

# **Oracle® TimesTen In-Memory Database**

Introduction

Release 18.1

**E61192-05**

November 2019

Copyright © 2012, 2019, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

---

---

# Contents

<b>Preface</b> .....	vii
Audience .....	vii
Related documents .....	vii
Conventions .....	viii
Documentation Accessibility .....	ix
<b>What's New</b> .....	xi
New features in Release 18.1.3.1.0 .....	xi
New features in Release 18.1.1.1.0 .....	xii
<b>1 Overview for the Oracle TimesTen In-Memory Database</b>	
<b>TimesTen database performance overview</b> .....	1-2
<b>TimesTen In-Memory Database general feature overview</b> .....	1-3
TimesTen API and language support .....	1-3
SQL and PL/SQL functionality .....	1-3
OCI, ODBC and JDBC interfaces .....	1-4
Pro*C/C++ Precompiler support .....	1-4
ODP.NET support .....	1-4
Open source programming languages .....	1-4
TTClasses .....	1-5
Transactional ACID compliance .....	1-5
Durability .....	1-5
Durability for TimesTen Classic .....	1-5
Durability for TimesTen Scaleout .....	1-5
Performance through query optimization .....	1-6
Concurrency .....	1-6
Database character sets and globalization support .....	1-7
Database connectivity .....	1-7
Load data from an Oracle database into a TimesTen table .....	1-7
System monitoring .....	1-8
System monitoring through system tables and views .....	1-8
System monitoring for TimesTen Classic .....	1-8
System monitoring for TimesTen Scaleout .....	1-8
Administration and utilities .....	1-9
<b>TimesTen Scaleout specific features</b> .....	1-10

High availability features for TimesTen Scaleout .....	1-10
Transparent data distribution .....	1-11
Online elastic scalability .....	1-12
Automatic high availability through K-Safety .....	1-12
Single point for administration and monitoring .....	1-12
<b>TimesTen Classic specific features</b> .....	1-12
High availability features for TimesTen Classic .....	1-12
Data replication between servers .....	1-13
Transaction Log API (XLA) .....	1-14
Automatic data aging .....	1-14
<b>TimesTen Cache specific features</b> .....	1-14

## 2 Using TimesTen

<b>Uses for TimesTen Scaleout</b> .....	2-1
TimesTen Scaleout application scenario .....	2-2
<b>Uses for TimesTen Classic</b> .....	2-3
TimesTen Classic application scenario .....	2-3
<b>Uses for TimesTen Cache</b> .....	2-5
TimesTen Cache application scenario .....	2-6

## 3 TimesTen Architecture

<b>Architectural overview</b> .....	3-1
Architectural overview of TimesTen Classic and TimesTen Cache .....	3-1
Architectural overview of TimesTen Scaleout .....	3-2
<b>Shared libraries</b> .....	3-3
<b>Memory-resident data structures</b> .....	3-4
<b>Database processes</b> .....	3-4
TimesTen daemon .....	3-4
Subdaemons .....	3-4
<b>TimesTen connection options</b> .....	3-5
Direct connection .....	3-5
Client/server connection .....	3-6
Driver manager connection .....	3-6
<b>Checkpoint and transaction log files</b> .....	3-7
<b>Cached data</b> .....	3-7
<b>Replication</b> .....	3-7

## 4 Concurrent Operations

<b>Transaction isolation</b> .....	4-1
Read committed isolation .....	4-1
Serializable isolation .....	4-2
<b>Locking levels</b> .....	4-3
Row-level locking .....	4-3
Table-level locking .....	4-4
Database-level locking .....	4-4

## 5 Query Optimization

Optimization time and memory usage.....	5-2
Statistics .....	5-2
Optimizer hints.....	5-3
Indexes.....	5-3
Scan methods .....	5-4
Join methods.....	5-4
Nested loop join.....	5-5
Merge join.....	5-6
Optimizer plan.....	5-7

## 6 Data Availability and Integrity

Transaction logging .....	6-1
Writing the transaction log buffer to disk .....	6-1
When are transaction log files deleted? .....	6-2
TimesTen commits .....	6-2
Checkpointing.....	6-2
Nonblocking checkpoints .....	6-3
Blocking checkpoints .....	6-3
Recovery from the transaction log and checkpoint files .....	6-3
Data Replication within TimesTen Classic.....	6-4
Active standby pair.....	6-4
Classic replication configurations.....	6-6
Unidirectional replication.....	6-6
Bidirectional replication.....	6-8
Asynchronous and return service replication.....	6-9
Replication failover and recovery.....	6-9
Automatic client failover for active standby pair replication.....	6-10

## 7 TimesTen Cache

Cache groups .....	7-1
Dynamic cache groups and explicitly loaded cache groups .....	7-3
Transmitting data between the TimesTen Cache and an Oracle database.....	7-3
Updating a cache group from the Oracle database tables .....	7-3
Updating the Oracle database tables from a cache group .....	7-4
Aging feature.....	7-4
Passthrough feature .....	7-5
Replicating cache groups .....	7-5
Disaster recovery through TimesTen Cache .....	7-6

## 8 Event Notification

Detect transaction modifications with the Transaction Log API.....	8-1
How XLA works.....	8-2
Log update records .....	8-2
Monitor tables with materialized views and XLA.....	8-3

Monitor problems with SNMP traps.....	8-4
---------------------------------------	-----

## 9 TimesTen Administration

Installing TimesTen.....	9-1
Access control .....	9-1
Command line administration.....	9-1
SQL administration.....	9-2
SQL Developer.....	9-3
Facilitating TimesTen Scaleout development and management .....	9-3
Facilitating TimesTen Classic development and management.....	9-3
ODBC Administrator.....	9-4
Migrating a database .....	9-4
Upgrading TimesTen.....	9-4
Upgrades for a grid in TimesTen Scaleout .....	9-4
Offline upgrades for TimesTen Classic.....	9-4
Online upgrades for TimesTen Classic .....	9-5

## Index

---

# Preface

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

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

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

## Audience

This document is intended for readers with a basic understanding of database systems.

## Related documents

TimesTen documentation is available at:

<https://docs.oracle.com/database/timesten-18.1>

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

## Conventions

TimesTen Classic is supported on multiple platforms. Unless otherwise indicated, the information in this guide applies to all supported platforms. The term Windows refers to all supported Windows platforms. The term UNIX applies to all supported UNIX platforms. The term Linux is used separately.

TimesTen Scaleout is only supported on the Linux platform.

See "Platforms and Compilers" in *Oracle TimesTen In-Memory Database Release Notes* (README.html) in your installation directory for specific platform versions supported by TimesTen..

---

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

---

This document uses the following text conventions:

Convention	Meaning
<b>boldface</b>	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.
<i>italic monospace</i>	Italic monospace type indicates a placeholder or a variable in a code example for which you specify or use a particular value. For example:  <code>LIBS = -Ltimesten_home/install/lib -ltten</code> Replace <i>timesten_home</i> with the path to the TimesTen instance home directory.
[ ]	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 alternative arguments.
...	An ellipsis ( . . . ) after an argument indicates that you may use more than one argument on a single command line.
% or \$	The percent sign or dollar sign indicates the UNIX shell prompt, depending on the shell that is used.
#	The number (or pound) sign indicates the UNIX root prompt.

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

Convention	Meaning
<i>installation_dir</i>	The path that represents the directory where the current release of TimesTen is installed.



Convention	Meaning
<i>timesten_home</i>	The path that represents the home directory of a TimesTen instance.
<i>release</i> or <i>rr</i>	The first two parts in a release number, with or without the dot. The first two parts of a release number represent a major TimesTen release. For example, 181 or 18.1 represents TimesTen Release 18.1.
<i>DSN</i>	The data source name.

## Documentation Accessibility

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

### Access to Oracle Support

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



---

---

# What's New

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

## New features in Release 18.1.3.1.0

There are significant updates to the *Oracle TimesTen In-Memory Database Introduction* to include information about TimesTen Scaleout in addition to new features for this release.

- ["TimesTen In-Memory Database general feature overview"](#) on page 1-3 describes features that are general to TimesTen Scaleout, TimesTen Classic and TimesTen Cache. These were updated as appropriate.
- Open source languages, which are new for this release, interact with TimesTen through the Oracle Database Programming Interface for C (ODPI-C). The languages currently supported are Python and Node.js. See ["Open source programming languages"](#) on page 1-4 for details.
- ["Durability"](#) on page 1-5 now describes how durability is accomplished with both TimesTen Scaleout and TimesTen Classic.
- ["System monitoring"](#) on page 1-8 now describes how to monitor both TimesTen Scaleout and TimesTen Classic systems.
- There are now sections that describe the specific features for each product. See ["TimesTen Scaleout specific features"](#) on page 1-10, ["TimesTen Classic specific features"](#) on page 1-12, and ["TimesTen Cache specific features"](#) on page 1-14 for details.
- ["Uses for TimesTen Scaleout"](#) on page 2-1 describes uses and a specific scenario for TimesTen Scaleout.
- ["Architectural overview of TimesTen Scaleout"](#) on page 3-2 details an architectural overview of TimesTen Scaleout.
- ["Disaster recovery through TimesTen Cache"](#) on page 7-6 describes how TimesTen Cache supports Oracle Data Guard, as well as providing its own HA capability through active standby pair replication of cache tables for read-only and AWT cache groups.
- ["Monitor problems with SNMP traps"](#) on page 8-4 describes how TimesTen uses Simple Network Management Protocol (SNMP).
- ["Command line administration"](#) on page 9-1 includes the `ttGridAdmin` utility.

- ["Facilitating TimesTen Scaleout development and management"](#) on page 9-3 details how to use SQL Developer to facilitate TimesTen Scaleout development and management.
- ["Installing TimesTen"](#) on page 9-1 and ["Migrating a database"](#) on page 9-4 now describe how to install TimesTen Scaleout and migrate from a TimesTen Classic database to a TimesTen Scaleout database.
- ["Upgrading TimesTen"](#) on page 9-4 contains new details on the process for upgrading to a new release of TimesTen.

## New features in Release 18.1.1.1.0

- There are now two modes for Oracle TimesTen In-Memory Database.  

Oracle TimesTen In-Memory Database in grid mode (TimesTen Scaleout) is a grid of interconnected hosts running TimesTen Scaleout instances that work together to provide fast access, fault tolerance, and high availability for in-memory data. A grid contains one or more databases and each database is distributed across all instances of the grid. TimesTen Scaleout delivers these features by distributing the data for each in-memory database in the grid across multiple hosts. See the *Oracle TimesTen In-Memory Database Scaleout User's Guide* for full details.

Oracle TimesTen In-Memory Database in classic mode or TimesTen Classic refers to single-instance environments and databases and referred to as TimesTen alone in previous releases.

Now, when the documentation refers to TimesTen alone (without TimesTen Classic or TimesTen Scaleout), the statement applies to both single-instance and multiple-instance, such as in references to TimesTen utilities, releases, distributions, installations, actions taken by the database, and functionality within the database.
- TimesTen Scaleout is only supported on the Linux platform, while TimesTen Classic is supported on multiple platforms. See the *Oracle TimesTen In-Memory Database Release Notes (README.html)* in your installation directory for specific platform versions supported by TimesTen.

---

# Overview for the Oracle TimesTen In-Memory Database

Oracle TimesTen In-Memory Database (TimesTen) is a relational database that is memory-optimized for fast response and throughput. You can create one or more in-memory, SQL relational, ACID-complaint databases. The database resides entirely in memory at runtime and is persisted to disk storage for the ability to recover and restart.

- Oracle TimesTen In-Memory Database in classic mode (TimesTen Classic) is a memory-optimized relational database that provides applications with the responsiveness and high throughput. High availability for the in-memory database is provided through transactional replication.
- Oracle TimesTen In-Memory Database in grid mode (TimesTen Scaleout) is a grid of interconnected hosts running TimesTen Scaleout instances that work together to provide fast access, fault tolerance, and high availability for in-memory data. K-safety provides fault tolerance by enabling you to define the number of copies of data to ensure that your database continues to operate in spite of various faults (as long as a single copy of the data is accessible). A grid contains one or more databases and each database is distributed across all instances of the grid.
- Oracle TimesTen Application-Tier Database Cache (TimesTen Cache) is an Oracle Database product option ideal for caching performance-critical subsets of an Oracle database for improved response time in the application tier. Cache tables can be read-only or updatable. Applications read and update the cache tables using standard SQL, and data synchronization between the cache and the Oracle database is performed automatically. TimesTen Application-Tier Database Cache offers applications the full generality and functionality of a relational database, the transparent maintenance of cache consistency with the Oracle database, and the high performance of an in-memory database.

The following sections provide an overview for TimesTen:

- [TimesTen database performance overview](#)
- [TimesTen In-Memory Database general feature overview](#)
- [TimesTen Scaleout specific features](#)
- [TimesTen Classic specific features](#)
- [TimesTen Cache specific features](#)

## TimesTen database performance overview

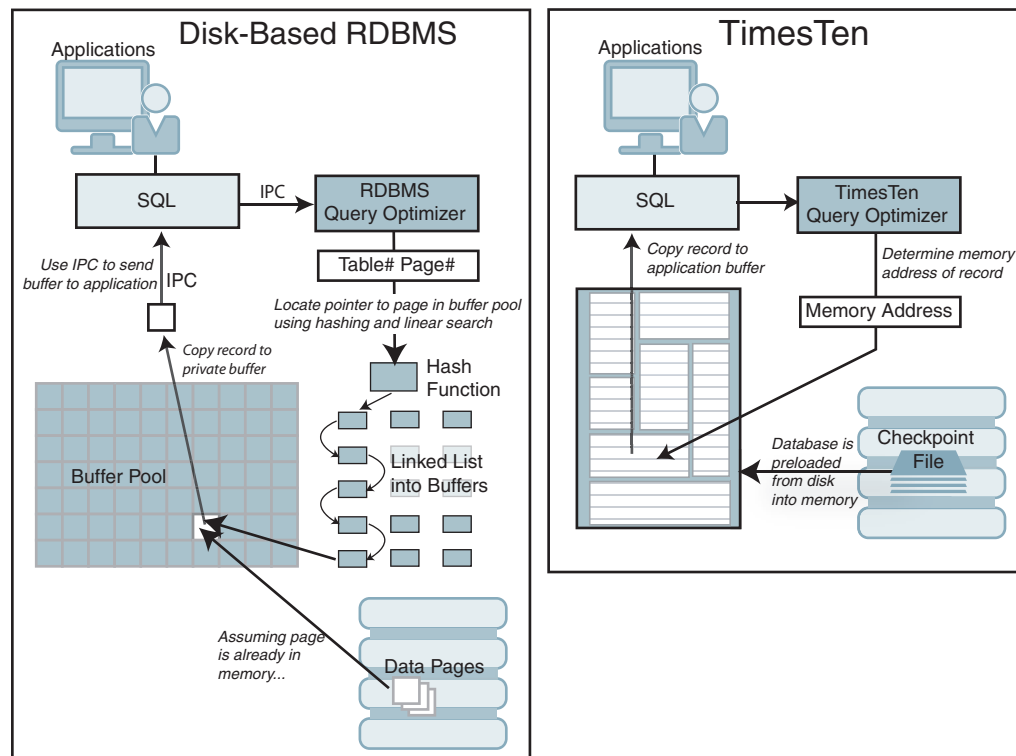
Much of the work that is done by a conventional, disk-optimized RDBMS is done under the assumption that data primarily resides on disk. Optimization algorithms, buffer pool management, and indexed retrieval techniques are designed based on this fundamental assumption.

Even when a disk-based RDBMS has been configured to hold all of its data in main memory, its performance is limited 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.

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 Classic**



TimesTen Scaleout, TimesTen Classic, and TimesTen Cache all deliver high performance since they all use the Oracle TimesTen In-Memory Database as their RDBMS. TimesTen Scaleout provides the best throughput; TimesTen Classic provides the best latency. The TimesTen in-memory database delivers high performance by changing the assumptions about where data resides at runtime. The TimesTen in-memory database manages data in memory and optimizes data structures and access algorithms accordingly. Thus, 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).

## TimesTen In-Memory Database general feature overview

Unless otherwise indicated, TimesTen Scaleout, TimesTen Classic, and TimesTen Cache have the same database features.

- [TimesTen API and language support](#)
- [Transactional ACID compliance](#)
- [Durability](#)
- [Performance through query optimization](#)
- [Concurrency](#)
- [Database character sets and globalization support](#)
- [Database connectivity](#)
- [Load data from an Oracle database into a TimesTen table](#)
- [System monitoring](#)
- [Administration and utilities](#)

## TimesTen API and language support

TimesTen supports:

- Industry standard language for RDBMS: SQL.
- Industry standard APIs: ODBC, JDBC, and ODP.NET.
- Oracle APIs: PL/SQL, Oracle Call Interface (OCI), and ProC/C++.
- TTClasses and built-in procedures provided by TimesTen that extend ODBC functionality for TimesTen-specific operations.
- Open source programming languages, such as Python and Node.js.

The runtime architecture of TimesTen supports connectivity through the ODBC, JDBC, OCI, Pro\*C/C++ Precompiler and ODP.NET APIs.

- [SQL and PL/SQL functionality](#)
- [OCI, ODBC and JDBC interfaces](#)
- [Pro\\*C/C++ Precompiler support](#)
- [ODP.NET support](#)
- [Open source programming languages](#)
- [TTClasses](#)

### SQL and PL/SQL functionality

TimesTen supports extensive SQL functionality to simplify the configuration and management of special features. Using SQL to shield applications from system internals enables a database 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. Any developer familiar with the Oracle database or SQL interfaces can be immediately productive developing

applications with TimesTen. For more information, see the *Oracle TimesTen In-Memory Database SQL Reference*.

TimesTen supports Oracle PL/SQL (Procedural Language Extension to SQL), a programming language that enables you to integrate procedural constructs with SQL for a TimesTen in-memory database. You can execute PL/SQL from all supported APIs. For more information, see the *Oracle TimesTen In-Memory Database PL/SQL Developer's Guide*.

### **OCI, ODBC and JDBC interfaces**

TimesTen supports the Oracle Call Interface (OCI) for TimesTen functionality.

TimesTen OCI support enables you to run many existing OCI applications with TimesTen in direct mode or client/server mode. TimesTen OCI also enables you to use other Oracle Database products that use OCI as a database interface. You can call PL/SQL from OCI applications.

TimesTen also supports the industry standard ODBC and JDBC API interfaces. TimesTen supports versions of these APIs that are both fully compliant with the standards and tuned for maximum performance in the TimesTen environment.

Unlike many other database systems, where the ODBC API support may be much slower than the proprietary interface, ODBC is a native TimesTen interface that operates directly with the database engine.

For more information, see *Oracle TimesTen In-Memory Database C Developer's Guide* and *Oracle TimesTen In-Memory Database Java Developer's Guide*.

### **Pro\*C/C++ Precompiler support**

TimesTen supports the Oracle Pro\*C/C++ Precompiler for C and C++ applications. You can use the precompiler with embedded SQL and PL/SQL applications that access a TimesTen database. Further, you can use your C or C++ program host variables in your embedded SQL or PL/SQL. For more information, see *Oracle TimesTen In-Memory Database C Developer's Guide*.

### **ODP.NET support**

Oracle Data Provider for .NET (ODP.NET) is an implementation of the Microsoft ADO.NET interface. ODP.NET support for TimesTen provides fast and efficient ADO.NET data access from .NET and C# client applications to a TimesTen in-memory database.

For more information, see *Oracle Data Provider for .NET Oracle TimesTen In-Memory Database Support User's Guide*.

### **Open source programming languages**

TimesTen supports open source programming languages, Python and Node.js, to access any TimesTen database through the Oracle Database ODPI-C driver.

ODPI-C is an open source library from Oracle Corporation designed to simplify access to Oracle databases from a variety of programming languages. It is a layer on top of the Oracle Call Interface.

See the *Oracle TimesTen In-Memory Database Open Source Languages Support Guide* for more details.



## TTClasses

TimesTen C++ Interface Classes (TTClasses) is more convenient 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, see *Oracle TimesTen In-Memory Database TTClasses Guide*.

## Transactional ACID compliance

TimesTen supports transactions that provide atomic, consistent, isolated and durable (ACID) access to data. All operations on a TimesTen database, even those that do not modify or access application data, are executed within a transaction. For more information, see "[Transaction isolation](#)" on page 4-1, "[Durability](#)" on page 1-5 in this guide and "Transaction Management" in the *Oracle TimesTen In-Memory Database Operations Guide*.

## Durability

A database in TimesTen is persistent across power failures and crashes. TimesTen accomplishes durability by periodically writing to disk:

- Changes made by transactions through transaction log files.

Transaction log records are written to disk asynchronously or synchronously 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 enables extremely high throughput with minimal exposure. In cases where data integrity must be preserved, such as securities trading, TimesTen ensures complete durability, with no loss of data. See "[Transaction logging](#)" on page 6-1 and "[When are transaction log files deleted?](#)" on page 6-2 for more details.

- All data through checkpoint files.

TimesTen maintains the disk-resident version of the database with a checkpoint operation that takes place in the background and has very little impact on database applications. This operation is performed automatically. TimesTen maintains two checkpoint files in case a failure occurs mid-checkpoint. Checkpoint files should reside on disks separate from the transaction logs to minimize the impact of checkpointing on application activity. See "[Checkpointing](#)" on page 6-2 for more details.

### Durability for TimesTen Classic

TimesTen Classic uses the transaction log 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 Database tables
- Enable applications to detect changes to tables (using the XLA API)

### Durability for TimesTen Scaleout

In TimesTen Scaleout, the data in your database is distributed into elements. Each element keeps its own checkpoint and transaction log files. As a result, the data stored

in each element is independently durable. Each instance in a grid manages one element of a database. In the event of a failure, an instance can automatically recover the data stored in its element from the checkpoint and transaction logs files while the remaining instances continue to service applications.

TimesTen Scaleout also enables you to keep multiple copies of your data to increase durability and fault tolerance.

You can change the durability settings of a database according to your performance and data durability needs. For example, you may choose if data is flushed to disk with every commit or periodically in batches in order to operate at a higher performance level.

See "Durability settings" in the *Oracle TimesTen In-Memory Database Scaleout User's Guide* for more details.

## Performance through query optimization

As with any mainstream RDBMS, a cost-based optimizer automatically determines the fastest way to process queries and transactions. TimesTen has a cost-based query optimizer that chooses the best query execution plan based on factors such as the presence of indexes, the cardinality of tables, and the presence of `ORDER BY` clauses in the query.

The method that the TimesTen optimizer analyzes SQL statements to minimize performance costs is different than in traditional 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 range and hash indexes. The query optimizer also uses two types of join methods (nested-loop and merge-join). The optimizer can create temporary indexes as needed. The optimizer accepts hints that provide applications the decision on whether to make tradeoffs between factors, such as temporary space usage and performance.

See [Chapter 5, "Query Optimization"](#) for more information about the query optimizer and indexing techniques.

## Concurrency

TimesTen provides full support for shared databases. Options are available so users can choose the optimum balance between response time, throughput and transaction semantics for an application.

Read-committed isolation provides nonblocking operations and is the default isolation level. For databases with extremely strict transaction semantics, serializable isolation is available. These isolation levels conform to the ODBC standard 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.

See [Chapter 4, "Concurrent Operations"](#) for more information about managing concurrent operations in TimesTen.

## Database character sets and 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. TimesTen Cache supports some database character sets that are compatible with the Oracle Database. The connection character set can be defined to enable an application running in a different encoding to communicate to a 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 "Globalization Support" in *Oracle TimesTen In-Memory Database Operations Guide*.

## Database connectivity

TimesTen supports direct and client/server connection options so that users can choose the best tradeoff between performance and functionality for their applications.

- A *direct connection* connects an application to a database where both exist on the same system.
- A *client/server connection* accommodates connections from remote client machines to databases across a network.

From an application's perspective, the TimesTen API is identical whether it is a direct connection or a client/server connection.

See "[TimesTen connection options](#)" on page 3-5 for overall information on connection options. See "Database connections" and "Connecting to a database" in the *Oracle TimesTen In-Memory Database Scaleout User's Guide* for details specific to TimesTen Scaleout.

## Load data from an Oracle database into a TimesTen table

You can load the results of a SQL query from a back-end Oracle database into a single table on TimesTen. TimesTen provides tools that execute a user-provided `SELECT` statement on the Oracle database and load the result set into a table on TimesTen.

TimesTen provides the following methods to accomplish these tasks:

- Use TimesTen Cache, which provides the connection and transfer of data between an Oracle database and a TimesTen database. You can cache Oracle Database data in a TimesTen database within cache groups. A cache group in a TimesTen database can cache a single Oracle Database table or a group of related Oracle Database tables.
- The `ttIsql` utility, an interactive SQL utility, provides the `createandloadfromoraquery` command that, once provided the TimesTen table name and the `SELECT` statement, automatically creates the TimesTen table, executes the `SELECT` statement on the Oracle database, and loads the result set into the TimesTen table.
- The `ttTableSchemaFromOraQueryGet` built-in procedure evaluates the user-provided `SELECT` statement to generate a `CREATE TABLE` statement that can be executed to create a table on TimesTen, which would be appropriate to receive the

result set from the `SELECT` statement. The `ttLoadFromOracle` built-in procedure executes the `SELECT` statement on the Oracle database and loads the result set into the TimesTen table.

## System monitoring

You can use system tables, system views, and the `ttStats` utility for system monitoring and reporting for both TimesTen Classic and TimesTen Scaleout. The following sections describe system monitoring and reporting tools:

- [System monitoring through system tables and views](#)
- [System monitoring for TimesTen Classic](#)
- [System monitoring for TimesTen Scaleout](#)

### System monitoring through system tables and views

TimesTen stores metadata (information about the contents of your database) in system tables in your database. TimesTen also provides system views that allow you to use SQL to query information.

Use the `ttIsql alltables` to display a list of all system and user tables. Use the `ttIsql allviews` command to display a list of all system and user views. See "ttIsql" in the *Oracle TimesTen In-Memory Database Reference* for more information on the `ttIsql` utility.

See "System Tables and Views" in the *Oracle TimesTen In-Memory Database System Tables and Views Reference* for full details.

### System monitoring for TimesTen Classic

Users have the following options for system monitoring and reporting.

- You can use the `ttStats` utility to monitor and display database performance metrics. It generates HTML reports from performance snapshots at user-specified intervals.

---

**Note:** System statistics are collected and stored in the `SYS.SYSTEMSTATS` table; the information is used by various TimesTen utilities and monitoring facilities.

---

- You can use the Oracle Enterprise Manager System Monitoring Plug-In for TimesTen In-Memory Database that provides a graphical user interface to monitor database metrics and performance. You can also view reports on the collected metric information, which is useful in diagnosing and troubleshooting database performance issues.
- You can use the `ttStatsConfig`, `ttSQLCmdCacheInfo`, and `ttSQLExecutionTimeHistogram` built-in procedures to measure and display execution time statistics for SQL operations to determine the performance of SQL statements.

### System monitoring for TimesTen Scaleout

Users have the following options for system monitoring and reporting for your grid and database:

---

**Note:** See "Monitoring TimesTen Scaleout" in the *Oracle TimesTen In-Memory Database Scaleout User's Guide* for full details on system monitoring for TimesTen Scaleout.

---

- You can use the `ttStats` utility enables you to monitor database metrics (statistics, states, and other information) or take and compare snapshots of metrics. It generates HTML reports from performance snapshots at user-specified intervals. For more information, see "Using the `ttStats` utility" in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

---

**Note:** System statistics are collected and stored in the `SYS.SYSTEMSTATS` table; the information is used by various TimesTen utilities and monitoring facilities.

---

- You can collect various logs from every host that is part of your grid. These logs are useful for troubleshooting errors that you may encounter while using your grid or database. See "Collecting grid logs" in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.
- You can monitor the management instances. See "Monitoring the management instances" in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.
- You can use SQL Developer to create, manage, and explore a grid and its components. Additionally, you can also browse, create, edit and drop particular database objects; run SQL statements and scripts; manipulate and export data; view and create reports; and view database metrics. For more information, see "Using SQL Developer to work with TimesTen Scaleout" in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

## Administration and utilities

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.

TimesTen supports typical database utilities such as the following:

- Interactive SQL (`ttIsql`).
- Migrate (`ttMigrate`) provides a way to quickly move data between different major versions of TimesTen.
- Many administrative activities are available through SQL statements and system views. TimesTen also uses SQL statements to set up materialized views.

TimesTen Scaleout provides database management through the `ttGridAdmin` utility.

TimesTen Classic also supports typical database utilities such as the following:

- Backup and restore (`ttBackup` and `ttRestore`).
- Copy data between different database systems.
- Using SQL statements to set up replication and caching from an Oracle database.

TimesTen built-in procedures and C language functions enable programmatic control over TimesTen operations and settings. TimesTen command-line utilities enable 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.

See [Chapter 9, "TimesTen Administration"](#) for more information on TimesTen administration.

## TimesTen Scaleout specific features

In addition to the TimesTen general features, the following features are supported within TimesTen Scaleout

- [High availability features for TimesTen Scaleout](#)
- [Transparent data distribution](#)
- [Online elastic scalability](#)
- [Automatic high availability through K-Safety](#)
- [Single point for administration and monitoring](#)

### High availability features for TimesTen Scaleout

The TimesTen Scaleout architecture provides high performance and supports high availability.

TimesTen Scaleout enables you to create a grid of interconnected instances running on one or more hosts. You can create a database hosted in the grid and the data stored in that database is distributed across the instances of the grid. Each instance contains one piece (an element) of every database hosted in the grid.

[Figure 1–2](#) highlights this portion of the architecture.

- TimesTen Scaleout provides high availability by enabling you to specify that the system maintain more than one copy of every item of data within a database, known as K-safety. The number of copies of data is referred to as the *k*-factor. Your choice to maintain more than one copy protects you from data loss in the event of a single failure. So a grid with  $k=1$  does not provide high availability (as there is only one copy of the data); however, a grid with  $k > 1$  provides high availability.

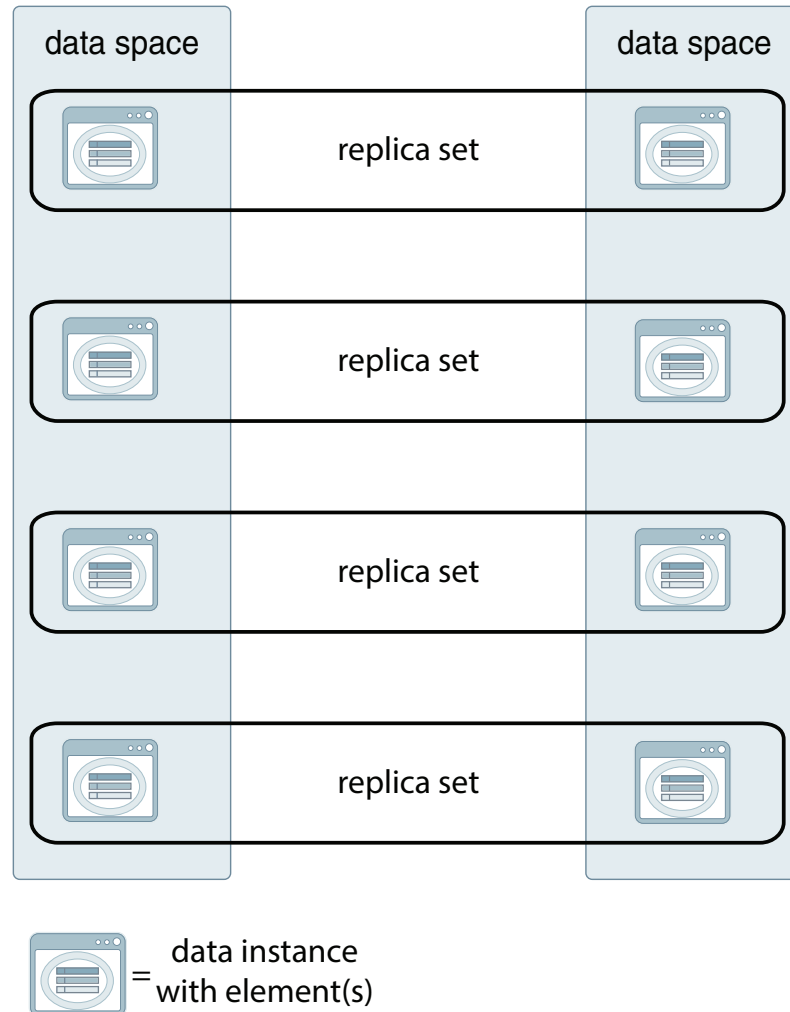
Each copy of the database is logically contained within a data space, which logically groups the instances that contain one entire copy of a database. There are always as many data spaces as the grid *k*-factor.

- TimesTen Scaleout spreads the work for the database across data instances in parallel, which computes the results of your SQL statements faster. the data in your database is distributed into elements.
- Your application can connect to any data instance in the grid and transparently access all of the data in the database without having to know where specific data is located.
- A replica set is a set of database elements with identical contents (the multiple copies resulting from the specified *k*-factor). Each replica set always contains *k* elements. The separate elements within a replica set are located across separate data spaces. If one element fails, your connection can be rerouted to the other element in the replica set, if it is still available.
- You can add or remove instances from your grid to:
  - Expand or shrink the storage capacity of your database as necessary.

- Expand or shrink the computing resources of your database to meet the performance requirements of your applications.

Figure 1–2 shows a database within the grid where the  $k$ -factor is set to 2.

**Figure 1–2 Performance and high availability architecture within TimesTen Scaleout**



For more details, see "[Architectural overview of TimesTen Scaleout](#)" on page 3-2 in this guide and "TimesTen Scaleout architecture" in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

## Transparent data distribution

While TimesTen Scaleout distributes your data across multiple instances, applications do not need to know how data is distributed. When an application connects to any instance in the grid, it has access to all of the data of the database without having to know the location of specific data.

Knowledge about the distribution of data is never required in TimesTen Scaleout, but it can be used to tune the performance of your application. You can use this knowledge to exploit locality where possible.

See "Data transparency" in the *Oracle TimesTen In-Memory Database Scaleout User's Guide* for more details.

## Online elastic scalability

TimesTen Scaleout enables you to add or remove instances in order to control both performance and the storage capacity of your database. Adding instances improves the throughput of your workload by providing the additional computing resources of the hosts running those instances. If your business needs change, then removing instances (and their hosts) enables you to meet your targets with fewer resources.

See "Scalability" in the *Oracle TimesTen In-Memory Database Scaleout User's Guide* for more details.

## Automatic high availability through K-Safety

As described in "[TimesTen Scaleout specific features](#)" on page 1-10, TimesTen Scaleout provides high availability and fault tolerance when you have multiple copies of data located across separate hosts with K-safety (k). When one copy of the data is unavailable due to a software error, SQL statements are automatically redirected to the other copy of the data (if possible). In the meantime, TimesTen Scaleout synchronizes the data on the failed system with the rest of the database.

See "High availability and fault tolerance" in the *Oracle TimesTen In-Memory Database Scaleout User's Guide* for more details.

## Single point for administration and monitoring

You do not need to log onto every host within a grid in order to perform management activities. Instead, you conduct all management activity from a single management instance using the ttGridAdmin utility. The ttGridAdmin utility is the main utility you use to define, deploy, and check on the status of each database.

You can also use the ttGridRollout utility or the Oracle SQL Developer GUI (both of which use the ttGridAdmin utility under the covers to execute all requests) to facilitate creating, deploying, and managing your grid.

See "Centralized management" in the *Oracle TimesTen In-Memory Database Scaleout User's Guide* for more details.

## TimesTen Classic specific features

In addition to the TimesTen general features, the following features are supported within TimesTen Classic.

- [High availability features for TimesTen Classic](#)
- [Data replication between servers](#)
- [Transaction Log API \(XLA\)](#)
- [Automatic data aging](#)

## High availability features for TimesTen Classic

TimesTen Classic has several features that provide performance and high availability. [Figure 1–3](#) highlights a few of these features.

- TimesTen Classic *replication* enables you to achieve near-continuous availability or workload distribution by sending updates between two or more hosts. An active standby pair replication scheme includes an active database, a standby database, and optional read-only subscriber databases. In [Figure 1–3](#), the active and standby



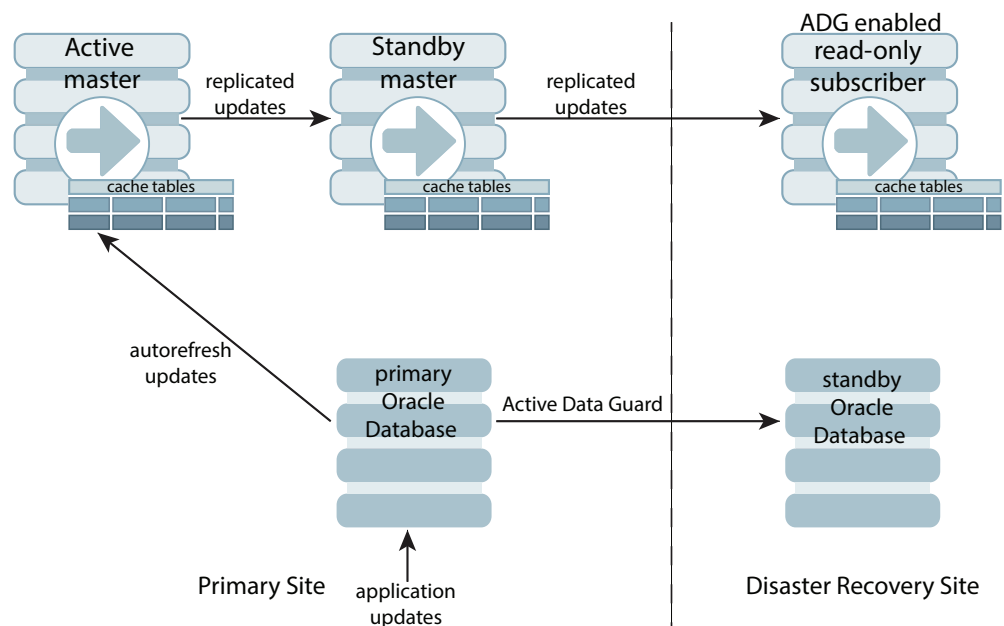
databases can be located at a primary site, while a read-only subscriber is located at a disaster recovery site.

- TimesTen Cache enables you to cache portions of an Oracle database in a TimesTen database in tables within a *cache group*. In [Figure 1–3](#), tables cached on the Oracle database can be cached as read-only copies in TimesTen. TimesTen Cache automatically propagates changes from the Oracle tables to the cached copy in the TimesTen database.

In [Figure 1–3](#), there are two Oracle databases set up within an Active Data Guard (ADG) configuration: one at the primary site and the other at the disaster recovery site. ADG propagates any data changes made on the primary Oracle database to the standby Oracle database. When used in conjunction with TimesTen Cache, the integration of ADG within TimesTen ensures that only changes that have been propagated to the standby Oracle database are refreshed to the cache. In the event of a failure of the primary Oracle database, we switch over to the standby Oracle database. After which, the TimesTen Cache also switches over and continues with automatic refresh while remaining consistent with the Oracle database.

There are other recovery methods for if the standby Oracle database fails or if the entire primary site fails. See "Recovery after failure when using asynchronous Active Data Guard" in the *Oracle TimesTen Application-Tier Database Cache User's Guide* for more details.

**Figure 1–3 Performance and high availability configuration for TimesTen Classic**



See ["Cached data"](#) on page 3-7 and ["Replication"](#) on page 3-7 for more details on TimesTen Cache and replication.

## Data replication between servers

TimesTen Classic provides replication schemes that enable 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. See ["Data Replication within TimesTen Classic"](#) on page 6-4 for more

details.

## Transaction Log API (XLA)

TimesTen Classic provides an API that enables applications to monitor update activities in order to generate actions outside the database. This capability is provided by the Transaction Log API (XLA), which enables applications to monitor update records as they are committed 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. This API is supported for C, Java (JMS/XLA) and C++ (TTClasses).

TimesTen Classic provides materialized views that can be used with XLA to enable notification of events described by SQL queries. The primary purpose of XLA is to be a high performance, asynchronous alternative to triggers.

For more information, see ["Detect transaction modifications with the Transaction Log API"](#) on page 8-1 and *Oracle TimesTen In-Memory Database C Developer's Guide*.

## 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 TimesTen Cache data 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 *Oracle TimesTen Application-Tier Database Cache User's Guide*.

## TimesTen Cache specific features

The TimesTen Application-Tier Database Cache (TimesTen Cache) provides all of the same features as TimesTen Classic as well as creating an updatable cache for the Oracle database data. It offloads computing cycles from Oracle databases and enables responsive and scalable applications. TimesTen Cache loads a subset of the Oracle database tables into a TimesTen database. It can be configured to propagate updates in both directions and to automate passthrough of SQL requests for uncached data. It automatically resynchronizes data after failures.

TimesTen Cache provides multiple types of cache groups that cache Oracle database tables with automatic data synchronization.

See [Chapter 7, "TimesTen Cache"](#) for more details.

---

## Using TimesTen

TimesTen can be used as:

- The *primary database* for applications, where all data needed by the applications resides in the TimesTen database.
- A *service* 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.

Users can take advantage of the integrated high performance parallel data replication capability to add high availability and disaster recovery capabilities to applications.

- [Uses for TimesTen Scaleout](#)
- [Uses for TimesTen Classic](#)
- [Uses for TimesTen Cache](#)

### Uses for TimesTen Scaleout

TimesTen Scaleout is used for multiple-instance environments and distributed databases. TimesTen Scaleout provides fast access, fault tolerance, and high availability for in-memory data.

With TimesTen Scaleout, you can:

- Create a grid that is a set of interconnected instances installed on one or more hosts.
- Maintain one or more copies of your data. Your choice to maintain more than one copy protects you from data loss in the event of a single failure.
- Distribute the data of each database across the instances in the grid in a highly available manner using a shared-nothing architecture.
- Connect applications to your database with full access to all the data, no matter what the distribution of the data is across the database.
- Add or remove instances from your grid to:

- Expand or shrink the storage capacity of your database as necessary.
- Expand or shrink the computing resources of your database to meet the performance requirements of your applications.

See the *Oracle TimesTen In-Memory Database Scaleout User's Guide* for full product features and details.

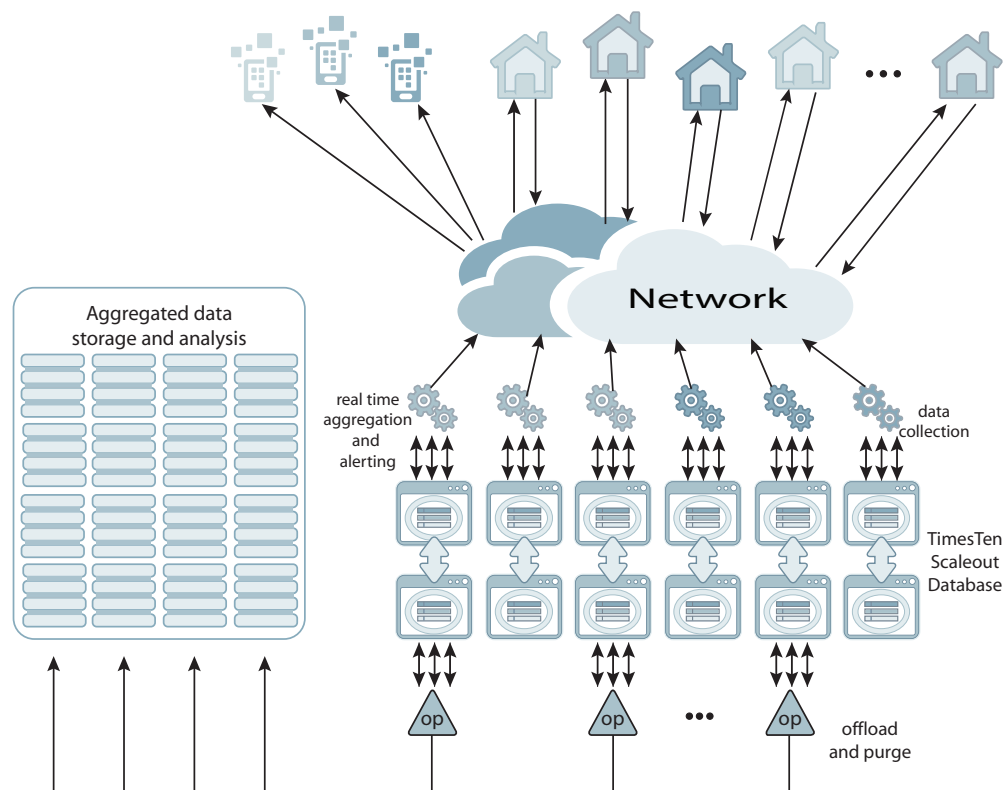
## TimesTen Scaleout application scenario

A typical use case scenario for TimesTen Scaleout is an Internet-of-Things (IoT) application, which is used to:

- Capture semi real-time data (sensor readings, geolocation data, telemetry, and so on) from a vast array of connected devices.
- Aggregate and analyze the data in real-time (for alerting purposes) before offloading the data to other systems for further reporting and analysis.

The Internet is pervasive as more and more devices are connected (such as smart homes, smart phones, tablets, cars, domestic appliances, utility meters, exercise devices, GPS trackers, RFID tags, and so on). Many services need to efficiently capture and process huge volumes of data generated in a real-time by these devices.

**Figure 2–1 Using TimesTen Scaleout for an Internet-of-Things application**



The TimesTen Scaleout application example illustrates how a vendor of smart home technology uses TimesTen Scaleout to capture sensor data (temperature, humidity, CO2 levels, device events). Once captured, the data is aggregated and analyzed in order to generate real-time alerts for anomalous conditions and then the aggregated data is offloaded to other systems for archival storage, in depth analysis, and so on. Once offloaded, the data is purged from the database.

- **Capture:** Sensor data collected from the smart home devices is sent over the Internet to the data collection applications running in the same vicinity as the database. These applications insert and update data in the database based on the received data.
- **Aggregate and analyze:** Real-time aggregation and alerting applications perform initial processing on the data as it arrives. If the application detects any important events, then the application sends alerts to the smart home devices, user's smartphone applications, and so on.

Offload and purge applications move historical aggregated data that is no longer needed online to storage, for long term storage and analysis. Once moved into storage, the aggregated data and underlying raw data is purged from the database.

TimesTen Scaleout is ideal for this type of application:

- High throughput and low latency enables TimesTen Scaleout to cope with a heavy read-write workload.
- Fault tolerant architecture enables TimesTen Scaleout to deliver continuous uptime even in the event of hardware or software failures.
- Elastic scalability features enable the storage and processing capacity of the database to dynamically increase while remaining online.
- Support for a broad range of industry standard languages and APIs makes building different application components easy for developers.

## Uses for TimesTen Classic

TimesTen Classic:

- Can be used for a single-instance environment and database.
- Can be used to create one or more in-memory, SQL relational, ACID-complaint databases.
- Provides a memory-optimized relational database that gives applications the best response time and high throughput.
- Supports high availability for the in-memory database through transactional replication.

## TimesTen Classic application scenario

This section demonstrates a TimesTen Classic application scenario with a Quote service application, which illustrates how TimesTen Classic can be integrated as part of a data management solution.

A financial services company is adding a quote and news service to its online trading facility. The quote service reads an incoming news wire from a major market data vendor and make a subset of the data available to trading applications that manage the automated trading operations for the company. The company plans to build an infrastructure that can accommodate future expansion to provide quotes, news, and other trading services to retail subscribers.

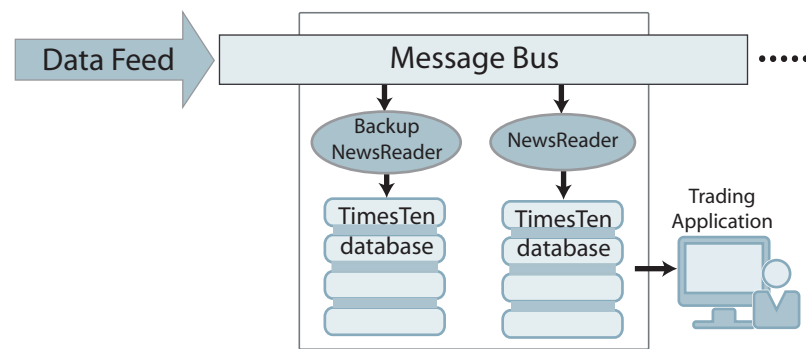
The quote service application uses the database to store stock quotes from a data feed for access by program trading applications. Quote data is collected from the data feed and published on a message bus. The data is read from the message bus and stored in the database, where it is accessed by the program trading applications.

The quote service includes a *NewsReader* process that reads incoming data from a 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 database. In this way, the message bus is used to fork incoming data to two databases for redundancy. In this scenario, forking the data from the message bus is more efficient than using TimesTen replication.

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 if a failure occurs. The current load requires four *NewsReader* pairs, but more *NewsReader* pairs can be added in the future to scale the service to deliver quotes to other types of clients over the Web or cellular phone.

Figure 2–2 shows the configuration for capturing data from a message bus and feeding it to *NewsReaders*.

**Figure 2–2 Capturing feed data from a message bus**

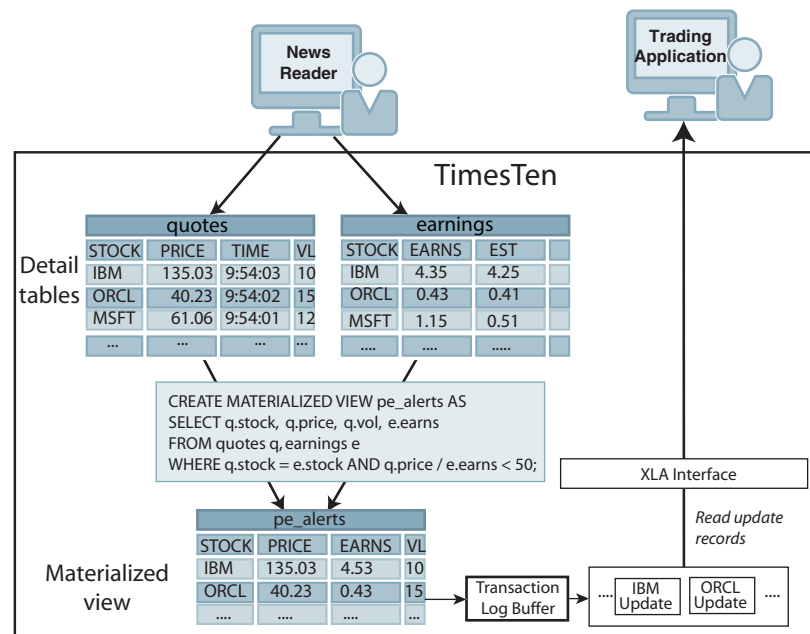


As shown in Figure 2–3, the *NewsReader* updates stock price data in a *QUOTES* table in the database. Less dynamic earnings data is updated in an *EARNINGS* table. The *STOCK* columns in the *QUOTES* and *EARNINGS* tables are linked through a foreign key relationship.

The purpose of the trading application is to track only those stocks with PE ratios below 50, then use 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 Transaction Log API (XLA) to monitor the transaction log for updates to the stocks of interest. See ["Detect transaction modifications with the Transaction Log API"](#) on page 8-1 for more information.

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 *EARNINGS* 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 receives alerts only for those stocks that meet its trading criteria.

Figure 2-3 Using materialized views and XLA



## Uses for TimesTen Cache

TimesTen Cache is supported in TimesTen Classic and can be used as:

- **A read-only cache.** Oracle Database data can be cached in an TimesTen Cache read-only cache group. Read-only cache groups are automatically refreshed when the Oracle database 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 Database data can be cached in TimesTen Cache updatable cache groups. Transactions on the cache groups can be committed synchronously or asynchronously to the associated Oracle database tables.
- **A data manager** for specific tasks in an overall workflow in collaboration with a disk-based RDBMS like the Oracle Database. For example, a phone billing application may capture and store recent call records in the TimesTen database while storing information about customers, their billing addresses and credit information in an Oracle database. It can also age and keep archives of all call records in the Oracle database. Thus, the information that requires immediate access is stored in the TimesTen database while the information needed for longer-term analysis, auditing, and archival is stored in the Oracle database.
- **A scalable cache.** Oracle Database 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.
- **An integrated method for high availability and disaster recovery.** TimesTen replication is tightly integrated with TimesTen Cache features enabling customers to deploy highly available cache configurations. TimesTen Cache also integrates with Oracle Database high availability features such as Oracle RAC and Oracle Data Guard to provide seamless cross-tier high availability and disaster recovery capabilities.

See [Chapter 7, "TimesTen Cache"](#) for more details on TimesTen Cache. See ["Replication"](#) on page 3-7 for more information on replication.

## TimesTen Cache application scenario

This section demonstrates a TimesTen Cache application scenario with a Caller usage metering application, which illustrates how a TimesTen Cache can store metering data on the activities of cellular callers.

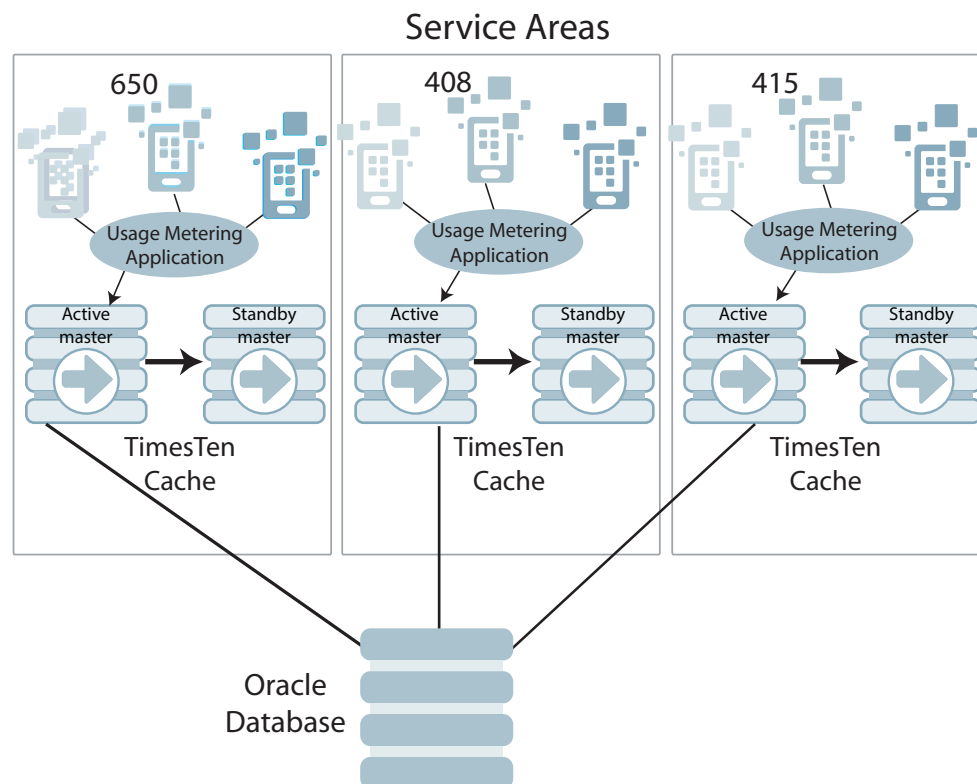
A wireless communications company has a usage metering application that keeps track of the duration of each cellular call and the services used. For example, if a caller makes a regular call, a base rate is applied for the duration of the call. If a caller uses special features (like roaming), extra 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. Thus, metering data is collected from multiple TimesTen databases distributed throughout a service area and archived in a central Oracle database for use by a central billing application.

The company uses TimesTen databases in an active standby pair 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 Oracle database. The company distributes multiple instances of the usage metering application and TimesTen Cache on individual nodes throughout its service areas. For maximum performance, each usage metering application connects to its local TimesTen database by an ODBC direct driver connection.

[Figure 2-4](#) shows the configuration.

**Figure 2-4 Distributed caching of usage data**





A usage metering application and TimesTen Cache are deployed on each node to handle the 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.

Transactions that impact revenue (inserts and updates) must be durable. To ensure data availability, each TimesTen Cache database is configured as an active-standby replicated pair using TimesTen replication.

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

In this scenario, the *CALLS* table is configured for write through caching; as data is inserted, updated, or deleted in the table in TimesTen, the changes are automatically propagated (in real-time) to the underlying Oracle Database. After the call records have been successfully propagated to the Oracle database, they are deleted from the TimesTen databases (but not from the Oracle database) by an automatic time-based aging process.



---

## TimesTen Architecture

This chapter includes the following topics:

- [Architectural overview](#)
- [Shared libraries](#)
- [Memory-resident data structures](#)
- [Database processes](#)
- [TimesTen connection options](#)
- [Checkpoint and transaction log files](#)
- [Cached data](#)
- [Replication](#)

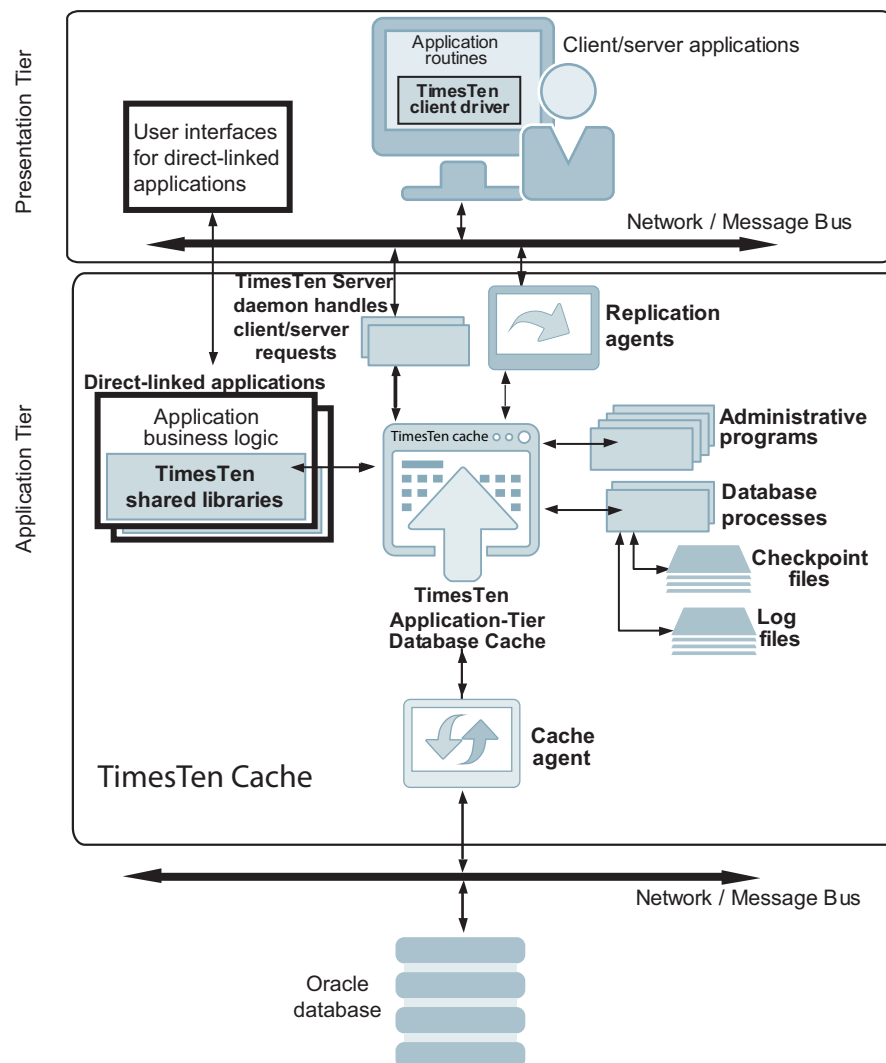
### Architectural overview

The architecture for TimesTen Classic and TimesTen Scaleout are described in the following sections:

- [Architectural overview of TimesTen Classic and TimesTen Cache](#)
- [Architectural overview of TimesTen Scaleout](#)

### Architectural overview of TimesTen Classic and TimesTen Cache

This section describes the architecture of TimesTen Classic and the TimesTen Application-Tier Database Cache (TimesTen Cache). The architecture of TimesTen Classic is the same as the architecture of the TimesTen Cache except that the Oracle database and cache agent are not included. [Figure 3–1](#) shows the architecture of the TimesTen Cache.

**Figure 3–1 TimesTen Application-Tier Database Cache architecture**

The architectural components include shared libraries, memory-resident data structures, database processes, and administrative programs. The application can connect to the TimesTen database by direct link and by client/server connections.

*Replication agents* receive information from master databases and send information to subscriber databases. The *cache agent* performs all asynchronous data transfers between cache groups in the TimesTen Cache and the Oracle database.

See "[TimesTen Architecture](#)" on page 3-1 for more details on these components.

## Architectural overview of TimesTen Scaleout

TimesTen Scaleout enables you to create a grid that is a set of interconnected instances installed on one or more hosts. TimesTen Scaleout enables you to distribute the data of a database across this grid.

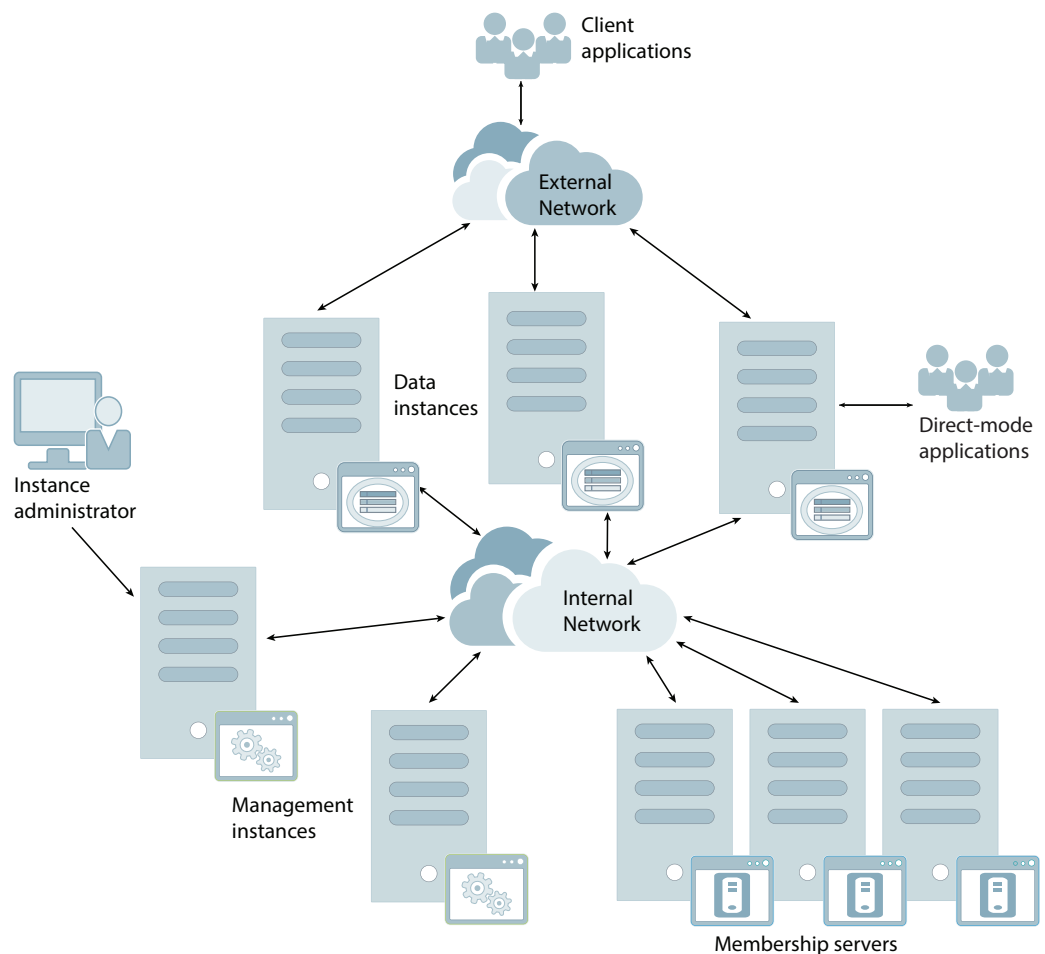
- You can create one or more in-memory, SQL relational, ACID-complaint databases.

The architectural components for each database include shared libraries, memory-resident data structures, database processes, and administrative programs. The application can connect to the TimesTen database by direct link and

by client/server connections. These components are described in subsequent sections.

- *Data instances* (in which data is contained and managed) execute SQL statements and PL/SQL blocks. A grid distributes the data within each database across data instances.
- You create one or two *management instances* through which the grid is managed. You manage all data instances through the active management instance.
- Set up a *membership service* to track which data instances are operational at any moment. The membership service consists of three or more *membership servers*.
- You can connect applications to your database with full access to all the data, no matter what the distribution of the data is across the database.

**Figure 3–2 TimesTen Scaleout Grid structure**



See "TimesTen Scaleout architecture" in the *Oracle TimesTen In-Memory Database Scaleout User's Guide* for full details of the TimesTen Scaleout architecture.

## 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 daemon processes to which applications connect, typically over a client/server network. Applications can also use a client/server connection to access a TimesTen database, though in most cases the best performance can be realized with a directly linked application. See ["TimesTen connection options"](#) on page 3-5 for more information.

## Memory-resident data structures

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

## Database processes

TimesTen initiates separate background processes to each database to perform operations:

- [TimesTen daemon](#)
- [Subdaemons](#)

### TimesTen daemon

The *TimesTen daemon* performs the following functions:

- Manage shared memory access
- Coordinate process recovery
- Keep management statistics on what databases exist, which are in use, and which application processes are connected to which databases
- Manage RAM policy
- Start replication processes (if requested), the TimesTen Server and the cache agent (if enabled).

The TimesTen daemon starts when TimesTen is installed. This main daemon runs in the background. The instance administrator must manually start and stop the daemon after each system reboot or the `root` user can start the daemon by executing the daemon startup script.

Application developers do not interact with the daemon directly. No application code runs in the daemon and application developers do not generally have to be concerned with it. Application programs that access TimesTen databases communicate with the daemon transparently using TimesTen internal routines.

See "Working with the TimesTen Daemon" in the *Oracle TimesTen In-Memory Database Operations Guide* for more information.

### Subdaemons

The main TimesTen daemon spawns *subdaemons* dynamically as they are needed. TimesTen uses subdaemons to perform the following:

- Manage databases
- Flush the transaction log buffer to disk

- Perform periodic checkpoints in the background for the active database
- Load the database into memory from a checkpoint file on disk
- Implement the data aging policies of various tables
- Detect and handle deadlocks
- Roll back transactions for abnormally terminated direct-mode applications
- Perform required background processing for the database
- Recover the database if it needs to be recovered after loading it into memory

See "Managing subdaemons" in the *Oracle TimesTen In-Memory Database Operations Guide* for more information.

## TimesTen connection options

TimesTen supports direct, driver manager, and client/server connection options so that users can choose the best tradeoff between performance and functionality for their applications. From an application's perspective, the TimesTen API is identical whether it is a direct connection, driver manager connection, or a client/server connection.

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

- [Direct connection](#)
- [Client/server connection](#)
- [Driver manager connection](#)

For more information about connecting to a TimesTen Classic database, see "Managing TimesTen Databases", "Working with the TimesTen Client and Server" and "Connecting to TimesTen with ODBC and JDBC drivers" in the *Oracle TimesTen In-Memory Database Operations Guide*.

For details on connecting to a grid using TimesTen Scaleout, see "Database connections" and "Connecting to a database" in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

### Direct connection

An application using a *direct connection* runs on the same system as the database. In a direct connection, the ODBC driver attaches the shared memory region of the TimesTen database into the application's process address space. 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 connection provides extremely fast performance and is the preferred way for applications to access the TimesTen database.

Applications that use direct connections must run in the environment of the TimesTen instance that provides the database. When using TimesTen Scaleout applications can run in any data instance.

The ODBC direct driver is a library of ODBC and TimesTen routines that implement the database engine used to manage the database. Applications using other APIs use the appropriate API library, which then in turn uses the TimesTen ODBC driver.

## Client/server connection

The *TimesTen client driver* and *TimesTen Server daemon* processes accommodate connections from remote client machines to databases across a network.

- Applications on a client machine issue ODBC, JDBC or OCI calls. These calls access a local ODBC client driver that communicates with the TimesTen instance. The TimesTen instance, in turn, issues native ODBC requests to the ODBC direct driver to access the TimesTen database.
- The TimesTen Server daemon manages incoming connections from remote client applications. Its behavior is much like the TimesTen daemon for local applications, but it allocates a server child process to access the database on behalf of the client.

Traditional client/server access is supported for functions such as reporting, or when a large number of application-tier platforms must share access to a common in-memory database.

All exchanges between client and server are sent over a TCP/IP connection. If the client and server reside on separate hosts in the internal 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 using direct connections.

- Client applications communicate with the TimesTen instance that provides the database.
- With TimesTen Scaleout, an application using a client/server connection may run on a data instance or on any host with access to the external network. Client applications are automatically connected to a working data instance. For more information, see "Database connections" in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

For more information on connecting using ODBC and JDBC interfaces, see "Working with TimesTen Databases in ODBC" in the *Oracle TimesTen In-Memory Database C Developer's Guide* and "Working with TimesTen Databases in JDBC" in the *Oracle TimesTen In-Memory Database Java Developer's Guide*.

## Driver manager connection

Applications can connect to the TimesTen database 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 enables applications to be written to operate independently of the database and to use interfaces that are not directly supported by TimesTen. You can use a driver manager with either direct or client/server connections, which enables a single process to have both direct and client connections to the database.

You can link a client application directly with the TimesTen Client ODBC driver or with the Windows ODBC driver manager to access the TimesTen database. Driver managers for Linux or UNIX systems are available as open-source software as well as from third-party vendors. Driver manager connections can provide support for ODBC applications written for a different ODBC version or for multiple RDBMS products with ODBC interfaces. Note that there are performance considerations in using a driver manager.

See "Connecting to TimesTen with ODBC and JDBC drivers" in the *Oracle TimesTen In-Memory Database Operations Guide* for more information.



## Checkpoint and transaction log files

*Checkpoint files* contain an image of the database in persistent storage on disk. TimesTen uses dual checkpoint files in case the system fails while a checkpoint operation is in progress. Changes to a database are captured in transaction log files that are periodically written to persistent storage on disk. If a database needs to be recovered, TimesTen merges the most recent database checkpoint file on disk with the completed transactions that are still in the transaction log files.

See ["Data Availability and Integrity"](#) on page 6-1 for more information.

## Cached data

When the TimesTen Cache is used to cache portions of an Oracle database in a TimesTen database, a *cache group* is created to hold the cached data.

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 table can contain all rows and columns or a subset of the rows and columns in the related Oracle database table. You can create or modify cache groups by using SQL statements or by using Oracle SQL Developer. Cache groups support these features:

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

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

Changes that originate in the Oracle database tables are refreshed into the cache by the *cache agent*.

See ["TimesTen Cache"](#) on page 7-1 for more information.

## Replication

TimesTen Classic *replication* enables you to achieve near-continuous availability or workload distribution by sending updates between two or more hosts. A master host is configured to send updates and a subscriber host is configured to receive them. A host 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.

TimesTen recommends the *active standby pair* replication scheme configuration for highest availability. It is the only replication configuration that you can use for replicating TimesTen Cache. An active standby pair includes an active database, a standby database, and optional read-only subscriber databases.

When replication is configured, a *replication agent* is started for each database. Each replication agent can 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 host, the replication agent updates the database through an efficient internal interface that avoids the overhead of the SQL layer.

See "[Data Replication within TimesTen Classic](#)" on page 6-4 for more information about replication configurations.

---

## Concurrent Operations

When a database is accessed by multiple applications, there must be a way to coordinate concurrent changes to data with reads 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](#)
- [Locking levels](#)

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

### Transaction isolation

Transaction isolation provides an application with the appearance that the system processes one transaction at a time, even though there are concurrent connections to the database. Applications can use the `Isolation` general 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 supports two isolation levels:

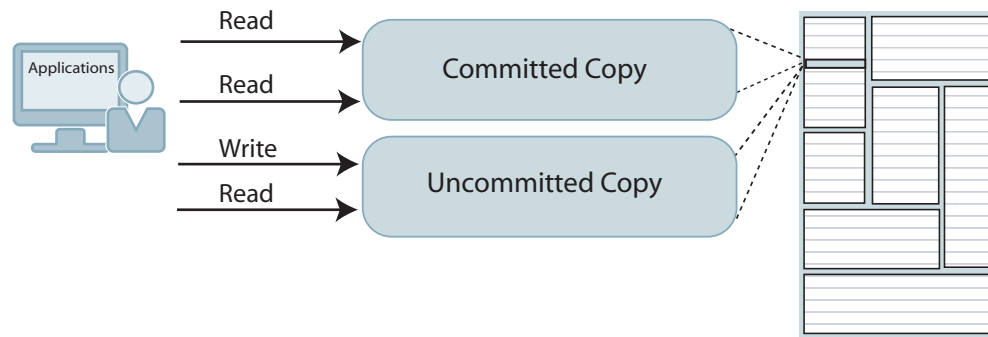
- [Read committed isolation](#)
- [Serializable isolation](#)

### 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 nonblocking for queries. 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 some applications read a committed copy of the data 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 block both readers and writers.

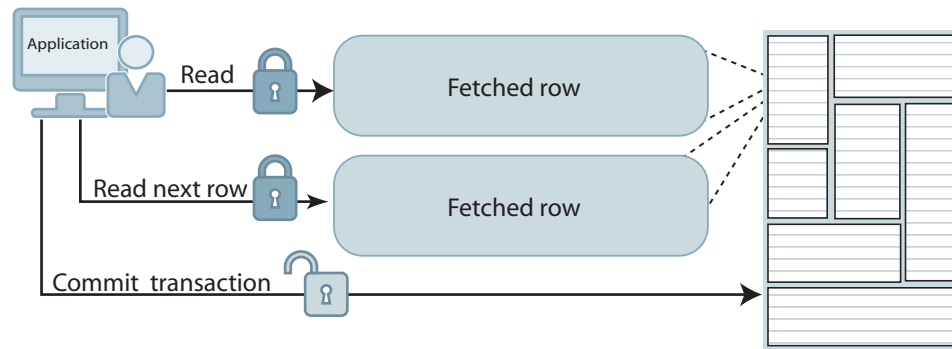
Read committed isolation does acquire read locks as needed during materialized view maintenance to ensure that views are consistent with their detail tables. These locks are not held until the end of the transaction but are instead released when materialized view maintenance has been completed.

## 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. As a result, a row that has been read by one transaction cannot be updated or deleted by any other transaction until the first transaction completes. Similarly, a row that has been inserted, updated, or deleted by a transaction cannot be accessed in any way by any other transaction until the first transaction completes.

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 must modify the data that is read by another transaction may encounter lock timeouts because read locks are held until the transaction commits.

## Locking levels

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 for all database accesses.

- A shared lock enables the associated resource to be shared, depending on the operations involved (that is, you cannot alter or drop a row, table, or view if it is being shared). Several transactions can acquire shared locks on the same resource.
- An exclusive lock ensures that there is no more than one active transaction in the database at any given time. This lock prevents the resource from being shared and is obtained to modify data. The first transaction to lock a resource exclusively is the only transaction that can alter the resource until the exclusive lock is released when the transaction completes.

Serializable transactions acquire shared 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:

- [Row-level locking](#)
- [Table-level locking](#)
- [Database-level locking](#)

## 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.

Applications can use the `LockLevel` general connection attribute, the `ttLockLevel` built-in procedure or the `ttOptSetFlag` built-in procedure to manage row-level locking.

---

---

**Note:** See "ttLockLevel" and "ttOptSetFlag" in the *Oracle TimesTen In-Memory Database Reference* for more information.

---

---

## 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.

---

---

**Note:** See "ttOptSetFlag" in the *Oracle TimesTen In-Memory Database Reference* for more information.

---

---

Table locks can reduce throughput, so they should be used only when a substantial portion of the table must 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.

## 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 bulk loading, 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.

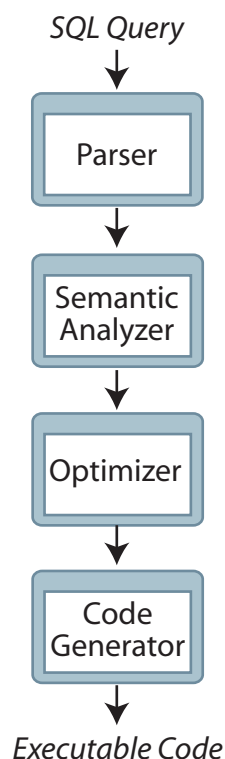
Use the `LockLevel` general connection attribute or the `ttLockLevel` built-in procedure to implement database-level locking. See "ttLockLevel" in the *Oracle TimesTen In-Memory Database Reference* for more information.

---

## Query Optimization

TimesTen has a cost-based query optimizer that ensures efficient data access by automatically determining the most efficient way to execute a given query by considering possible query plans. Optimization is performed in the third stage of the compilation process. The stages of compilation are shown in [Figure 5-1](#).

**Figure 5-1** *Compilation stages*



TimesTen invokes the optimizer for SQL statements when more than one execution plan is possible. The optimizer chooses what it thinks is the optimum plan. This plan persists until the statement is either invalidated or dropped by the application.

The optimizer determines the cost of a plan based on:

- Table and column statistics
- Metadata information (such as referential integrity, primary key)
- The presence or absence of indexes. Index choices, including the creation of temporary indexes.

- The volume of data
- The number of unique values
- The selectivity of predicates
- Scan methods (full table scan, rowid lookup, range index scan, hash index lookup)
- Join algorithm choice (nested loop join, nested loop join with indexes, or merge join)

This chapter includes the following topics:

- [Optimization time and memory usage](#)
- [Statistics](#)
- [Optimizer hints](#)
- [Indexes](#)
- [Scan methods](#)
- [Join methods](#)
- [Optimizer plan](#)

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

## Optimization time and memory usage

The optimizer is designed to generate the best possible plan within reasonable time and memory constraints. Because of the size of the search space (which grows exponentially with the number of tables in a join), the optimizer cannot always choose an optimal plan for every query. Instead, the goal of the optimizer is to choose a good plan from among a set of plans generated by using strategies for finding the most promising areas within the search-space of plans. Since query optimization is relatively expensive relative to execution, 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 5-3 to disable the use of temporary indexes and tables in order to create plans that trade maximum performance for reduced 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 and unique constraints within a table, and the size and configuration of any existing indexes.

Table-level and column-level statistics are stored in the `SYS.TBL_STATS` and `SYS.COL_STATS` tables. These tables are populated when an application calls the `ttOptUpdateStats` or `ttOptEstimateStats` built-in procedures, or uses the `ttIsql statsupdate` or `statsestimate` commands.

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

You can apply *hints* to pass instructions to the TimesTen query optimizer. The optimizer considers these hints when choosing the best execution plan for your query. You can apply hints as follows:

- To apply a hint only for a particular SQL statement, use a statement level optimizer hint.
- To apply a hint for an entire transaction, use a transaction level optimizer hint with the appropriate TimesTen built-in procedure.
- To apply a hint for every SQL statement in the user application, use a connection level optimizer hint with the `OptimizerHint` connection attribute.

The order of precedence for optimizer hints is statement level hints, transaction level hints and then connection level hints.

See "Use optimizer hints to modify the execution plan" in the *Oracle TimesTen In-Memory Database Operations Guide* for more information on all optimizer hints.

## Indexes

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

TimesTen uses these types of indexes:

TimesTen uses two types of indexes, each of which can be defined as unique or not unique (the default):

- *Hash index*: Hash indexes are useful for finding rows with an exact match on one or more columns. You create a hash index with the `CREATE HASH INDEX SQL` statement. In general, hash indexes are faster than range 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 range indexes.
- *Range index*: Range indexes are useful for finding rows with column values within a range specified as an equality or inequality. Range indexes can be created over one or more columns of a table. You create a range index with the `CREATE INDEX SQL` statement.

For more information about indexing, see "Understanding indexes" in *Oracle TimesTen In-Memory Database Operations Guide*. Also, see descriptions for the "CREATE INDEX" statement in *Oracle TimesTen In-Memory Database SQL Reference*.

The Index Advisor tool can be used to recommend a set of indexes that could improve the performance of a specific SQL workload. For more details, see "Using the Index Advisor to recommend indexes" in the *Oracle TimesTen In-Memory Database Operations Guide*.

## Scan methods

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

- Full table scan: A *full table scan* examines every row in a table. Because it is typically the least efficient way to evaluate a query predicate, a full scan is only used when no other method is available.
- Rowid lookup: A *rowid* is an address (a unique ID) of a physical tuple in a table 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 index lookups.

However, since TimesTen Scaleout may be configured to have multiple copies, then each copy would have a different ROWID. Since ROWID is the identifier of a specific copy of a row, if that copy is not available, you cannot access the row by ROWID. In this case, you should access the row by primary key.

See "Understanding ROWID in data distribution" in the *Oracle TimesTen In-Memory Database Scaleout User's Guide* for more information.

- Range index scan (on either a permanent or temporary index): A *range index scan* uses a range index to access a table. Such a scan is applicable to exact match predicates such as `t1.a=2` or to range predicates such as `t1.a>2` and `t1.a<10` as long as the column used in the predicate has a range index defined over it. If a range index is defined over multiple columns, it can be used for multiple column predicates. For example, the predicates `t1.b=100` and `t1.c>'ABC'` result in a range index scan if a range index is defined over columns `t1.b` and `t1.c`. The index can be used if it is defined over more columns. For example, if a range index is defined over `t1.b`, `t1.c` and `t1.d`, the optimizer uses the index prefix over columns `b` and `c` and returns all the values for column `d` that match the stated predicate over columns `b` and `c`.
- Hash index lookup (on either a permanent or temporary index): A *hash index lookup* uses a hash index to find rows with an exact match on one or more columns. Such lookups are applicable for equality searches over one or more specified columns.

TimesTen performs fast exact matches through hash indexes and rowid lookups. They perform range matches through range indexes. Optimizer hints can be used to allow or disallow the optimizer from considering certain scan methods when choosing a query plan.

## Join methods

The optimizer can select from multiple join methods. When the rows from two tables are joined, one table is designated the *outer table* and the other the *inner table*. The optimizer decides which of the tables should be the outer table and which should be 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 selects from two join methods when determining the query optimizer plan:

- [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
```

For this example, the optimizer has decided that `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:

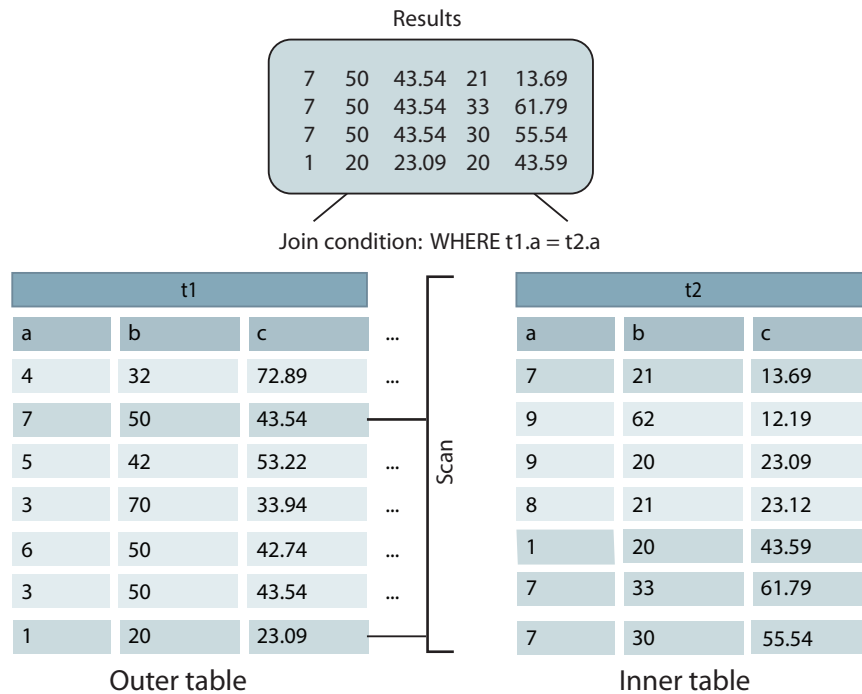
```
7 50 43.54 21 13.69
```

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 range 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, then advance inner cursor
- If Inner.JoinColumn = Outer.JoinColumn, then read match
- If Inner.JoinColumn > Outer.JoinColumn, then 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 range indexes have been created for the tables before preparing the query. If no range indexes exist for the tables being joined before preparing the query, the optimizer may in some situations create temporary range 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

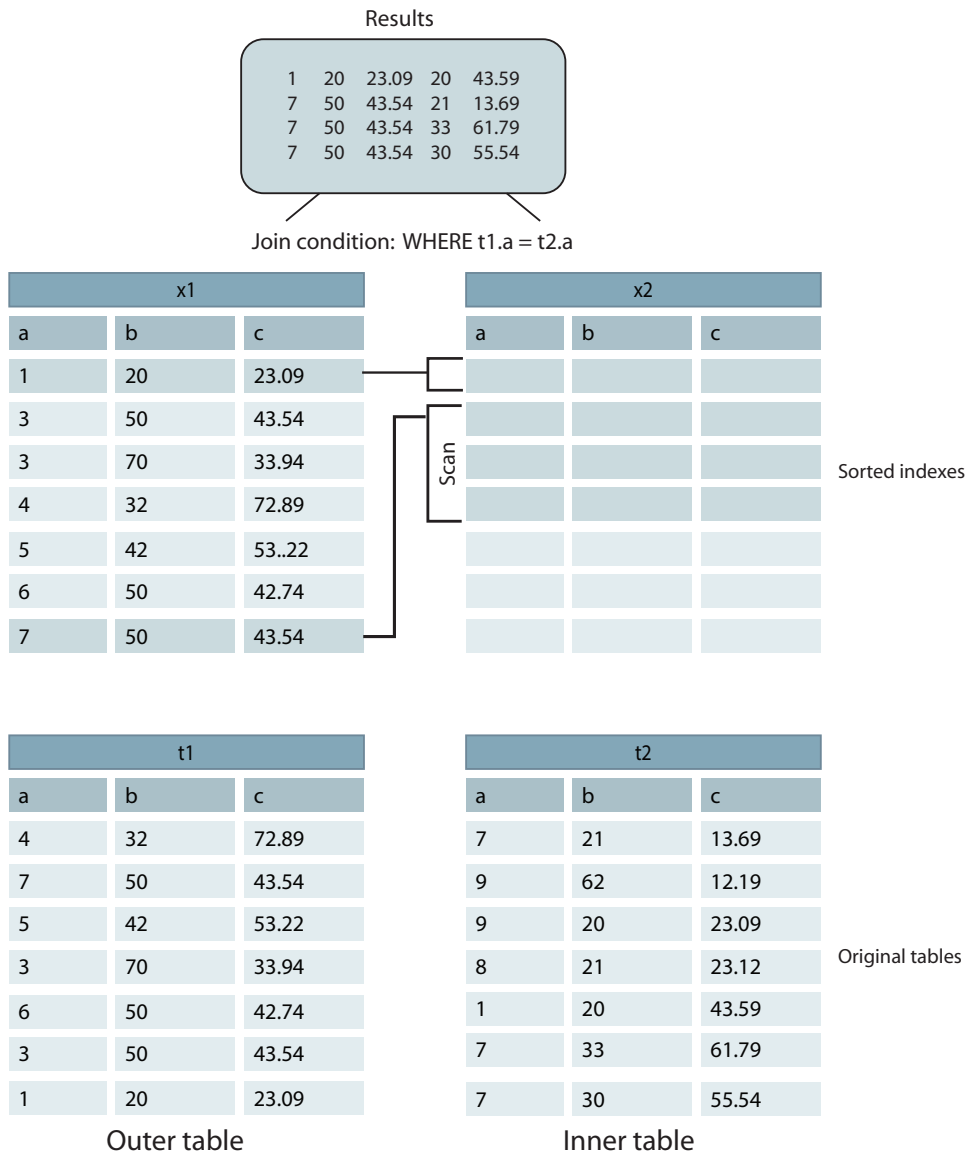
It concatenates a row in x1:

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 the optimizer in most other relational databases, the query optimizer stores the details on how to most efficiently perform SQL operations in a query 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. You can use either the `ttSqlCommandQueryPlan` built-in procedure or the `ttIsql explain` command to display the query plans for recently executed SQL statements. Users can direct the query optimizer to enable or disable the creation of an execution plan in the `SYS.PLAN` table with the `generate plan optimizer` hint. (For transaction level hints, use the `GenPlan` optimizer flag in the `ttOptSetFlag` built-in procedure; for statement level hints, use

the `TT_GenPlan` hint in the SQL statement.) See "The TimesTen Query Optimizer" in the *Oracle TimesTen In-Memory Database Operations Guide* for more information.

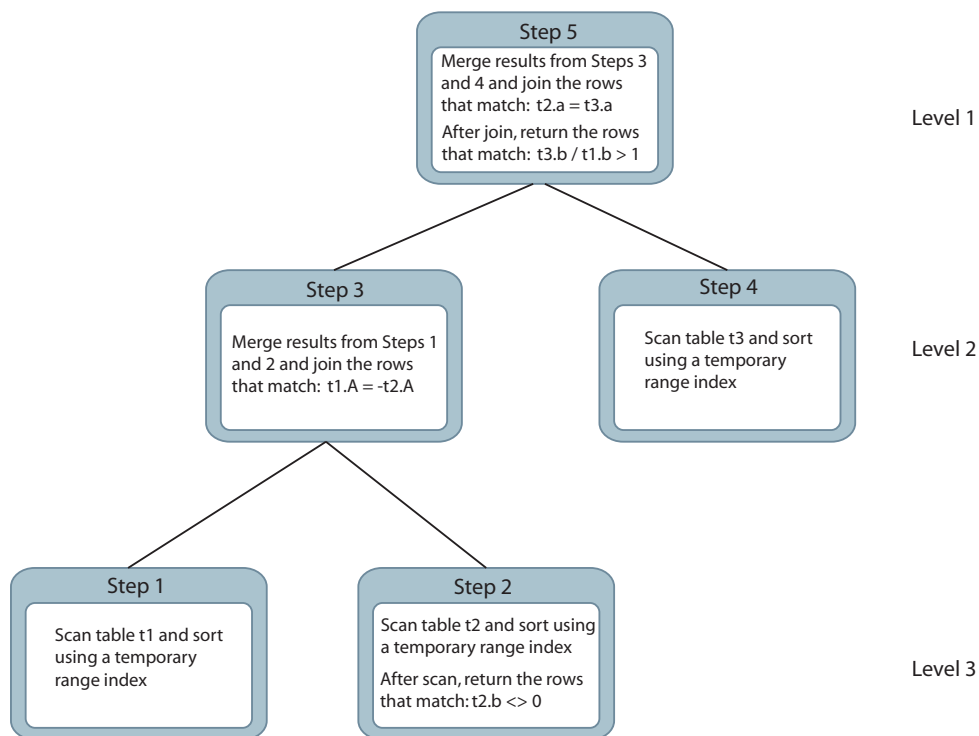
The execution plan designates a separate step for each database operation to be performed to execute the query. The steps in the plan are organized into levels that designate which steps must be completed to generate the results required by the step or steps at the next level.

Consider this 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 breaks down 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**



---

## Data Availability and Integrity

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

- [Transaction logging](#)
- [Checkpointing](#)
- [Data Replication within TimesTen Classic](#)

### Transaction logging

The transaction 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
- When using TimesTen Cache, replicate changes to an Oracle database
- For TimesTen Classic, enable applications to monitor changes to tables through the XLA interface

The transaction log is stored in files on disk. The end of the transaction log resides in an in-memory buffer.

---

**Note:** For more information about logging and checkpointing, see "Transaction Management" in the *Oracle TimesTen In-Memory Database Operations Guide*.

---

### Writing the transaction log buffer to disk

TimesTen writes the contents of the in-memory transaction log buffer to transaction log files on disk at every durable commit, at every checkpoint, and at other times defined by the implementation. Applications that cannot tolerate the loss of any committed transactions if a failure occurs should request a durable commit for every transaction that is not read-only by setting the appropriate durability connection attribute to 1.

Applications that can tolerate the loss of some recently committed transactions can significantly improve their performance by committing some or all of their transactions nondurably. To do so, set the appropriate durability connection attribute to 0 and request explicit durable commits either at regular time intervals or at specific points in their application logic.

For setting the correct durability connection attribute and how to request explicit durable commits for TimesTen Classic, see "Durable options for logging transactions" in the *Oracle TimesTen In-Memory Database Operations Guide*. For setting the correct durability connection attribute and how to request explicit durable commits for TimesTen Scaleout, see "Durability settings" in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

## When are transaction log files deleted?

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

- All transactions writing log records to the transaction log file (or a previous transaction log file) have committed or rolled back.
- All changes recorded in the transaction log file have been written to the checkpoint files.
- When using TimesTen Classic replication, all changes recorded in the transaction log file have been replicated.
- When using TimesTen Cache, all changes recorded in the transaction log file have been propagated to the Oracle database if TimesTen Cache has been configured for that behavior.
- When XLA is used in TimesTen Classic, all changes recorded in transaction log files have been reported to the XLA applications.

When a transaction log file is purgeable, the next checkpoint operation deletes that transaction log file. See ["Checkpointing"](#) on page 6-2 for more details.

## TimesTen commits

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.

TimesTen issues an implicit commit before and after any data definition language (DDL) statement.

## Checkpointing

Checkpoints are used to keep a snapshot of the database. If a system failure occurs, TimesTen can use a checkpoint file and transaction 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 transaction log files that are no longer needed (those that have been declared as purgeable). See ["When are transaction log files deleted?"](#) on page 6-2 for more details.

TimesTen provides two kinds of checkpoints:

- [Nonblocking checkpoints](#)
- [Blocking checkpoints](#)



TimesTen creates nonblocking checkpoints automatically.

For more information about checkpointing, see "Checkpoint operations" in the *Oracle TimesTen In-Memory Database Operations Guide*.

## 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. See "Setting and managing checkpoints" in the *Oracle TimesTen In-Memory Database Operations Guide* for more details.

## Blocking checkpoints

When using TimesTen Classic, an application can call the `ttCkptBlocking` built-in procedure to initiate a blocking checkpoint in order to construct a transaction-consistent checkpoint. Blocking checkpoints acquire an exclusive database lock. While a blocking checkpoint operation is in progress, any other new transactions are put in a queue behind the checkpointing transaction. Thus, a long running transaction may cause other transactions to be held up. A long-running transaction could prevent a blocking checkpoint from acquiring the exclusive database lock, which could cause the checkpoint and all subsequent transactions to wait until the long-running transaction commits or rolls back (or the checkpoint request is cancelled). No transaction log is needed to recover from a blocking checkpoint since the checkpoint record contains the information needed to recover.

See "Setting and managing checkpoints" in the *Oracle TimesTen In-Memory Database Operations Guide* for more details.

## Recovery from the transaction log and checkpoint files

During recovery, the latest checkpoint file is read into memory. All transactions that have been committed since the last checkpoint and whose log records are on disk are rolled forward from the appropriate transaction log files. Note that the transactions on disk include all transactions that were committed durably as well as all transactions whose log records aged out of the in-memory transaction log buffer. Uncommitted or rolled-back transactions are not recovered.

- When using TimesTen Scaleout, there is a process to facilitate automatic recovery of the data from the checkpoint and transaction log files. See "Recovering from Failure" in the *Oracle TimesTen In-Memory Database Scaleout User's Guide* for more information.
- When using TimesTen Classic, 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 region for the database and recovers its data from the checkpoint and transaction log files.

For applications that require uninterrupted access to TimesTen Classic data in the event of failures, see ["Data Replication within TimesTen Classic"](#) on page 6-4.

## Data Replication within TimesTen Classic

The fundamental motivation behind data replication in TimesTen Classic is to make data highly available to applications with minimal performance impact. In addition to its role in failure recovery, replication is also useful for distributing application workloads across multiple databases for maximum performance and for facilitating online upgrades and maintenance.

Replication is the process of copying data from a *master* database to a *subscriber* database. 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 the transaction log for the master database. 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 transaction log until they can be applied at the subscriber.

You can increase replication throughput by configuring parallel replication at database creation time. You configure the number of threads for applying updates to subscribers. The updates are transmitted in commit order. See "Configuring automatic parallel replication" in the Oracle TimesTen In-Memory Database Replication Guide for more details.

TimesTen recommends the active standby pair configuration for highest availability. It is the only replication configuration that you can use for replicating TimesTen Cache.

---

---

**Note:** For more information about replication, see *Oracle TimesTen In-Memory Database Replication Guide*.

---

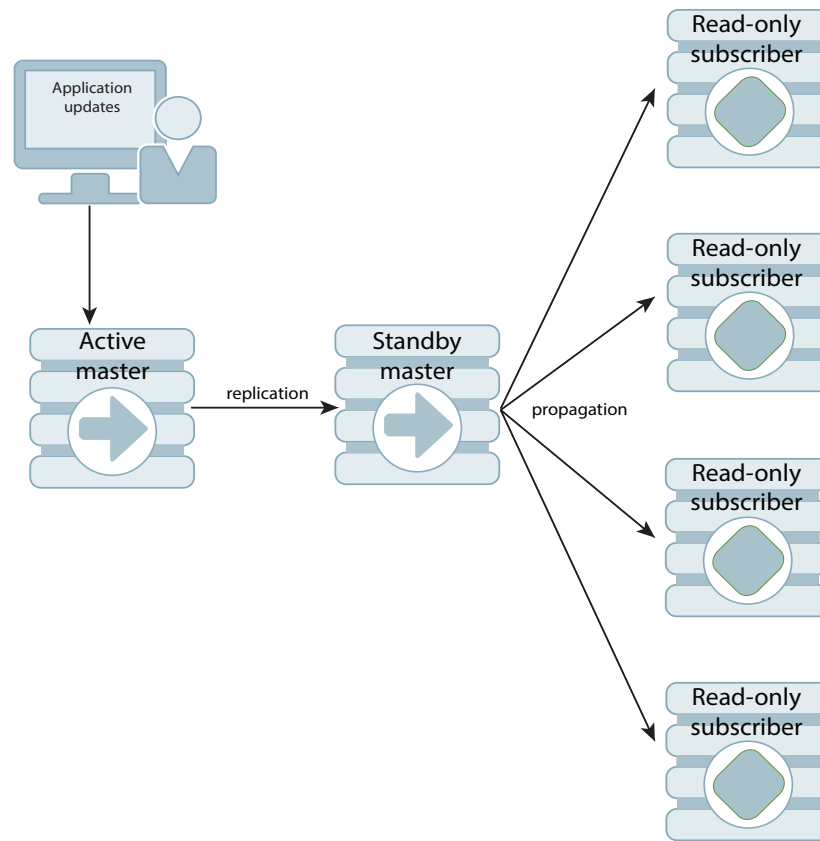
---

The rest of this section includes the following topics:

- [Active standby pair](#)
- [Classic replication configurations](#)
- [Asynchronous and return service replication](#)
- [Replication failover and recovery](#)
- [Automatic client failover for active standby pair replication](#)

### Active standby pair

An active standby pair includes an active database, a standby database, optional read-only subscriber databases, and the tables and cache groups that comprise the databases. [Figure 6–1](#) shows an active standby pair.

**Figure 6–1 Active standby pair**

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

Only one of the master databases can function as an active database at a specific time. If the active database fails, the role of the standby database must be changed to active before recovering the failed database as a standby database. The replication agent must be started on the new standby database.

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

Active standby replication can be used with TimesTen Cache to achieve cross-tier high availability. Active standby replication is available for both read-only and asynchronous writethrough cache groups. When used with read-only cache groups, updates are sent from the Oracle database to the active database. Thus, the Oracle database plays the role of the application in this configuration. When used with asynchronous writethrough cache groups, the standby database propagates updates that it receives from the active database to the Oracle database. In this scenario, the Oracle database plays the role of one of the read-only subscribers.

An active standby pair that replicates one of these types of cache groups can perform failover and recovery automatically with minimal chance of data loss. See "Replicating cache groups within active standby pairs" in *Oracle TimesTen In-Memory Database Replication Guide*.

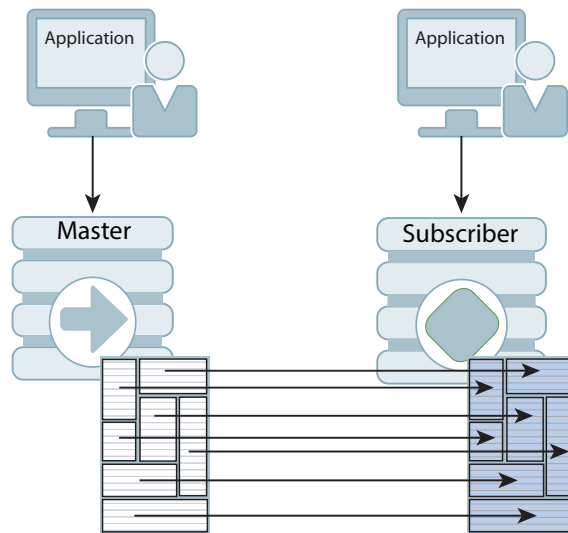
## Classic replication configurations

TimesTen replication architecture is flexible enough to achieve balance between performance and availability. In general, classic replication can be configured to be unidirectional from a master to one or more subscribers, or bidirectional between two or more databases that serve as both master and subscriber.

### Unidirectional replication

[Figure 6–2](#) shows a *unidirectional* replication scheme. The application is configured on both hosts so that the subscriber is ready to take over if the master host fails. While the master is up, updates from the application to the master database are replicated to the subscriber database. The application on the subscriber host does not execute any updates against the subscriber database, but may read from that database. If the master fails, the application on the subscriber host takes over the update function and starts updating the subscriber database.

**Figure 6–2 Unidirectional replication scheme**



Replication can also be used to copy updates from a master database to many subscriber databases. [Figure 6–3](#) shows a replication scheme with multiple subscribers.

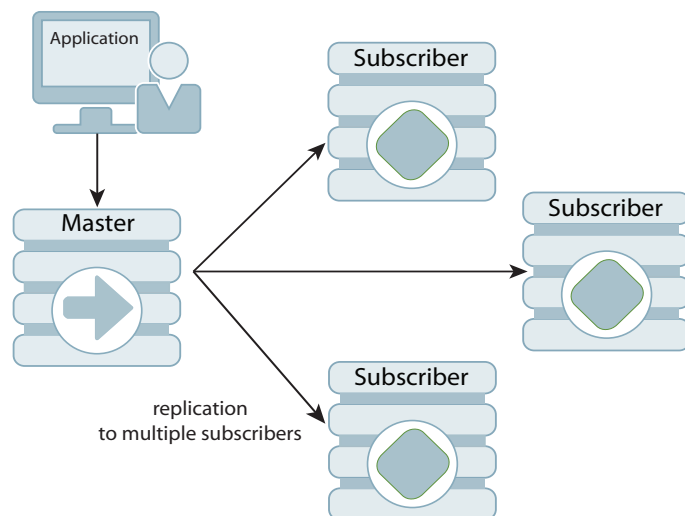
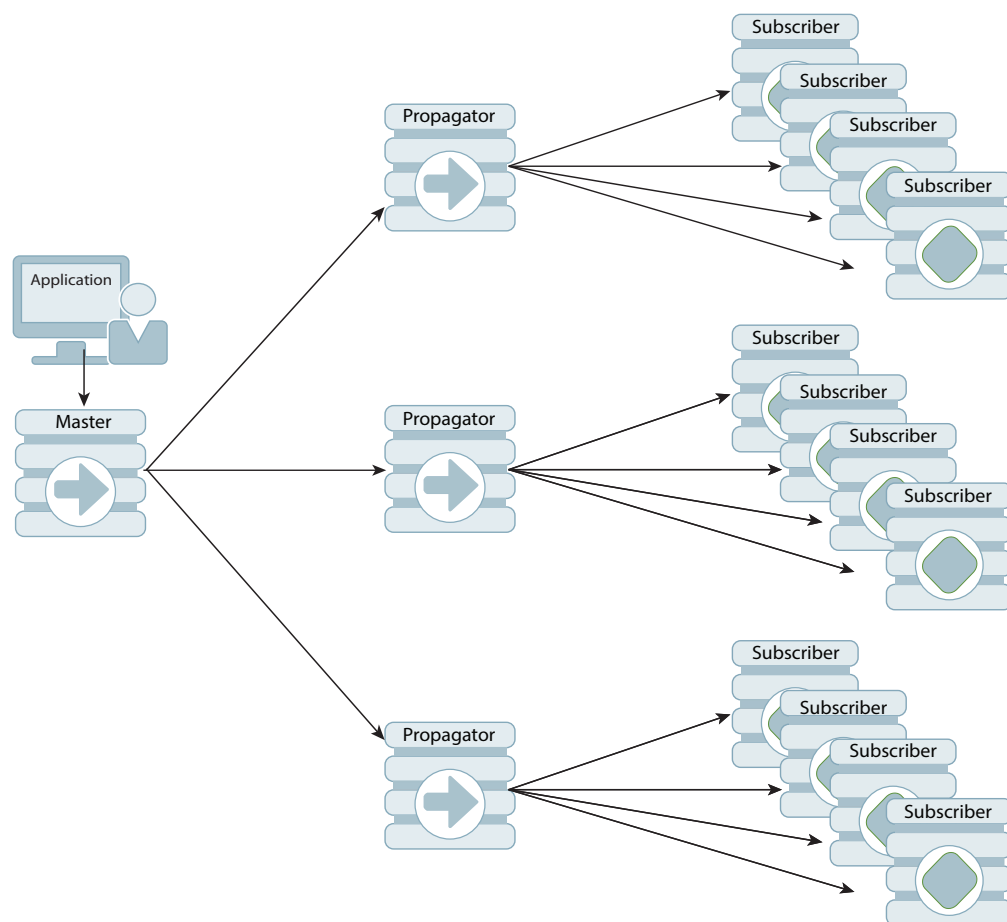
**Figure 6–3 Unidirectional replication to multiple subscribers**

Figure 6–4 shows a *propagation* configuration. One master copies updates to three subscriber nodes, which act as propagator nodes to forward those same updates to additional subscribers.

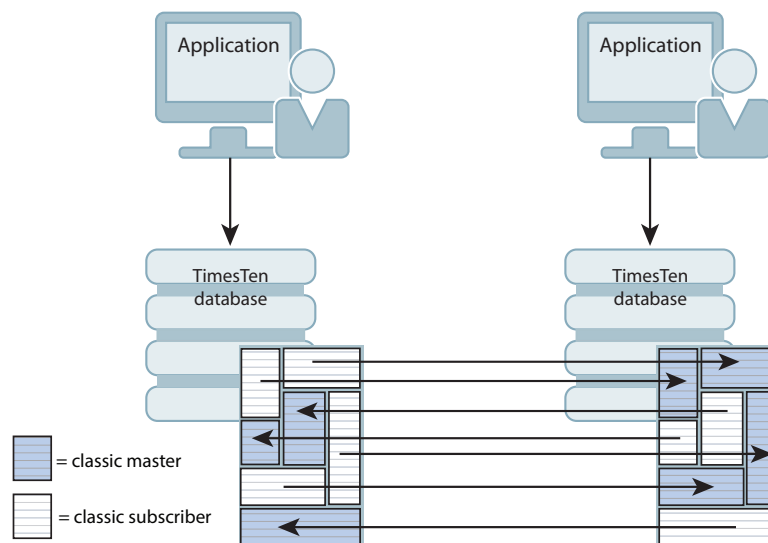
**Figure 6–4 Propagation configuration**

## Bidirectional replication

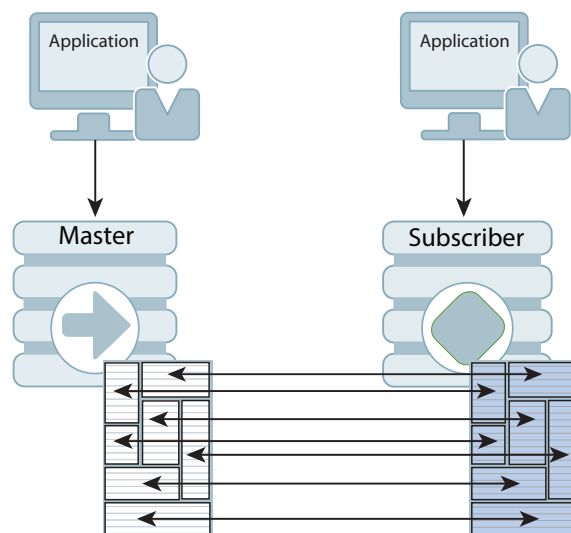
Bidirectional replication schemes are used for load balancing. The workload can be split between two bidirectionally replicated databases. There are two basic types of load-balancing configurations:

- *Split workload* where each database bidirectionally replicates a portion of its data to the other database. [Figure 6–5](#) shows a split workload configuration.
- *Distributed workload* where user access is distributed across duplicate application/database combinations that replicate updates to each other. 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. [Figure 6–6](#) shows a distributed workload configuration.

**Figure 6–5 Split workload replication**



**Figure 6–6 Distributed workload replication**



## Asynchronous and return service replication

TimesTen replication is asynchronous by default. 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* 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.

---

**Note:** Do not use the return twosafe service with bidirectional replication. This can cause deadlocks.

---

Applications that use the return services trade some performance to ensure higher levels of consistency and reduce the risk of transaction loss 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 potential loss of transactions.

## Replication failover and recovery

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

You can use Oracle Clusterware with TimesTen Classic to manage failures automatically in systems with active standby pairs. Other kinds of replication schemes can be managed with custom and third-party cluster managers. They detect failures, redirect users or applications from the failed database to either a standby database or a subscriber, and manage recovery of the failed database. The cluster manager or administrator can use TimesTen-provided utilities and functions 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 standby database or a subscriber in order to continue service with no or minimal interruption.

## **Automatic client failover for active standby pair replication**

You can configure automatic client failover for TimesTen Classic databases that have active standby pairs with client/server connections. This enables the client to fail over automatically to the server on which the standby database resides.

For more information about automatic client failover, see "Using automatic client failover" in the *Oracle TimesTen In-Memory Database Operations Guide*.



---

## TimesTen Cache

TimesTen Cache is supported in TimesTen Classic and provides the ability to transfer data between an Oracle database and a TimesTen Cache database.

You can cache data from an Oracle database in a TimesTen Cache database by creating cache groups in TimesTen where each cache group maps to a single table in the Oracle database or to a group of tables related by foreign key constraints.

This chapter includes the following topics:

- [Cache groups](#)
- [Dynamic cache groups and explicitly loaded cache groups](#)
- [Transmitting data between the TimesTen Cache and an Oracle database](#)
- [Aging feature](#)
- [Passthrough feature](#)
- [Replicating cache groups](#)
- [Disaster recovery through TimesTen Cache](#)

### Cache groups

You can cache data in an Oracle database by creating a cache group in a TimesTen Cache database. A cache group can be created to cache a single Oracle Database table or a set of related Oracle Database tables. The cached data from the Oracle database can consist of all the rows and columns or a subset of the rows and columns in the Oracle database tables.

TimesTen Cache supports the following features:

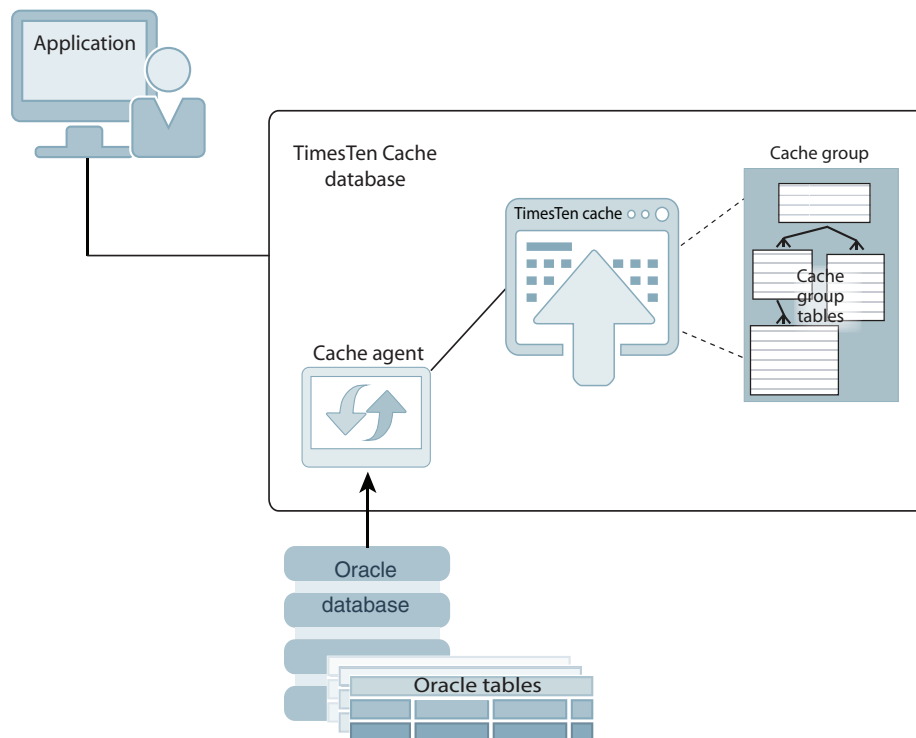
- Applications can both read from and write to cache groups.
- Cache groups can be refreshed (upload the Oracle database data into the cache group) automatically or manually.
- Cache updates can be sent to the Oracle database automatically or manually. The updates can be sent synchronously or asynchronously.

The TimesTen Cache database interacts with the Oracle database to perform all of the synchronous cache group operations, such as creating a cache group and propagating updates between the cache group and the Oracle database. A process called the cache agent performs asynchronous cache operations, such as loading data into the cache group, manually refreshing the data from the Oracle database to the cache group, and automatically refreshing the data from the Oracle database to the cache group.

For full information about TimesTen Cache, see *Oracle TimesTen Application-Tier Database Cache User's Guide*.

Figure 7–1 illustrates the TimesTen Cache features and processes.

**Figure 7–1 TimesTen Cache**



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 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 Cache record belongs to only one cache instance and has only one parent in its cache group.

The most commonly used cache group types are:

- *Read-only cache group* - A read-only cache group enforces a caching behavior in which committed updates to the Oracle database tables are automatically refreshed to the corresponding cache tables in the TimesTen Cache database.
- *Asynchronous writethrough (AWT) cache group* - An AWT cache group enforces a caching behavior in which committed updates to cache tables in the TimesTen Cache database are automatically propagated to the corresponding the Oracle database tables asynchronously.

Other types of cache groups are:

- *Synchronous writethrough (SWT) cache group* - An SWT cache group enforces a caching behavior in which committed updates to cache tables in the TimesTen Cache database are automatically propagated to the corresponding Oracle database tables synchronously.

- *User managed cache group* - A user managed cache group defines customized caching behavior. For example, individual cache tables in a user managed cache are not constrained to be all of the same type. Some tables may be defined as read-only while others may be defined as updatable.

If you have multiple TimesTen Cache databases, the data in cached tables is not shared among separate TimesTen Cache databases. Consequently, the content of the databases may overlap with no coordination from the TimesTen Cache unless your applications logically partition their data between different nodes or use read-only cache groups.

For more information about cache group types, see "Cache group types" in the *Oracle TimesTen Application-Tier Database Cache User's Guide*.

## Dynamic cache groups and explicitly loaded cache groups

Cache groups can be either dynamically loaded or explicitly loaded.

In explicitly loaded cache groups, the application preloads data into the cache tables from the Oracle database using a load cache group operation. From that point on, all data needed by the application is available in the TimesTen Cache database.

In dynamic cache groups, cache instances are automatically loaded into the TimesTen Cache from the Oracle database when the application references cache instances that are not already in the TimesTen Cache. The use of dynamic cache groups is typically coupled with least recently used (LRU) aging so that less recently used cache instances are aged out of the cache to free up space for recently used cache instances. Using dynamic cache groups is appropriate when the size of the data that qualifies for caching exceeds the size of the memory available for the TimesTen Cache database.

All cache group types (read-only, AWT, SWT, user managed) can be defined as a explicitly loaded or dynamic.

For more information about explicitly or dynamically loaded cache groups, see "Loading data into a cache group: Explicitly loaded and dynamic cache groups" in the *Oracle TimesTen Application-Tier Database Cache User's Guide*.

## Transmitting data between the TimesTen Cache and an Oracle database

The TimesTen Cache maintains consistency between cached data and the Oracle database by automatically propagating updates from cache groups to the Oracle database and automatically refreshing data in cache groups from the Oracle database.

The rest of this section includes the following topics:

- [Updating a cache group from the Oracle database tables](#)
- [Updating the Oracle database tables from a cache group](#)

### Updating a cache group from the Oracle database tables

The following mechanisms are available to keep a cache group synchronized with the corresponding data in the Oracle database tables:

- *Autorefresh* - An *incremental autorefresh* operation updates only records that have been modified in the Oracle database since the last refresh. The TimesTen Cache automatically performs the incremental refresh at specified time intervals. You can also specify a *full autorefresh* operation, which automatically refreshes the entire cache group at specified time intervals.

- *Manual refresh* - An application issues a `REFRESH CACHE GROUP` statement to refresh either an entire cache group or a specific cache instance. It is equivalent to unloading and then loading the cache group or cache instance.

These mechanisms are useful under different circumstances. A full autorefresh may be the best choice if the Oracle database table is updated only once a day and many rows are changed. An incremental autorefresh is the best choice if the Oracle database 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.

For more information about autorefresh or manual refresh operations for cache groups, see "Cache Group Operations" in the *Oracle TimesTen Application-Tier Database Cache User's Guide*.

## Updating the Oracle database tables from a cache group

The *propagate* and *flush* mechanisms are available to keep the Oracle database up to date with the cache group:

- *Propagate* - The most common way to propagate cache group data to the Oracle database is by using an asynchronous writethrough (AWT) cache group. Other methods of updating the Oracle database tables are using a synchronous writethrough (SWT) cache group or specifying the `PROPAGATE` option in a user managed cache group.

Changes to an AWT cache group are committed without waiting for the changes to be applied to the Oracle database tables. AWT cache groups provide better response times and performance than SWT cache groups and user managed cache groups with the `PROPAGATE` option, but the TimesTen Cache database and the Oracle database do not always contain the same data because changes are applied to the Oracle database tables asynchronously.

You can increase throughput from AWT cache groups to the Oracle database tables by configuring parallel replication at database creation time. You configure the number of threads for applying updates to the Oracle database tables.

- *Flush* - A flush operation can be used to propagate updates manually from a user managed cache group to the Oracle database. An application initiates a flush operation by issuing a `FLUSH CACHE GROUP` statement. Flush operations are useful when frequent updates occur for a limited period of time over a set of records. Flush operations do not propagate deletes.

For more information about autorefresh or manual refresh operations for cache groups, see "Cache Group Operations" in the *Oracle TimesTen Application-Tier Database Cache User's Guide*.

## Aging feature

Records can be automatically aged out of a TimesTen database, and cache instances can be automatically aged out of a TimesTen Cache database. Aging can be usage-based or time-based. 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.

Dynamic load can be used to reload a requested cache instance that has been deleted by aging.

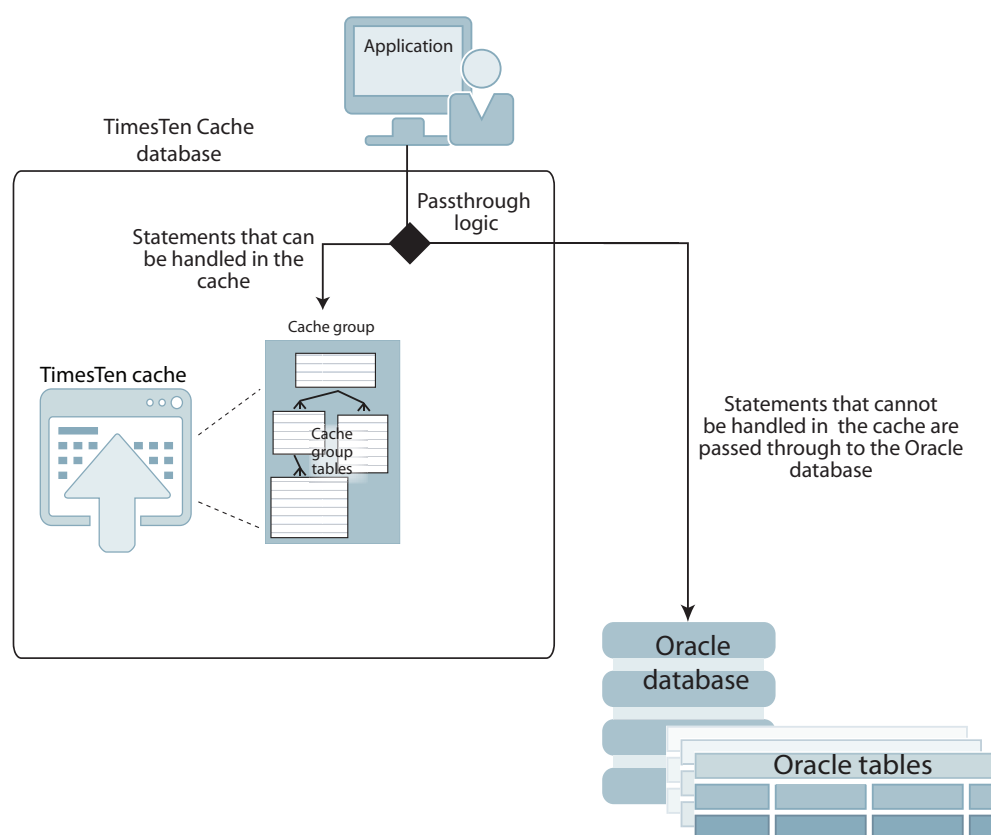
For more information about aging in cache groups, see "Implementing aging in a cache group" in *Oracle TimesTen Application-Tier Database Cache User's Guide*. For information

about aging in tables that are not in cache groups, see "Implementing aging in your tables" in *Oracle TimesTen In-Memory Database Operations Guide*.

## Passthrough feature

Applications use a single connection to a TimesTen Cache to send SQL statements to either a cache group or to the Oracle database. This single-connection capability is enabled by a *passthrough* feature that checks whether the SQL statement can be handled locally by the cached tables in the TimesTen Cache or if it must be redirected to the Oracle database, as shown in Figure 7–2. The passthrough feature provides settings that specify what types of statements are to be passed through and under what circumstances. The specific behavior of the passthrough feature is controlled by the `PassThrough` TimesTen Cache general connection attribute.

**Figure 7–2 Passthrough feature**



For more information about the passthrough feature, see "Setting a passthrough level" in *Oracle TimesTen Application-Tier Database Cache User's Guide*.

## Replicating cache groups

You can use an active standby pair to replicate AWT cache groups and read-only cache groups. For more information about replicating cache groups, see "Cache groups and replication" in the *Oracle TimesTen In-Memory Database Replication Guide*.

## Disaster recovery through TimesTen Cache

TimesTen Cache supports Oracle Data Guard, as well as providing its own HA capability through active standby pair replication of cache tables for read-only and AWT cache groups.

You can use the following methods to set up a disaster recovery site:

- Replicating an AWT cache group with a subscriber propagating to an Oracle database.

You can recover from a complete site failure by creating a special disaster recovery read-only subscriber as part of the active standby pair replication scheme. The standby database sends updates to cache group tables on the read-only subscriber. This special subscriber is located at a remote disaster recovery site and can propagate updates to a second Oracle database, also located at the disaster recovery site. The disaster recovery subscriber can take over as the active in a new active standby pair at the disaster recovery site if the primary site suffers a complete failure. Any applications may then connect to the disaster recovery site and continue operating, with minimal interruption of service.

See "Using a disaster recovery subscriber in an active standby pair" in the *Oracle TimesTen Application-Tier Database Cache User's Guide* for details.

- Using TimesTen Cache with either synchronous or asynchronous Data Guard.

Oracle Data Guard provides the management, monitoring, and automation software infrastructure to create and maintain one or more synchronized standby Oracle databases to protect data from failures, disasters, errors, and corruptions. If the primary Oracle database becomes unavailable because of a planned or an unplanned outage, Data Guard can switch any standby Oracle database to the primary role, thus minimizing downtime and preventing any data loss.

- Asynchronous Data Guard support: You can cache tables from an Oracle Active Data Guard with the asynchronous redo transport mode into read-only cache groups. The read-only cache groups are replicated within an active standby pair replication scheme. The Active Data Guard configuration includes a primary Oracle database that communicates over an asynchronous transport to a single physical standby Oracle database.
- Synchronous Data Guard support: TimesTen Cache works with synchronous physical standby failover and switchover and logical standby switchover. During a transient upgrade, a physical standby Oracle database is transformed into a logical standby Oracle database.

See "Using TimesTen Cache with Data Guard" in the *Oracle TimesTen Application-Tier Database Cache User's Guide* for details.

---

## Event Notification

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

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

This chapter includes the following topics:

- [Detect transaction modifications with the Transaction Log API](#)
- [Monitor tables with materialized views and XLA](#)
- [Monitor problems with SNMP traps](#)

### Detect transaction modifications with the Transaction Log API

TimesTen Classic provides a Transaction Log API (XLA) that enables applications to monitor the transaction log of a 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 Classic 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 monitor the database for updates on certain stock prices and use that information to determine whether to execute orders. See "[TimesTen Classic application scenario](#)" on page 2-3 for a complete example.

XLA can also be used to build a custom data replication solution in place of the TimesTen replication. Such XLA-enabled replication solutions might involve replicating data to a non-TimesTen database.

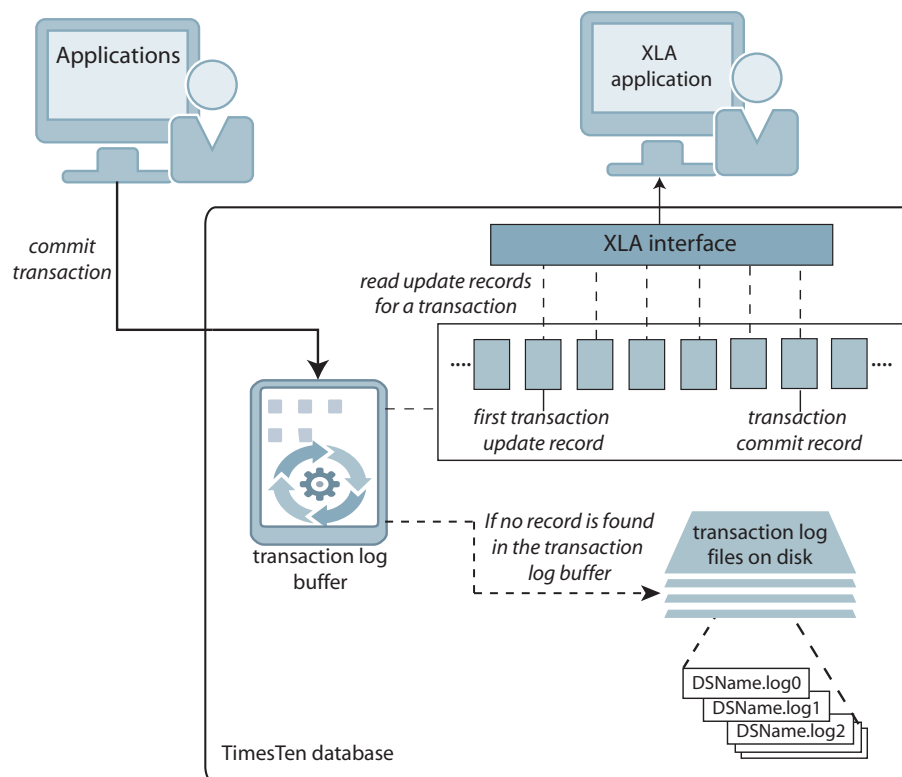
For more information about XLA, see "XLA and TimesTen Event Management" in the *Oracle TimesTen In-Memory Database C Developer's Guide* and "Using JMS/XLA for Event Management" in *Oracle TimesTen In-Memory Database Java Developer's Guide*.

For more information about TTClasses, see "Using TTClasses XLA" in the *Oracle TimesTen In-Memory Database TTClasses Guide*.

## How XLA works

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 transaction log files on disk, as shown in [Figure 8-1](#). 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. As described in ["When are transaction log files deleted?"](#) on page 6-2, the presence of an XLA bookmark prevents transaction log files from being deleted until the XLA log records in that transaction log file have been read and acknowledged by the XLA application.

**Figure 8-1** How XLA works



## Log update records

Update records are available to be read from the log as soon as the transaction that created them commits. A log monitoring 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



## Monitor tables with 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 materialized views offer this same capability in TimesTen, another purpose of materialized views in TimesTen Classic is their role in working with XLA to keep track of specific rows and columns in multiple tables.

---

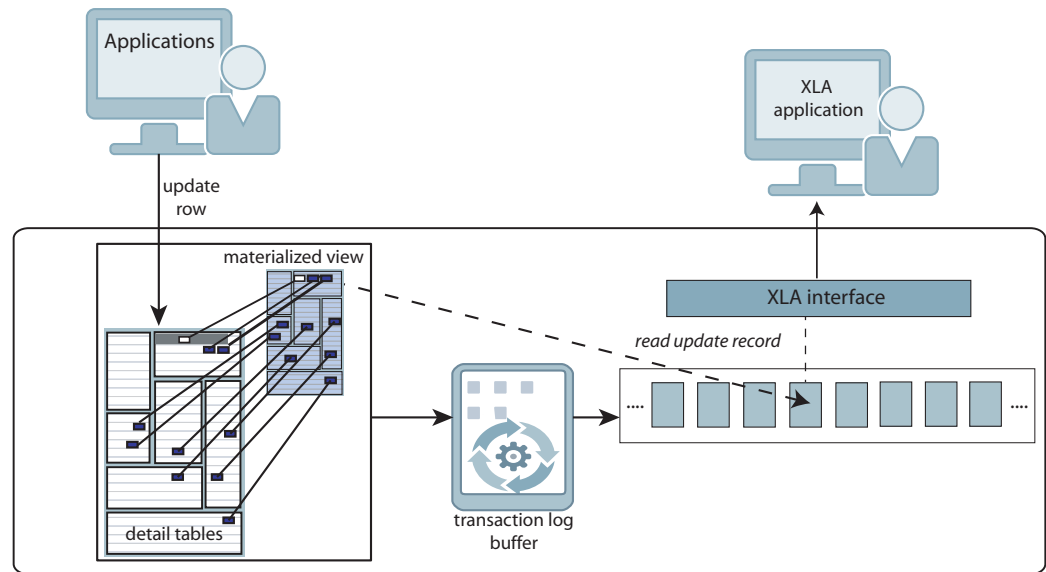
**Note:** For more information about materialized views, see "Understanding materialized views" in *Oracle TimesTen In-Memory Database Operations Guide*. Also see the `CREATE MATERIALIZED VIEW` statement in *Oracle TimesTen In-Memory Database SQL Reference*.

---

When a materialized view is present, an XLA application needs to monitor only 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 the detail tables of that materialized view, including records reflecting updates to rows and columns of no interest to the application.

Figure 8–2 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 columns and rows in the same detail table that are not part of the materialized view are not seen by the XLA application.

**Figure 8–2 Using XLA to detect updates on a materialized view table**



See "TimesTen Classic application scenario" on page 2-3 for an example of a trading application that uses XLA and a materialized view to detect updates to specific stocks.

The TimesTen implementation of materialized views emphasizes performance as well as the ability to detect updates across multiple tables. Readers familiar with other implementations of materialized views should note that the following tradeoffs have been made:

- The application must explicitly create materialized views. The TimesTen query optimizer has no facility to create materialized views automatically.

- The query optimizer does not rewrite queries on the detail tables to reference materialized views. Application queries must directly reference views.
- There are some restrictions to the SQL used to create materialized views.

## Monitor problems with 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 Cache 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 enables properly configured network management software to take immediate action.

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

---

## TimesTen Administration

This chapter includes the following topics:

- [Installing TimesTen](#)
- [Access control](#)
- [Command line administration](#)
- [SQL administration](#)
- [SQL Developer](#)
- [ODBC Administrator](#)
- [Migrating a database](#)
- [Upgrading TimesTen](#)

### Installing TimesTen

For TimesTen Scaleout, TimesTen is installed on Linux systems by unzipping the TimesTen distribution on the host of the first management instance. Then, use the `ttGridAdmin` utility to copy the distribution to other hosts involved in the grid. See Chapter 2, "Prerequisites and Installation of TimesTen Scaleout" in the *Oracle TimesTen In-Memory Database Scaleout User's Guide* for details.

For TimesTen Classic, TimesTen is installed by unzipping a distribution file. See *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide* for details on installing TimesTen Classic.

### Access control

TimesTen is installed with access control to allow only users with specific privileges to access particular TimesTen features. TimesTen Access Control uses standard SQL operations to establish user accounts with specific privileges. TimesTen offers object-level access control as well as database-level access control.

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

### Command line administration

Most TimesTen administration tasks are performed with command line utilities. The following table summarizes common utilities:

Name	Description
ttAdmin	A general utility for managing a database in TimesTen Classic. Used to specify policies for automatically or manually loading and unloading a database from RAM, as well as to starting and stopping cache agents and replication agents.
ttBackup and ttRestore	Used to create a backup copy of a database in TimesTen Classic and restore it at a later time.
ttBulkCp	Used to transfer data between TimesTen tables and ASCII files.
ttGridAdmin	A general utility for all aspects of administering TimesTen Scaleout, such as creating a grid, adding or removing data instances or management instances, creating a database, and redistributing data to new data instances.
ttIsql	Used to run SQL interactively from the command line, similar to Oracle SQL*Plus. Also provides a number of administrative commands to reconfigure and monitor a database.
ttMigrate	Used to migrate the contents of a database between major TimesTen releases.
ttRepAdmin	Used to monitor replication status for replication schemes in TimesTen Classic.
ttSize	Used to estimate the amount of space to allocate for a table in the database.
ttStats	Used to monitor database metrics (statistics, states, and other information). Automatically captures system snapshots, and takes and compares snapshots of metrics.
ttStatus	Used to display information that describes the current state of a TimesTen database.
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, as well as latches held by each connection. The ttXactAdmin utility also enables users of TimesTen Classic to commit, abort or forget an XA transaction branch.

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

For more information about administering TimesTen replication, see the *Oracle TimesTen In-Memory Database Replication Guide*.

For information about TimesTen utilities (both general-purpose and TimesTen Scaleout-specific), see the *Oracle TimesTen In-Memory Database Reference*.

## SQL administration

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

Administrators can execute SQL statements within the ttIsql utility. For example, there are several built-in ttIsql commands that display information on database structures.

---

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

---

TimesTen stores metadata (information about the contents of your database) in system tables in your database. TimesTen also provides system views that enable you to use SQL to query information.

In addition, TimesTen provides several local (v\$) and global (gv\$) system views you can query to retrieve metadata information about your database.

- v\$ views: Views that return information for a TimesTen Classic database or for the element to which your application is connected in TimesTen Scaleout.
- gv\$ view: gv\$ views return the same type of information as the v\$ views, except that the information returned is global in nature as it contains the contents of the v\$ view for every element of the TimesTen Scaleout database.

See "System Tables and Views" in the *Oracle TimesTen In-Memory Database System Tables and Views Reference* for more information.

## SQL Developer

Oracle SQL Developer is a graphical tool for database development tasks. SQL Developer is a Java application that supports direct-linked and client/server connections to a TimesTen database. Support for connecting to multiple databases enables SQL Developer users to work with data in the TimesTen and the Oracle databases concurrently.

- [Facilitating TimesTen Scaleout development and management](#)
- [Facilitating TimesTen Classic development and management](#)

For more information about SQL Developer, see *Oracle SQL Developer Oracle TimesTen In-Memory Database Support User's Guide*.

### Facilitating TimesTen Scaleout development and management

Use SQL Developer to facilitate TimesTen Scaleout development and management.

- Create, manage, and explore a grid and its components.
- Browse, create, edit and drop particular database objects
- Run SQL statements and scripts
- Manipulate and export data
- View and create reports
- View database metrics

For more information, see "Using SQL Developer to work with TimesTen Scaleout" in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

### Facilitating TimesTen Classic development and management

Use SQL Developer to facilitate TimesTen Classic development and management:

- Browse, create, and manage database objects and PL/SQL programs
- Automate cache group operations

- Manipulate and export data
- Execute SQL and PL/SQL statements and scripts
- View and create reports
- View database metrics

## ODBC Administrator

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

## Migrating a database

In TimesTen Scaleout, you can migrate a database from TimesTen Classic to TimesTen Scaleout. See "Migrating, Backing Up and Restoring Data" in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

In TimesTen Classic, you can migrate data between major releases (for example, from TimesTen 11.2.2 to 18.1) by using the `ttMigrate` utility to export the data from the old release and import it to the new release. The `ttMigrate` utility saves database objects in a binary file and upgrades or downgrades database objects by restoring the objects from the binary file into the target database. For more information, see "Migrating a database" in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.

## Upgrading TimesTen

The following sections describe the process for upgrading to a new release of TimesTen.

- [Upgrades for a grid in TimesTen Scaleout](#)
- [Offline upgrades for TimesTen Classic](#)
- [Online upgrades for TimesTen Classic](#)

### Upgrades for a grid in TimesTen Scaleout

Upgrading a grid consists of ensuring that every instance uses for its operations the installation files provided by a newer patch release of TimesTen. For more information about upgrading TimesTen Scaleout, see "Upgrading a grid" in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

### Offline upgrades for TimesTen Classic

In TimesTen Classic, offline upgrades are performed either by using the `ttInstanceModify` or the `ttMigrate` utilities.

Offline upgrades can be used to:

- Apply a patch set or a patch level within a patch set, run the `ttInstanceModify` utility to upgrade the instance and its databases.
- Move to a different major release, you must run `ttMigrate` to export the database to a flat file and then use `ttMigrate` again to import the data into the new database.

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.

For more information, see "Upgrades in TimesTen Classic" in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.

## Online upgrades for TimesTen Classic

In TimesTen Classic, replication enables online upgrades, 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.

Upgrading a database to a new release is performed by disconnecting all applications from one of two replicated copies of a database, making a backup of the database with the ttMigrate utility from the older release, loading the backup into a newer release database using the ttMigrate utility from the newer release, and then reconnecting all applications to the database in the upgraded instance.

For more information, see "Upgrades in TimesTen Classic" in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.





## A

---

- Access Control
  - database level, 9-1
  - object level, 9-1
- active standby pair, see replication, active standby pair
- administration
  - command line utilities, 9-1
- aging
  - cache groups, 7-3, 7-4
  - data, 1-14
- architecture
  - TimesTen, 3-1
  - TimesTen cache, 3-1
- autorefresh, 7-3
- AWT cache group, 7-2
  - parallel propagation, 7-4

## C

---

- C++ interface, 1-5, 8-1
- cache group
  - aging, 7-3
  - asynchronous writethrough, 7-2
  - definition, 3-7
  - description, 7-1
  - dynamic, 7-3
  - explicitly loaded, 7-3
  - passthrough feature, 7-5
  - read-only, 7-2
  - replicating, 7-5
  - synchronous writethrough, 7-2
  - user managed, 7-3
- cache instance, 7-2
- caching Oracle data in TimesTen
  - overview, 3-7
- character sets, 1-7
- checkpoints
  - blocking, 6-3
  - fuzzy, 6-3
  - nonblocking, 6-3
  - operations, 1-5
  - purpose, 3-7
  - recovery, 6-3
- classic replication, see replication, classic

- client
  - automatic failover, 6-10
- client/server connection, 1-7, 3-6
- cluster managers, 6-9
- commit behavior, 6-2
- concurrency, 1-6
- connection
  - client/server, 3-6
  - direct driver, 3-5
  - driver manager, 3-6

## D

---

- daemon
  - attributes, see timesten.conf file
  - control operations, 3-4
  - overview, 3-4
- data
  - multiple copies, 1-12
  - transparency, 1-11
- Data Guard, 7-6
- data structures, 3-4
- database
  - metrics, 1-8
  - state, 1-8
  - statistics, 1-8
- deadlock detection
  - description, 4-3
- direct driver connection, 1-7, 3-5
- disaster recovery, 7-5
- driver manager connection, 3-6
- durability, 1-5
- durable commits, 6-1
- dynamic cache group, 7-3
  - definition, 7-3

## E

---

- explicitly loaded cache group
  - definition, 7-3

## F

---

- failover
  - configuring for client, 6-10
- fault tolerance, 1-12

files  
    timesten.conf, 3-4  
flush  
    from TimesTen Cache to Oracle database, 7-4

## G

---

globalization support, 1-7  
grid  
    monitor, 1-8

## H

---

hash index  
    definition, 5-3  
    overview, 5-3  
    when used, 5-3  
high availability, 1-12  
    replication, 3-7, 6-4

## I

---

index  
    advice, 5-3  
    and query optimizer, 5-3  
    hash, 5-3  
    range, 5-3  
    supported types, 5-3  
Index Advisor, 5-3  
isolation, 1-6  
    read committed, 4-1  
    serializable, 4-2  
    transactions, 4-1

## J

---

JDBC interface, 1-4  
join  
    merge, 5-6  
    methods, 5-4  
    nested loop, 5-5

## K

---

K-safety  
    number of copies of data, 1-12

## L

---

linguistic sorting, 1-7  
locks  
    database level, 4-4  
    description, 4-3  
    row level, 4-3  
    table level, 4-4  
log files  
    interaction with checkpoints, 3-7  
    when deleted, 6-2  
logging, 1-5  
    transaction, 6-1

## M

---

management  
    centralized, 1-12  
materialized view  
    and XLA, 8-3  
    comparing with other databases, 8-3  
memory usage  
    query optimization, 5-2  
migrate, 9-4

## O

---

OCI support, 1-4  
ODBC Administrator, 9-4  
ODBC interface, 1-4  
optimizer  
    description, 5-1  
    hints, 5-3  
    plan, 5-7  
    scan methods, 5-4  
Oracle Call Interface support, 1-4  
Oracle Clusterware, 6-9  
overview, 6-6

## P

---

parallel replication, 6-4  
passthrough feature, 7-5  
PL/SQL support, 1-3  
Pro\*C/C++ Precompiler support, 1-4  
processes  
    database, 3-4  
propagate  
    changes from TimesTen Cache to Oracle  
        database, 7-4

## Q

---

query optimizer, 1-6  
    description, 5-1  
    hints, 5-3  
    memory usage, 5-2  
    plan, 5-7  
    using statistics, 5-2

## R

---

range index, 5-3  
read committed isolation  
    description, 4-1  
read-only cache group, 7-2  
recovery  
    using checkpoint files, 6-3  
refresh  
    manual (cache group), 7-4  
replication  
    active standby pair  
        automatic client failover, 6-10  
        high availability, 3-7  
        overview, 6-4

- architecture, 3-7
- classic, 6-6
  - bidirectional, 6-8
  - distributed workload, 6-8
  - multiple subscribers, 6-6
  - propagation to subscribers, 6-7
  - split workload, 6-8
  - unidirectional, 6-6
- failover, 6-9
- high availability, 6-4
- number of threads, 6-4
- overview, 6-4
- parallel, 6-4
- support, 1-13
- TimesTen daemon, 3-4

## S

---

- scalability, 1-12
- scan methods, 5-4
- serializable isolation
  - description, 4-2
- shared libraries, 3-3
- shared-nothing architecture, 1-10
- SNMP traps, 8-4
- SQL Developer, 9-3
  - create grid, 1-8
  - manage grid, 1-8
- statistics
  - query optimizer, 5-2
  - ttOptEstimateStats, 5-2
  - ttOptUpdateStats, 5-2
- subdaemons
  - minimum required, 3-4
  - services performed, 3-4
  - setting allowable number, 3-4
  - specifying allowable range, 3-4
- SWT cache group, 7-2
- system tables
  - described, 1-8, 9-3

## T

---

- TimesTen
  - architecture, 3-1
  - scenarios, 2-1
  - using, 2-1
- TimesTen Application-Tier Database Cache, 1-1
- TimesTen Cache, 1-14
  - architecture, 3-1
  - scenarios, 2-1
  - using, 2-1
- TimesTen Grid
  - fault tolerance, 1-12
  - high availability, 1-12
  - scalability, 1-12
- timesten.conf file, 3-4
- timestend, 3-4
- transaction isolation
  - overview, 4-1

- read committed, 4-1
- Transaction Log API, 8-1
  - overview, 1-14
- transaction log buffer
  - writing to disk, 6-1
- transaction logging, 1-5, 6-1
- transactions
  - recovery, 1-5
  - replication, 1-5
  - rollback, 1-5
- TTClasses, 1-5
- ttGridAdmin utility, 1-12
- ttLoadFromOracle built-in procedure, 1-8
- ttMigrate utility, 9-5
- ttOptEstimateStats built-in procedure
  - statistics computing, 5-2
- ttStats utility
  - monitor database, 1-8

## U

---

- upgrade, 9-4
  - offline, 9-4
  - online, 9-5
- user managed cache group, 7-3

## X

---

- XLA, 8-1
  - materialized views, 8-3
  - overview, 1-14

