* data from Oracle to the cache group.

Figure 8.3 shows data being loaded from Oracle to TimesTen, updates being propagated from TimesTen to Oracle, and updates being refreshed from Oracle to TimesTen.

Figure 8.3 Loading data and propagating or refreshing updates



An application can select from the following techniques to load the cache group data from the Oracle database into TimesTen:

- Load the entire cache group at once. This is a suitable technique to use if the content of the entire cache group can fit in memory. TimesTen also provides the ability to unload an entire cache group.
- Load cache instances by their cache instance key. In this case, the records that make up a cache instance are brought into TimesTen on demand. If the cache instance is already loaded in TimesTen, then the request is ignored. This technique is useful if the capacity of the cache is not large enough to hold all the data described by the cache group. TimesTen also provides the ability to unload cache instances by their cache instance key.
- Load cache instances by WHERE clause if the cache group is USERMANAGED or READONLY. This technique is similar to loading a cache instance by its cache instance key, but in this case, the cache instances

are described by a WHERE clause rather than by ID. TimesTen also provides the ability to unload cache instances by WHERE clause.

- Transparently load data from Oracle tables into cache group tables when a SELECT query on the TimesTen cache group tables does not find data in the tables.

The rest of this section includes the following topics:

- Oracle-to-TimesTen updates
- TimesTen-to-Oracle updates
- Aging feature
- Passthrough feature

Oracle-to-TimesTen updates

The following mechanisms are available to keep the TimesTen cache group up to date with the Oracle base tables:

- *Full autorefresh* – This is specified by the AUTOREFRESH MODE FULL clause in the CREATE CACHE GROUP statement. TimesTen automatically refreshes the entire cache group at specified time intervals.
- *Incremental autorefresh* – This is specified by the AUTOREFRESH MODE INCREMENTAL clause in the CREATE CACHE GROUP statement. Unlike full autorefresh, an incremental autorefresh updates only the records that have been modified in Oracle since the last refresh. TimesTen automatically performs the incremental refresh at specified time intervals.
- *Manual refresh* – This is specified by an application issuing an explicit REFRESH CACHE GROUP statement to refresh either an entire cache group or specific cache instances. It is equivalent to an UNLOAD CACHE GROUP operation followed by a LOAD CACHE GROUP operation.
- *Transparent load* – This is specified by setting the **TransparentLoad** attribute. It enables TimesTen to automatically load data from Oracle tables into cache group tables when a SELECT query on the cache group tables does not find data in the tables. Transparent load can be implemented for all types of cache groups except cache groups with the AUTOREFRESH attribute.

Each mechanism has advantages and tradeoffs. *Incremental autorefresh* refreshes only changed rows, but requires the use of triggers on Oracle to keep track of the updates. This adds overhead and slows down updates. *Full autorefresh* does not require Oracle to keep track of the updates but updates everything in the cache at once. *Manual refresh* is controlled by the application and can be customized to specific cache instances and specific schedules, but the logic needed to determine when to refresh adds overhead to the application. *Transparent load* enables specific data to be loaded automatically on demand into TimesTen from Oracle tables.

These mechanisms are useful under different circumstances. For example, a full autorefresh may be the best choice if the Oracle table is updated only once a day and many rows are changed. An incremental autorefresh is the best choice if the Oracle table is updated often, but only a few rows are changed with each update. A manual refresh is the best choice if the application logic knows when the refresh should happen. Transparent load is useful when cache content is dynamic and a specific cache instance with the latest data is needed. For example, when a customer contacts a call center, the customer's information can be loaded quickly.

TimesTen-to-Oracle updates

For data that is updated in a TimesTen cache group, the following mechanisms are available to keep the Oracle database up to date with the cache group:

- *Propagate* – This is performed by one of the following methods:
  - Specifying the PROPAGATE option in the CREATE USERMANAGED CACHE GROUP statement.
  - Creating a synchronous writethrough (SWT) cache group with the CREATE SYNCHRONOUS WRITETHROUGH CACHE GROUP statement.
  - Creating an asynchronous writethrough (AWT) cache group with the CREATE ASYNCHRONOUS WRITETHROUGH CACHE GROUP statement.

  With the PROPAGATE option enabled, all modifications to a cache group are automatically propagated to the Oracle database. When the application completes a transaction that has modified one or more cache groups that have the PROPAGATE option enabled or are SWT cache groups, TimesTen first commits the transaction in Oracle and then in TimesTen. This technique allows Oracle to apply any required logic related to the data before it is committed in TimesTen, so that Oracle always reflects the latest image of the data. Use the PROPAGATE option or SWT cache groups when the cache and Oracle must be synchronized at all times.

  Modifications to an AWT cache group are committed without waiting for the changes to be applied to Oracle. AWT cache groups provide better response times and performance, but the cache and Oracle do not always contain the same data because changes are applied to Oracle asynchronously.
- *Flush* – This is specified by an application issuing an explicit FLUSH CACHE GROUP statement. A flush operation can be used to propagate updates manually from a USERMANAGED TimesTen cache group to Oracle. Flush operations are useful when frequent updates occur for a limited period of time over a set of records. Rather than propagate the updates of each transaction individually, the updates are batched all at once with the flush operation. Flush operations require that the PROPAGATE option in the

CREATE CACHE GROUP statement be disabled. Flush operations do not propagate deletes.

Aging feature

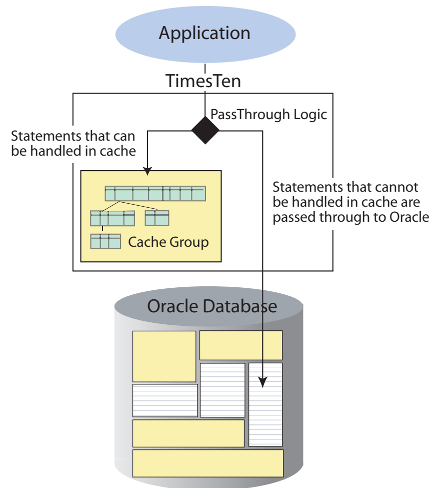
When cache instances are loaded into TimesTen, they can be automatically aged out of TimesTen. You can specify usage-based aging or time-based aging. Usage-based aging removes least recently used (LRU) data within a specified range of memory usage. Time-based aging removes data based on the specified lifetime and frequency of the aging process. Aging can be defined on only the root table of a cache group because aging is based on the cache instance. You can configure both usage-based and time-based aging in the same system, but you can define only one type of aging on a specific cache group.

For more information, see "Implementing aging in a cache group" in *TimesTen Cache Connect to Oracle Guide*.

Passthrough feature

TimesTen applications can send SQL statements to either a TimesTen cache group or to the Oracle database through a single connection to a TimesTen database. This single-connection capability is enabled by a *passthrough* feature that checks if the SQL statement can be handled locally by the cached tables in TimesTen or if it must be redirected to Oracle, as shown in Figure 8.4. The passthrough feature provides settings that specify what types of statements are to be passed through and under what circumstances.

Figure 8.4 TimesTen passthrough feature



The specific behavior of the passthrough feature is controlled by the **PassThrough** database attribute. See "Setting a passthrough level" in *TimesTen Cache Connect to Oracle Guide*.

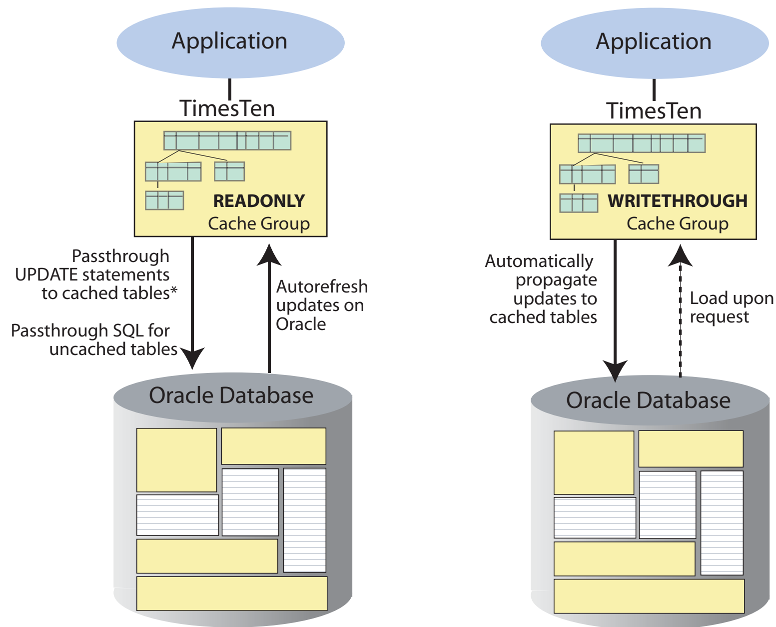
System-managed cache groups

System-managed cache groups are predefined frameworks for caching Oracle data in TimesTen. When using a system-managed cache group, operations such as propagating data are managed automatically by the cache agent.

As shown in Figure 8.5, there are two basic types of system-managed cache groups:

- READONLY cache groups hold read-only data, so updates on the tables in the cache group are not allowed. Updates must be done on Oracle. These updates can either be done directly in Oracle or in TimesTen using the passthrough feature described in "Passthrough feature" on page 73. By default, a READONLY cache group is automatically refreshed from Oracle.
- WRITETHROUGH cache groups load the cached table data from Oracle once when created. Thereafter, all updates to the cache group are automatically propagated to Oracle. Writethrough cache groups can be either asynchronous (AWT) or synchronous (SWT). SWT cache groups wait for a commit on Oracle before committing in the cache. AWT cache groups commit changes in the cache without waiting for a commit on Oracle.

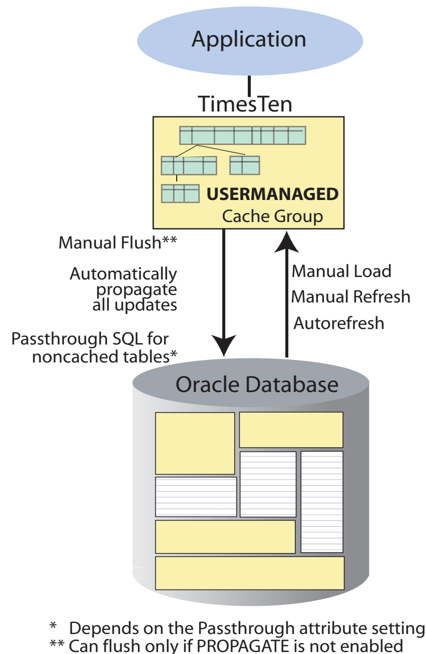
Figure 8.5 Read-only and writethrough cache groups



Usermanaged cache groups

Usermanaged cache groups allow users to select from a full range of attributes and SQL statements to define customized caching behaviors. Users have full control over how and when the cached data is loaded, propagated, and removed from the cache group. Figure 8.6 shows some of the features of a usermanaged cache group and the operations that can be performed.

Figure 8.6 User-managed cache group



Replicating cache groups

TimesTen replication can be configured to replicate a cache group from one database to a cache group in another database. Only cache groups of the same type can be replicated. For example, a READONLY cache group can be bidirectionally replicated only with other READONLY cache groups.

You can also replicate tables in a cache group to regular TimesTen tables.

For more information

For more information about Cache Connect, see *TimesTen Cache Connect to Oracle Guide*.

For more information about replicating cache groups, see "Replicating cache groups" in *TimesTen to TimesTen Replication Guide*.

9

TimesTen Administration

This chapter includes the following topics:

- Installing TimesTen
- TimesTen Access Control
- Command line administration
- SQL administration
- Browser-based cache group administration
- ODBC Administrator
- Upgrading TimesTen

Installing TimesTen

TimesTen software is easily installed. On UNIX systems, TimesTen is installed by a simple set-up script. On Windows, TimesTen is installed by running InstallShield®. The installation includes the TimesTen client, server, and data manager components, or just the TimesTen client or data manager by themselves. You can also choose whether to install the Cache Connect to Oracle option.

TimesTen can be installed with Access Control enabled or disabled, as described in "TimesTen Access Control" on page 77.

TimesTen Access Control

TimesTen can be installed with *Access Control* enabled to allow only users with specific privileges to access particular TimesTen features.

TimesTen Access Control uses standard SQL operations, such as CREATE USER, DROP USER, GRANT, and REVOKE, to establish TimesTen user accounts with specific privilege levels. In TimesTen, privileges are granted for the entire TimesTen instance, rather than for specific tables. Each user's privileges apply to all databases in a TimesTen instance or installation.

If Access Control is to be enabled, installation must be performed by the instance administrator for the database. The instance administrator owns all files in the installation directory tree and is the only user with the privileges to start and stop the TimesTen daemon and other TimesTen processes, such as the replication and cache agents.

To enable client connections to an access-controlled database, you must enable the **Authenticate** database attribute and the client must specify the correct user name and password in the client DSN or when connecting to TimesTen.

The Access Control feature of TimesTen provides an environment of basic control for applications that use the defined privileges. Access Control does not provide definitive security for all processes that might be able to access the database. For example, this feature does not protect the database from user processes that may have sufficient privileges to attach to the database when in memory or that can access files on disk that are associated with the database, such as log files and checkpoint files.

Command line administration

Most TimesTen administration tasks are performed with command line utilities. Common utilities are summarized in the following table:

Name	Description
ttAdmin	A general utility for managing TimesTen databases. Used to specify policies for automatically or manually loading and unloading databases from RAM, as well as to starting and stopping TimesTen cache agents and replication agents.
ttBackup and **ttRestore**	Used to create a backup copy of a database and restore it at a later time.
ttBulkCp	Used to transfer data between TimesTen tables and ASCII files.
ttIsql	Used to run SQL interactively from the command line. Also provides a number of administrative commands to reconfigure and monitor databases.
ttMigrate	Used to save TimesTen tables and cache group definitions to a binary data file. Also used to restore tables and cache group definitions from the binary file.
ttRepAdmin	Used to monitor replication status.
ttSize	Used to estimate the amount of space to allocate for a table in the database.
ttStatus	Used to display information that describes the current state of TimesTen.

Name	Description
ttTraceMon	Used to enable and disable the TimesTen internal tracing facilities.
ttXactAdmin	Used to list ownership, status, log and lock information for each outstanding transaction. The **ttXactAdmin** utility also allows users to commit, abort or forget an XA transaction branch.

SQL administration

TimesTen provides SQL statements for administrative activities such as creating and managing tables, replication schemes, cache groups, materialized views, and indexes.

The metadata for each TimesTen database is stored in a group of system tables. Applications can use SQL SELECT queries on these tables to monitor the current state of a database.

Administrators can use the **ttIsql** utility for SQL interaction with a database. For example, there are several built-in **ttIsql** commands that display information on database structures.

Browser-based cache group administration

TimesTen provides a Web browser-based tool called the *Cache Administrator* that can be used by any machine within the intranet to create and manage Cache Connect cache groups. The Cache Administrator can be used to create a cache group directly or to generate the SQL statements that define a cache group.

ODBC Administrator

The ODBC Administrator is a utility program used on Windows to create, configure and delete data source definitions. You can use it to define a data source and set connection attributes.

Upgrading TimesTen

TimesTen provides the facilities to perform three types of upgrades:

- In-place upgrades
- Offline upgrades
- Online upgrades

In-place upgrades

In-place upgrades are typically used to move to a new patch release (or "dot-dot release") of TimesTen.

In-place upgrades can be done without destroying the existing databases. However, all applications must first disconnect from the databases, and the databases must be unloaded from shared memory. After uninstalling the old release of TimesTen and installing the new release, applications can reconnect to the databases and resume operation.

Offline upgrades

Offline upgrades are performed by using the **ttMigrate** utility to export the database into an external file and to restore the database with the desired changes.

Use offline upgrades to perform the following tasks:

- Move to a new major TimesTen release or a dot release
- Move to a different directory or machine
- Reduce database size

During an offline upgrade, the database is not available to applications. Offline upgrades usually require enough disk space for an extra copy of the upgraded database.

Online upgrades

TimesTen replication enables online upgrades, which can be performed online by the **ttMigrate** and **ttRepAdmin** utilities while the database and its applications remain operational and available to users. Online upgrades are useful for applications where continuous availability of the database is critical.

Use online upgrades to perform the following tasks:

- Move to a new major release (or "dot release") of TimesTen and retain continuous availability to the database
- Increase or reduce the database size
- Move the database to a new location or machine

Updates made to the database during the upgrade are transmitted to the upgraded database at the end of the upgrade process. Because an online upgrade requires that the database be replicated to another database, it can require more memory and disk space than offline upgrades.

For more information

For more information about installing and upgrading TimesTen, see *Oracle TimesTen In-Memory Database Installation Guide*.

For more information about general administration of TimesTen, see "Creating TimesTen Data Stores" and "Working with Data in a TimesTen Data Store" in *Oracle TimesTen In-Memory Database Operations Guide*. These chapters include the use of the ODBC Administrator.

For more information about Cache Connect administration, see *TimesTen Cache Connect to Oracle Guide*.

For more information about administration of TimesTen replication, see *TimesTen to TimesTen Replication Guide*.

For a complete list of SQL statements, see "SQL Statements" in *Oracle TimesTen In-Memory Database SQL Reference Guide*.

For a complete list of TimesTen command-line utilities, see "Utilities" in *Oracle TimesTen In-Memory Database API Reference Guide*.

Index

A

Access Control
  security 10, 77
active standby pair 55
administration of TimesTen 14, 77
aging
  cache groups 73
  tables 13
application scenarios 19
architecture
  TimesTen base product 26
asynchronous writethrough cache groups 72, 75
autocommit mode 49
autorefresh
  full 71
  incremental 71
availability
  replicating cache groups 56
AWT cache groups 72, 75

B

B-tree index 43
blocking checkpoints 51

C

cache
  distributed 18
Cache Administrator 68, 79
cache agent
  TimesTen architecture 30
Cache Connect
  autorefresh 71
  described 67
  loading cache groups 70
  overview 67
  TimesTen architecture 28
  updating cache groups 70
Cache Connect to Oracle 14
cache group
  aging 73
  asynchronous writethrough 72, 75
  AWT 75
  described 67
  flush 72
  manual refresh 71
  passthrough feature 73
  PROPAGATE option 72
  read-only 18
  READONLY 75
  SWT 75
  synchronous writethrough 72, 75
  system-managed 75
  TimesTen architecture 30
  updatable 18
  using 24
cache instance 68
character set support 13
checkpoint files
  TimesTen architecture 27
checkpoints 11
  "fuzzy" 51
  blocking 51
  described 50
  nonblocking 51
  recovering from 51
client/server connection 11, 31
cluster manager, role of 58
command line utilities 78
concurrency 12
connections
  client/server 31
  direct driver 31
  driver manager 32
  types of 31

D

data aging 13
database
  recovery of 51, 58
database processes
  TimesTen architecture 27
direct driver connection 11, 31
distributed cache 18
distributed transaction processing 11, 33
distributed workload replication 54
  recovery 59
driver manager connection 11, 32
DTP 33
DurableCommits connection attribute 49

F
failover and recovery 58
FLUSH CACHE GROUP SQL statement 72
full table scan 44
fuzzy checkpoints 51

G
globalization support 13

H
hash index 44
hash index scan 44
high availability
  replicating cache groups 56
hot-standby replication
  recovery issues 58
  use of 20, 22

I
in-place upgrade 80
indexing techniques 43
inner table 45
installing TimesTen 77
instance administrator 77
isolation level 36
  read committed 36
  serializable 37

J
Java Transaction API 11, 33
JDBC support 10
join methods 45
JTA 33

L
length semantics 13
linguistic sorts 13
load on SELECT 71
locks 38
  database level 38
  row level 39
  table level 38
log files
  deleting 50
  TimesTen architecture 27
logging 11, 49
  disabled 50

M
master database 51
materialized views
  and XLA 63
  use of 23
merge join 46

N
nested loop join 45

O
ODBC Administrator 79
ODBC direct driver connection 31
  use of 20
ODBC support 10
off-line upgrade 80
optimizer hints 42
optimizer plan 47
outer table 45

P
passthrough 73

Q
query optimizer 12, 41

R
read committed isolation 36
read-only cache 18
READONLY cache group 75
recovering a database 51, 58
replication
  configuration 52
  described 51
  failover and recovery 58
  return receipt 56
  return twosafe 56
  TimesTen architecture 28
Replication - TimesTen to TimesTen 14
replication agent
  defined 52
  TimesTen architecture 28
return receipt replication 56
return receipt service 56
return twosafe replication 56
return twosafe service 56
Rowid lookup 44

S

scan methods 44
security
  Access Control 10, 77
selectivity, defined 42
serializable isolation 37
SNMP traps 64
split workload replication 54
SQL administration 79
SQL-92 support 10
subscriber database 51
SWT cache groups 72, 75
synchronous writethrough cache groups 72, 75
SYS.COL_STATS table 42
SYS.PLAN table 47
SYS.TBL_STATS table 42
system-managed cache group 75

T

T-tree index 43
T-tree index scan 44
TimesTen administration 77
TimesTen architecture
  connecting to the database 31
TimesTen C++ Interface Classes 33
TimesTen uses 18
transaction isolation levels 12, 36
Transaction Log API 13, 33
transaction log monitoring 13
transparent load 71
  using 24
TransparentLoad connection attribute 71
TTClasses 33
ttDurableCommit procedure 49
ttOptGetFlag procedure 43
ttOptSetFlag procedure 42, 47
ttOptSetOrder procedure 43
ttOptUpdateStats procedure 42
ttOptUseIndex procedure 43

U

updatable cache 18
upgrade modes
  in-place upgrade 80
  off-line upgrade 80
upgrading TimesTen 79

V

versioning 36

X

XA interface 11
XLA 33
  and materialized views 63
  described 13, 61
  modes 62
  use of 23