Oracle TimesTen In-Memory Database Recommended Programming Practices

Release 6.0

Methods for designing a TimesTen application for maximum stability and performance

B25273-03

ORACLE

TIMESTEN

March 2006

Printed in the United States of America

Contents

About this Guide

This document describes in detail how to develop an application with TimesTen that has maximum performance and maximum robustness.

Conventions used in this guide

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

TimesTen documentation uses these typographical conventions:

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

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

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

`%`	The percent sign indicates the UNIX shell prompt.
`#`	The number (or pound) sign indicates the UNIX root prompt.

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

If you see...	It means...
install_dir	The path that represents the directory where the current release of TimesTen is installed.
TTinstance	The instance name for your specific installation of TimesTen. Each installation of TimesTen must be identified at install time with a unique alphanumeric instance name. This name appears in the install path. The instance name “giraffe” is used in examples in this guide.
bits or *bb*	Two digits, either 32 or 64, that represent either the 32-bit or 64-bit operating system.
release or *rr*	Two digits that represent the first two digits of the current TimesTen release number, with or without a dot. For example, 51 or 5.0 represents TimesTen Release 5.0.
jdk_version	Two digits that represent the version number of the major JDK release. For example 14 for versions of jdk1.4.
`timesten`	A sample name for the TimesTen instance administrator. You can use any legal user name as the TimesTen administrator. On Windows, the TimesTen instance administrator must be a member of the Administrators group. Each TimesTen instance can have a unique instance administrator name.
DSN	The data source name.

TimesTen documentation

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

- The *Oracle TimesTen In-Memory Database Installation Guide* provides information needed to install and configure TimesTen on all supported platforms.
- The *Oracle TimesTen In-Memory Database Architectural Overview*provides a description of all the available features in TimesTen.
- The *Oracle TimesTen In-Memory Database Operations Guide*provides information on configuring TimesTen and using the ttIsql utility to manage a data store. This guide also provides a basic tutorial for TimesTen.
- The *Oracle TimesTen In-Memory Database C Developer's and Reference Guide* and the *Oracle TimesTen In-Memory Database Java Developer's and Reference Guide* provide information on how to use the full set of available features in TimesTen to develop and implement applications that use TimesTen.
- The *Oracle TimesTen In-Memory Database Recommended Programming Practices* provides information that will assist developers who are writing applications to work with TimesTen.
- The *Oracle TimesTen In-Memory Database API and SQL Reference Guide* contains a complete reference to all TimesTen utilities, procedures, APIs and other features of TimesTen.
- The *Oracle TimesTen In-Memory Database TTClasses Guide* describes how to use the TTClasses C++ API to use the features available features in TimesTen to develop and implement applications that use TimesTen.
- The *TimesTen to TimesTen Replication Guide*. This guide is for application developers who use and administer TimesTen and for system administrators who configure and manage TimesTen Replication. This guide provides background information to help you understand how TimesTen Replication works and step-by-step instructions and examples that show how to perform the most commonly needed tasks.
- The *TimesTen Cache Connect to Oracle Guide* describes how to use Cache Connect to cache Oracle data in TimesTen. This guide is for developers who use and administer TimesTen for caching Oracle data. It provides information on caching Oracle data in TimesTen

data stores. It also describes how to use the Cache Administrator, a web-based interface for creating cache groups.

- The *Oracle TimesTen In-Memory Database Troubleshooting Procedures Guide* provides information and solutions for handling problems that may arise while developing applications that work with TimesTen, or while configuring or managing TimesTen.

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

Background reading

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

- Hamilton, Cattell, Fisher. *JDBC Database Access with Java*. Reading, MA: Addison Wesley. 1998.

For a Java reference, see:

- Horstmann, Cornell. *Core Java*. Palo Alto, CA: Sun Microsystems Press. 1999.
- For the JDBC API specification, refer to java.sql package in the appropriate Java Platform API Specification.
- If you are working with JDK, refer to the specification for your version of JDK at: `http://java.sun.com/apis.html`

An extensive list of books about ODBC and SQL is in the Microsoft ODBC manual included in your developer's kit. In addition to this guide, your developer's kit includes:

- ***SQL***—*Oracle TimesTen In-Memory Database API and SQL Reference Guide* is a complete reference to TimesTen SQL.
- ***ODBC***— *Microsoft ODBC 2.0 Programmer's Reference and SDK Guide* provides general information on ODBC.

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

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

For a review of SQL, see:

- Jim Melton and Alan R. Simon. *Understanding the New SQL: A Complete Guide*. San Francisco, CA: Morgan Kaufmann Publishers. 1993.

For information on Unicode, see:

- The Unicode Consortium, *The Unicode Standard, Version 4.0*, Addison-Wesley, 2003.
- The Unicode Consortium Home Page at `http://www.unicode.org`

Technical Support

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

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

Email: timesten-support_us@oracle.com

1

Overview

This chapter includes the following topics:

Terminology used in this document

This section includes terms and definitions used in this document:

“TimesTen application” or “direct-linked application”

This term denotes an application that is direct-linked to a TimesTen data store. C and C++ applications explicitly link in one of the following shared libraries:

- `libtten.so` (Solaris, Linux)
- `libtten.sl` (HP-UX)
- `tten60.lib` (Windows)

Java applications that use TimesTen in “direct mode” connect to a data source of the form “`jdbc:TimesTen:direct:...`”

These applications reside on the same machine as the TimesTen data store they are accessing. Because of their location and because they use the “direct” TimesTen drivers, these applications see better performance from TimesTen than the client/server applications described in the next section.

“TimesTen client/server application”

This term denotes an application that uses the TimesTen client/server interface to connect to a data store. The application explicitly links in one of the following shared libraries:

- `libttclient.so` (Solaris, Linux)
- `libttclient.sl` (HP-UX)
- `ttcl60.lib` (Windows)

Java applications that use TimesTen in client/server mode connect to a data source of the form “`jdbc:TimesTen:client:...`“

Client/server applications can reside on the same machine as the TimesTen data store or can reside on other machines. Client/server applications run significantly slower than direct-linked applications.

Whether applications run direct-linked or as client/server, the APIs and functionality available to them are the same.

install_dir

This is the directory where you installed TimesTen. See *Oracle TimesTen In-Memory Database Installation Guide* for more information. By default, this directory is:

- `/opt/TimesTen/`*`InstanceName`* (UNIX; instance installed by root)
- `$HOME/TimesTen/`*`InstanceName`* (UNIX; instance installed by non-root user)
- `C:\TimesTen\tt60` (Windows)

Note: For security reasons, install TimesTen as a non-root user where possible.

C++ users: consider using TTClasses

The TimesTen C++ Interface Classes, known as TTClasses, was written to provide an easy-to-use, high performance interface to TimesTen. The C++ class library provides “wrappers” around most of the common ODBC functionality including SQL query execution, event notification (XLA), and system catalog information. In addition, TTClasses design incorporates many of the recommended practices covered in this document. TTClasses is included with the Oracle TimesTen In-Memory Database.

TimesTen ships with several useful example programs that demonstrate how to use TTClasses. Among the example programs are programs that

demonstrate how to monitor and administer a TimesTen data store; how to implement an event notification program using TimesTen's XLA; how to estimate the data store's size; and how to administer XLA bookmarks.

Operating system security considerations

There are two mutually exclusive modes of operation for TimesTen that have OS security implications.

- ***Non-root installation*** (available on all non-Windows platforms). In general, it is safer not to run any processes as a privileged user, such as root, unless absolutely necessary. When performing nonroot installations, certain procedures must be performed as user `root`. See the "Prerequisites for non-root installations and Access Control on UNIX systems" in the *Oracle TimesTen In-Memory Database Installation Guide*.
- ***GroupRestrict mode.*** When a data store is first created, it can be created in **GroupRestrict** mode so that all of its files and shared memory segments are ownership restricted to that of a particular operating system group. This mode works only if TimesTen is installed and running as `root`. See the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

2

Maximizing Performance

This chapter recommends coding practices for a TimesTen application for maximum performance. The information in this chapter builds on the information found in "Data Store Performance Tuning" in the *Oracle TimesTen In-Memory Database Operations Guide*.

Recommendations for maximizing performance

This section includes the following recommendations:

- Run performance-critical applications directly linked
- Prepare all SQL statements in advance
- Control the frequency of disk writes
- Create the right indexes for your queries
- Use “showplan” to verify that the right indexes are used for queries
- Turn autocommit off and commit regularly
- C/C++ (ODBC) vs. Java (JDBC) Performance
- Speed up large data loads by creating indexes after loading data
- Avoid OLEDB / ADO / third party middleware; use TTClasses instead

Run performance-critical applications directly linked

By using TimesTen’s “direct” ODBC and JDBC drivers, and by running the application on the same machine as the data store, it is simple to get the maximum possible performance from TimesTen. It is possible for applications on other machines to access the TimesTen data store by using the client/server features, but the overhead of network traffic will significantly reduce performance.

Using replication and Cache Connect to Oracle enables you to access data locally without incurring the overhead of client/server access.

Note: TimesTen running in a client/server architecture is not as fast as TimesTen in direct-memory mode, but it is still fast. TimesTen client/

server applications perform better than applications connected to traditional client/server databases; in particular, TimesTen is able to achieve significantly higher throughput compared with traditional databases.

Applications should consider connecting to a TimesTen data store by using a Client/Server connection in the following situations.

- The application must run on a different host from the host where the TimesTen data store resides.
- A 32-bit client application must connect to a 64-bit TimesTen data store and the 32-bit client application cannot be recompiled in 64-bit. If the application is or can be relocated to the same host as the TimesTen data store, but cannot be recompiled in 64-bit mode, TimesTen recommends considering the use of the TimesTen shared memory interprocess communications (SHMIPC). SHMIPC provides significant performance improvements over TCP/IP connects for client/server data store connectivity.

If you design a system that requires a large number of TimesTen client/server connections, you should be aware that each data store connection consumes operating system resources on the server host. This factor should be included in the sizing and operating system tuning of the server host.

Prepare all SQL statements in advance

For best performance, all SQL statements that are executed more than once should be prepared in advance. This is true for all relational databases, but for TimesTen and its extremely fast transaction rates, the time spent to compile a statement can actually take several times longer than it takes to execute it. In addition to preparing statements in advance, input parameters and output columns for those statements should also be bound in advance:

- ODBC users use the `SQLPrepare` function to prepare statements in advance.
- TTClasses users use the `TTCmd::Prepare()` method to prepare statements in advance.
- JDBC users use the `PreparedStatement` class to prepare statements in advance.

The TimesTen query optimizer in general is very good at choosing good query plans; however, it needs additional information about the tables involved in complex queries in order to choose good plans.

Data store statistics are an essential aid to the optimizer for choosing a good plan. By knowing the number of rows and data distributions of column values for a table, the optimizer has a much better chance of choosing an efficient query plan to access that table.

It is generally a good idea to update statistics on all tables in your data store before preparing queries that will access those tables.

Note that if you update statistics on your tables before they contain rows, then the queries will be optimized with the assumption that the tables contain no rows (or very few). If you later populate your tables with millions of rows and then execute the queries, the plans that worked well for tables containing few rows may now be very slow.

Thus, you should update statistics on your tables *after* loading them with data, but *before* preparing your queries.

For more information about updating statistics, please consult the descriptions of **ttOptUpdateStats** and **ttOptEstimateStats** in "Built-In Procedures" in *Oracle TimesTen In-Memory Database API and SQL Reference Guide*. Further useful information can be found in the descriptions of the other `ttOpt*` built-in procedures in that same chapter.

Control the frequency of disk writes

Most applications require some level of data store durability. That is, if the system crashes because of hardware or software failure, applications require that recent updates to the data store not be lost. There are several ways to configure TimesTen using the **DurableCommits** connection attribute:

- The first option is to connect to the TimesTen data store with the connection attribute **DurableCommits**=1. This causes every commit in that connection to flush the transaction log to disk; as a result, your application has complete durability. However, setting **DurableCommits**=1 causes the application's write performance to be significantly reduced. Note that good overall throughput can still be achieved with **DurableCommits**=1 if there are numerous concurrent writers in the data store.
- The second option is to allow the TimesTen data store to flush its in-memory transaction buffer to disk whenever it becomes full (the default behavior, or **DurableCommits**=0). This option allows for maximum performance, because disk writes occur only when absolutely necessary, and transactions will never block on disk writes. However, the application that takes this approach cannot

know for certain its level of transactional durability (beyond the fact that it cannot lose more than the in-memory log buffer's size worth of transactions).

For applications using **DurableCommits**=0, the **ttDurableCommit** built-in procedure allows the application developer to decide precisely when the TimesTen in-memory log buffer is flushed to disk, so that the appropriate level of durability and performance can be explicitly managed. Applications can choose which transactions should result in synchronous disk writes and which should not. In this way applications can carefully balance performance and data durability. For some applications it makes sense to call **ttDurableCommit** at specific time intervals, so that a guarantee of only, for example,100 milliseconds of unflushed log buffer can be lost at a time. Other applications, such as a funds transfer between two accounts, require the data store to be in a durable state.

TimesTen replication or propagation of updates from TimesTen to Oracle can be used to provide additional levels of transaction durability, regardless of the **DurableCommits** setting chosen.

Create the right indexes for your queries

Knowing how many indexes to create for good data store performance is a bit tricky. If you create too few indexes, then some of your frequent data store operations will perform more slowly than usual. If you create too many indexes, then insert/update/delete operations take longer because of the extra time needed to update the indexes.

There are a few issues to consider when designing your TimesTen table and index schema.

- There are two types of indexes: hash indexes and T-tree indexes. A well-tuned hash index is faster than the corresponding T-tree index for exact match lookups, but hash indexes cannot be used for range queries (T-tree indexes can be used for both exact match and range lookups, and for sorts, such as for SQL queries involving ORDER BY, GROUP BY, or DISTINCT).
- The primary key of each table is indexed with a hash index.
- Hash indexes need to be tuned. Use the PAGES= option of the CREATE TABLE statement to specify the expected size of the table. Divide the number of rows expected in the table by 256 for the number of pages to specify. Specifying too many pages for the index wastes space; specifying too few rows hurts the performance of the hash index because the buckets overflow. ALTER TABLE can be used to change the size of the primary key index later if required.

- Full table scan performance over a table with zero T-tree indexes can be improved by including a T-tree index (*any* T-tree index), even if the table scan does not reference columns in that index. This is not intuitive but is easily demonstrated. Thus, you should consider building at least one T-tree index for every table referenced by your application's full table scans.
- A hash index on three columns can be used only for an exact match lookup of all three columns. Any leading prefix of the columns in a T-tree index can be used for an exact match lookup.
- Hash indexes provide better performance over T-tree indexes for equality predicates in cases where either one can be used. However, hash indexes require more space than T-tree indexes.

  For example:

```
SELECT … FROM T1 WHERE COL1 = ? AND COL2 = ?
```

Situation 1 There are two indexes on T1:

- Hash index: (COL1, COL2)
- T-tree index: (COL1, COL2, COL3)

In this case, both indexes can be used to answer this query.

The hash index can be used because the columns in the WHERE clause exactly match the columns in the hash index. The T-tree index can also be used to answer the query because the columns in the WHERE clause are the leading prefix (the first two) of the columns in the index.

The TimesTen optimizer chooses the hash index because it is faster.

Situation 2 There are two indexes on T1:

- Hash index: (COL1, COL2, COL3)
- T-tree index: (COL1, COL2)

In this case, only the T-tree index can be used to answer the query.

The hash index cannot be used because one of the columns in the hash index (COL3) is not specified in the WHERE clause of the query. The T-tree indexed columns, however, exactly match the columns in the WHERE clause of the query.

The TimesTen optimizer chooses the T-tree index.

Situation 3 There are two indexes on T1:

- Hash index: (COL1)
- T-tree index: (COL3, COL1, COL2)

In this case, only the hash index can be used to answer the query.

The hash index can be used because all of its columns are contained in the WHERE clause of the query. Note that there are additional columns in the WHERE clause of the query, so every row read from the hash index will have the “COL2 = ?” predicate applied before the query result is returned.

The T-tree index cannot be used because the T-tree is first sorted by COL3, and the query does not refer to COL3.

The TimesTen optimizer chooses the hash index.

Situation 4 There are two indexes on T1:

Hash index: (COL1, COL2, COL3)

T-tree index: (COL3, COL1, COL2)

In this case, neither index can be *efficiently* used to answer the query, although the T-tree index can be used for a full table scan.

The hash index cannot be used for the same reason as in Situation 2. The T-tree index cannot be used because the columns in the query (COL1, COL2) are not a leading prefix of the index.

Since neither index can be used, the data store must perform a table scan to answer the query (or possibly it will create a temporary index), and as a result the query’s performance is poor.

As you can see from these examples, you have to choose carefully which indexes you create based on the queries that you plan to ask the data store most frequently.

For more information about creating the right indexes, see "Tune statements and use indexes" and "Select hash or T-tree indexes appropriately" in *Oracle TimesTen In-Memory Database Operations Guide*.

Use “showplan” to verify that the right indexes are used for queries

If you find that a specific query runs more slowly than expected, it is possible that the TimesTen optimizer is not choosing the optimal query plan to answer that query. See "Viewing a plan" in *Oracle TimesTen In-Memory Database Operations Guide* for more information about how to look at the plan generated by your query.

Turn autocommit off and commit regularly

By default in ODBC and JDBC, all connections to a data store have autocommit turned on. This means that each individual SQL statement is committed in its own transaction. By turning autocommit off and explicitly committing multiple SQL statements in a single transaction, an application's performance can be greatly improved. This makes particular sense for a large operation, such as a bulk insert or bulk delete. (Users of TTClasses will notice that autocommit is turned off by default.)

However, it is also possible to group *too many* operations into a single transaction (thus reducing overall system concurrency, as locks are held for an overly long time). In general, bulk insert/update/delete operations tend to work most effectively when you commit every few thousand rows.

C/C++ (ODBC) vs. Java (JDBC) Performance

TimesTen's native API is ODBC. The TimesTen Java JDBC driver is a type II driver built on top of the TimesTen ODBC driver. Thus, C/C++ applications that use TimesTen's ODBC driver will experience somewhat better performance compared with Java applications using TimesTen's JDBC driver. Applications that require the highest levels of performance should be developed using the TimesTen ODBC driver.

Note: TimesTen's JDBC driver is very fast. TimesTen's JDBC performance is significantly higher than the JDBC performance of other databases. TimesTen's JDBC driver is only somewhat slower than TimesTen's ODBC driver; but the differences are measurable.

The expected difference in performance results between ODBC and JDBC are influenced by the types of queries executed (SELECT vs. UPDATE/INSERT/DELETE) and the transaction mix (reads to writes). The factors affecting performance include:

- The number of bound parameters for prepared statements - increasing the number of bound parameters tends to decrease Java application performance.
- The number of columns returned by a SELECT statement - increasing the number of select columns tends to decrease Java application performance.
- The types of parameters and columns - data types consisting of a larger number of bytes are typically slower in JDBC.

Fundamentally, a Java application has reduced performance when the amount of data that is transferred between the application and the data

store increases. For specific information on expected performance differences between ODBC and JDBC, contact TimesTen support or your TimesTen account team.

There are other issues inherit in Java that influence the performance of a Java application:

- JVM garbage collection - Java application can experience response time spikes during JVM garbage collection. This garbage collection can make it difficult to guarantee consistent performance for high performance “real-time” applications. Careful tuning of JVM garbage collection is essential to minimize these performance dips.
- Java's runtime memory management - The design of the Java runtime memory management subsystem can often result in hidden data copying, particularly in the case of String objects.You should also ensure that there is sufficient Java heap space.
- Scaling of heavily multithreaded Java applications - TimesTen’s experience has been that heavily multithreaded Java applications do not scale as well as expected. This is due in part to the design of the JVM thread scheduler compared to the operating system thread scheduler. In many cases, using multiple JVMs can give better overall performance than a single JVM executing numerous threads.

Certain JVMs are faster than others. Evaluate your choices carefully.

Use the MAXROWS option for SELECT statements when returning large result sets.

Speed up large data loads by creating indexes after loading data

If your application design allows it, you can minimize the time it takes to load data by creating T-tree indexes *after* loading the data. In this situation, the sequence of operations should be:

1. Load data into tables. (Disabling logging and using data store-level locking provides better performance during the table load operation.)
2. Create T-tree indexes.
3. Update statistics.
4. Prepare queries.

   Note that this is relevant only for very large data load operations (millions of rows).

Avoid OLEDB / ADO / third party middleware; use TTClasses instead

Many third-party database interface packages impose a penalty on data store performance. These interfaces can be used if performance is not critical in a particular application, but users should be aware of the performance tradeoff.

Because these third-party packages can be slow, TimesTen has developed its own C++ ODBC wrapper classes named TTClasses, which is included with TimesTen. For more information on TTClasses, see *Oracle TimesTen In-Memory Database TTClasses Guide*.

Tuning performance for multiple CPUs

There are additional things that an application must do correctly in order to obtain maximum SMP performance. This section is devoted to those additional items that are required for maximum performance on a multiple-CPU machine.

Use connection pooling

If your TimesTen application is multithreaded and opens multiple connections to the same data store, you should carefully manage your connections. In general, it is hard to achieve the best performance if there are more connections active simultaneously on a data store than CPUs on your machine. In fact, having more concurrent TimesTen connections to the same data store can result in significantly lower overall data store throughput.

One approach for avoiding this problem is to use connection pooling. A good example of connection pooling is the `TTConnectionPool` class in TTClasses. For more information on TTClasses see *Oracle TimesTen In-Memory Database TTClasses Guide*.

Maximizing data store concurrency

Users can impact the performance of other users if they use too many system resources. This section includes the following recommendations:

- Close cursors promptly
- Avoid large delete statements
- Shorten unnecessarily long transactions
- Use XLA acknowledgement modes effectively

Close cursors promptly

Similar to the Oracle Database, TimesTen read-only transactions do not need to be committed. However, it is important to close read-only cursors promptly in order to release all resources held by a read-only SQL query (such as temporary space used to do a sort).

The following methods close a SQL cursor:

- For ODBD, use SQLFreeStmt(SQL_CLOSE)
- For TTClasses, TTCmd::Close()
- For JDBC, PreparedStatement.close()

Avoid large delete statements

Consider the following ways to avoid large delete statements:

- Avoid DELETE FROM statements
- Prefer the TRUNCATE TABLE statement
- Consider using the DELETE FIRST clause

Avoid DELETE FROM statements

If there are many rows in a table (100,000 or more), attempting to delete them in a single SQL statement (in a single transaction) can be problematic. First, this approach is slow. TimesTen logs each row deleted, in case the operation needs to be rolled back, and writing all of those log records can be very time-consuming because it is a disk-bound operation.

Another problem with such a huge delete operation is that other data store operations will be slowed down while the write-intensive delete transaction is occurring. Deleting millions of rows in a single transaction can take minutes to complete.

A third problem with deleting millions of rows at once occurs when the table is being replicated. Because replication transmits only committed transactions, the replication agent can be slowed down by transmitting a single, multi-hundred MB (or GB) transaction. TimesTen's replication has been optimized for lots of small transactions, and performs slowly when millions of rows are deleted in a single transaction.

Prefer the TRUNCATE TABLE statement

Rather than deleting all rows in a table, consider using the TRUNCATE TABLE statement. This statement has the same final effect as a DELETE with no WHERE clause, with substantially less logging. In addition, when using replication, the TRUNCATE operation is replicated to another data store as a single operation, rather than one operation for each deleted row.

Consider using the DELETE FIRST clause

When large numbers of rows must be deleted and TRUNCATE is not appropriate, consider using the DELETE FIRST *NumRows* clause to delete rows from a table in batches. The DELETE FIRST *NumRows* syntax allows you to change "DELETE FROM *TableName* WHERE ..." into a sequence of "DELETE FIRST 10000 FROM *TableName* WHERE ..." operations.

By splitting a huge delete operation into a batch of smaller operations, the rows will be deleted much faster, and the system's overall concurrency and replication will not be affected.

Shorten unnecessarily long transactions

As described in "Handle deadlock and lock timeout errors" on page 30, a long-running transaction can cause other data store operations to fail with lock timeout errors. In general, long-running transactions reduce overall system concurrency because resources remain locked for a long period of time. Thus long-running transactions can hurt both overall system stability and overall system performance.

Use XLA acknowledgement modes effectively

The XLA acknowledgement mechanism is designed to ensure that an application not only receives a message but successfully processes it. Acknowledging an update permanently resets the application's XLA bookmark to the last record that was read. This prevents previously returned records from being read again, ensuring that an application does not receive previously acknowledged records if the bookmark is reused when an application reconnects to XLA.

JMS/XLA can automatically acknowledge XLA update messages, or applications can choose to acknowledge messages explicitly. Specify how updates are to be acknowledged when you create the `Session`. JMS/XLA supports three acknowledgement modes:

- AUTO_ACKNOWLEDGE—Updates are automatically acknowledged when they are received. Each message is delivered only once. In AUTO_ACKNOWLEDGE mode, JMS/XLA does not prefetch multiple records. (The `xlaprefetch` attribute in the topic is ignored.)
- DUPS_OK_ACKNOWLEDGE—Updates are automatically acknowledged, but duplicate messages may be delivered if the application fails. JMS/XLA prefetches records according to the `xlaprefetch` attribute specified for the topic and sends an acknowledgement when the last record in a prefetched block is read. If the application fails before reading all of the prefetched records, then all of the records in the block are presented to the application when it restarts.
- CLIENT_ACKNOWLEDGE—Applications are responsible for acknowledging receipt of update messages by calling `acknowledge` on the `MapMessage`. JMS/XLA prefetches records according to the `xlaprefetch` attribute specified for the topic.

Prefetching updates

Prefetching multiple update records at a time is more efficient than obtaining each update record from XLA individually. If possible, you should design your application to tolerate duplicate updates so you can use DUPS_OK_ACKNOWLEDGE or explicitly acknowledge updates.

Acknowledging updates

To explicitly acknowledge an XLA update, call `acknowledge` on the update message. Acknowledging a message implicitly acknowledges all previous messages. If you are using the CLIENT_ACKNOWLEDGE mode and intend to reuse a durable subscription in the future, you should call `acknowledge` to reset the bookmark to the last-read position before exiting.

3

Maximizing Stability

This chapter focuses on how to code an application for maximum robustness and stability. A stable application stays connected to a data store and continues operations in a predictable manner.

This chapter includes the following topics:

TimesTen architecture and data store recovery

The contents of a data store reside in a shared memory segment while the data store is in memory. When an application attaches to a TimesTen data store, it maps the shared memory segment of that data store into its address space. The use of shared memory allows multiple processes to connect to a single data store image.

In traditional applications that use shared memory, the stability of each application process connected to a shared memory segment impacts the consistency of the segment. An unstable application can corrupt a shared memory segment, which causes application failures. Corruption can also occur when an application is abruptly terminated (for example, by the "`kill -9`" command) while it is updating shared memory. This can leave the shared memory in an inconsistent state, with data structures only partially updated.

TimesTen avoids these problems by using Micrologging™ technology that protects other applications from abrupt application failure. If an application fails while modifying data in a TimesTen data store, TimesTen can detect the condition and roll back the partial update, thus restoring the shared memory segment to a consistent state.

While TimesTen can recover from abrupt application failures, doing so takes resources that would be better spent running your applications.

Avoiding this overhead is easy to do by following a few simple steps in application development.

Back up the data store

Taking periodic backups of the data store is essential for avoiding data loss. Backing up the data store and storing the backups on removable media or on a second set of permanently mounted media reduces the opportunity for data loss if something were to happen to the hardware, such as the failure of a disk drive.

You can use the TimesTen **ttBackup** utility to back up the data store. Backups do not acquire locks, so they can proceed in the background while normal data store operations continue.

Understand checkpoint behavior

Periodic checkpointing is important to the operation of a TimesTen data store. Tuning checkpoint frequency can benefit some applications.

TimesTen maintains two complete images of the contents of a data store on disk. These images are called *checkpoints*. TimesTen automatically updates these checkpoints on disk to synchronize them with the current data store contents. This is done in the background, asynchronously, without application involvement. These periodic checkpoints are important because:

- Checkpoints remove unnecessary log files from the log directory. Periodic checkpoints thus prevent the accumulation of log files. Excessive log file accumulation can cause the disk to fill up, preventing TimesTen operations from continuing. See "Recovering from a full disk" on page 30 on the importance of monitoring disk space.
- Checkpoints reduce the amount of log data needed for data store recovery, thus reducing the amount of downtime after a system failure.

By default, TimesTen checkpoints the data store every 600 seconds (10 minutes), or whenever 64 megabytes of log data have been written, whichever comes first. This behavior can be changed if needed by using the **CkptFrequency** and **CkptLogVolume** data store attributes.

Other good practices

This section includes the following recommendations:

- Check and handle return codes from all ODBC functions
- Recovering from a full disk

Check and handle return codes from all ODBC functions

When using ODBC, it is essential to check the return codes from all ODBC functions. If warnings or errors are detected, then the application should handle these appropriately. A few of the most important errors that can arise are described in the next two sections.

For a complete list of error codes and the mnemonics that TimesTen uses to describe these codes, see the *install_dir*/include/tt_errCode.h file.

For a brief description of how an application can handle errors in general, see "Transaction Management and Recovery" in *Oracle TimesTen In-Memory Database Operations Guide*.

When using JDBC (Java) and TTClasses (C++), exceptions are returned for most data store-related error conditions. These errors must be handled through appropriate try/catch blocks.

Handle data store invalidation errors

Data stores can become unavailable for a number of reasons:

- An administrator may have terminated the TimesTen instance
- A fatal hardware error may have affected the data store
- An internal error may cause TimesTen to “invalidate” the data store, requiring all applications to disconnect and reconnect

There are two TimesTen error codes that indicate that an application is attached to a data store that has been invalidated: 846 (tt_ErrBadConnect) and 994 (tt_ErrDrtyByte). When an application receives either of these error codes, applications must roll back any transactions they have started, and then must disconnect from and reconnect to the data store before proceeding.

A simple but effective implementation of handling data store invalidation errors can be found in the C++ source code for (monitor.cpp) that ships with TimesTen.

Handle deadlock and lock timeout errors

A deadlock occurs when two concurrent transactions in a data store are competing for the same resources, and each of them holds a resource that the other needs to continue. When a deadlock occurs, the data store chooses one of the deadlocked transactions and denies its resource request, returning error code 6002 (tt_ErrDeadlockVictim).

A lock timeout occurs when two concurrent transactions in a data store are competing for the same resources, and one of them is unable to acquire a resource in a given amount of time. One common way for this to occur is when one transaction runs for a long period of time without committing (thus holding locks for a long time), and thereby preventing other transactions from acquiring locks on the resources that the long-running transaction has locks on. The transaction that is unable to acquire a lock receives TimesTen error code 6003 (tt_ErrTimeoutVictim).

Poorly designed applications can easily form deadlocks. For example, if multiple threads of execution often try to modify the same sets of rows at the same time, deadlocks may occur.

You can use the TimesTen **ttXactAdmin** utility to diagnose data store concurrency problems. See *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

TimesTen provides the following tools for application developers to use for reducing data store concurrency problems:

- TimesTen READ_COMMITTED isolation level, which enables "nonblocking reads". That is, readers do not block writers, and writers do not block readers.
- The SELECT ... FOR UPDATE SQL statement, which can help prevent deadlocks when an application performs a SELECT and then immediately updates the fetched rows.

For more information about these SQL statements, see SELECT and UPDATE in *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

Recovering from a full disk

When the disk containing the TimesTen transaction log fills up because of infrequent checkpoint operations or if replication or XLA has fallen behind, the data store may be invalidated and connections may not be able to reconnect automatically because there is no room to write subsequent log files. See "Handle data store invalidation errors" on page 29.

In general, when TimesTen transaction log files fill up a disk, it is necessary to free up some space on the disk and then connect to the data store and perform one or two quick checkpoint operations to delete the unnecessary log files that have accumulated.

Note that if replication or XLA caused the transaction log files to accumulate and fill the disk, then the replication or XLA bookmarks need to be cleared before the transaction log files can be purged successfully.

4

Replication and XLA

TimesTen replication is powerful and flexible enough to allow system designers to develop highly available, fault-tolerant applications.

TimesTen XLA is used to monitor changes to a table or tables within a TimesTen data store.

This chapter includes the following topics:

- Replication topics
- XLA topics

Replication topics

This chapter is focused on some of the issues that are important to understand when designing a system that uses TimesTen replication and XLA.

An additional source for replication information is the *Online Upgrades Guide*, another document available from TimesTen Professional Services.

Use the DSN as the file name prefix

This is not necessary, but it avoids unnecessary confusion.

The DSN is the data source name. See "Lesson 1: Defining a data source name (DSN)" in *Oracle TimesTen In-Memory Database Operations Guide*.

The file name prefix is defined in the next section.

Definition of “file name prefix”

On UNIX, the DSN called `myDSN` might look like the following:

```
[myDSN]
Driver=/opt/TimesTen/tt60/lib/libtten.so
Datastore=/directory/ds_file_prefix
```

The file name prefix of `myDSN` is `ds_file_prefix`.

On Windows, in the ODBC Control Panel, it is quite similar. In the field labeled "Data Store Path and Name", if you have configured `myDSN` to be in the following path:

```
C:\directory\ds_file_prefix
```

Then the file name prefix for `myDSN` is `ds_file_prefix`.

TimesTen replication configuration uses the file name prefix

The replication configuration commands refer to the file name prefix of a data store. It is used in the SQL-based replication configuration commands:

```
CREATE REPLICATION rep.name ELEMENT e TABLE rep.t1 MASTER
<file name prefix of some data store> ON <host name> …
```

The file name prefix of the data store is also used any time the **ttRepAdmin** command-line utility needs to refer to a data store's "name":

```
ttRepAdmin -duplicate -from <file name prefix of other data
store> -host <host name> dsn=myDSN
```

Furthermore, in TimesTen replication, the <*file name prefix*, *host name*> pair must be unique; that is, you cannot set up replication for two data stores on the same computer if they both have the same file name prefix.

Conclusion: always set the file name prefix equal to the DSN

Because of these reasons, it is very convenient to always set the file name prefix of a data store to be the same as its DSN:

- All of the replication configuration commands are easier to write if you do not have to keep track of both the DSN and the file prefix name.
- The <*file name prefix*, *host name*> pair is automatically unique when the file name prefix is the same as the DSN, because ODBC already guarantees that <*DSN*, *host name*> is unique.

Perform two checkpoints before performing a duplicate operation

TimesTen recommends issuing two explicit checkpoints on the master data store immediately before performing a **ttRepAdmin** `-duplicate` operation. Use the **ttCkpt** built-in procedure to force a checkpoint.

Performing checkpoints at the master data store purges unneeded transaction log files. This in turn reduces the amount of data that needs

to be transferred as part of the duplicate operation, thereby speeding up the duplicate operation.

Replication configuration should manually specify port numbers

If you ever expect to perform an online upgrade of your TimesTen application where you upgrade the version of TimesTen itself, your replication configuration commands and scripts must manually specify the replication port number.

If you let replication dynamically allocate a port number instead of specifying it manually, then different versions of TimesTen replication cannot to talk to each other successfully.

For more information about online upgrades, ask TimesTen Professional Services for their document titled *Online Upgrades*.

Monitor replication

It is important to monitor TimesTen replication in order to ensure high availability. There are a few places where your application should monitor its use of replication:

- If the link between the replication sender and receiver goes down (for example, the network connection is severed or the receiver node dies), then the sender needs to make a decision to stop replicating to that receiver. The point at which it decides depends entirely upon the application and its requirements. In general, if two nodes in a replication scheme become unsynchronized for a long time, it can be much faster to restore the receiver node by simply dropping its data store and then re-initializing by performing a **ttRepAdmin** `-duplicate` operation.
- If there is prolonged update/insert/delete activity at high transaction rates on the sender side of a replication scheme, then TimesTen replication can fall behind. Due to the nature of communication over a network, it is very easy to run TimesTen at a high insert/update/delete rate with which replication cannot keep up. TimesTen replication will work hard to catch up as soon as it can, but if the transaction rate never slows, replication will fall further and further behind.

Some types of applications may perform a large number of data store operations upon system initialization, and then the transaction rate may slow down to a lower rate. In such a situation, it may make more sense

to perform a **ttRepAdmin** `-duplicate` operation at the receiver nodes and start replication only after the sender node has been fully populated.

In both of these circumstances (communication with a replication receiver goes down, or replication falls behind), extra disk space on the sender side of replication is required to store those transaction logs which hold the data that is waiting to be replicated. In this situation, checkpoint operations will *not* delete log files, and as a result an application may become in danger of running out of disk space.

The mechanism that replication uses to hold onto log files that it needs, preventing them from being deleted by checkpoint operations, is called the replication bookmark. To see the replication bookmark of a data store at the command line, execute a **ttRepAdmin** `-bookmark` command. Your application should periodically check the replication backlog and take appropriate action when replication falls multiple log files behind.

Consider configuring replication with a threshold, so that it automatically stops saving logs when a subscriber gets too far behind.

For more information about **ttRepAdmin**, see Chapter 2, "Utilities" in *Oracle TimesTen In-Memory Database API and SQL Reference Guide*. Information about **ttBookmark** can be found in Chapter 3, "Built-In Procedures" in *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

Interaction of sequences with replication and failover

Sequences are independent on each node, and they are not replicated. On *db1*, a named sequence might be defined that starts at 1 and goes to 100000 On *db2*, a named sequence might be defined that starts at 100001 and goes to 200000. Thus, for example, the sequence at each node can be used to populate the primary key column of a table, so that each node can insert into that table without fear of conflicting with the other node.

After a link failure, if replication is allowed to recover normally (by catching up or by replaying saved log files), then nothing special must be done. However, if the failed node was down too long, then the recovery process is to use **ttRepAdmin** `-duplicate to` repopulate the data store on the failed node from the surviving one. (This is necessary if the queue of transactions gets too large.) The **ttRepAdmin** `-duplicate` command copies the sequence definition from one node to the other; the end result in our example is that both nodes would

produce sequence values between 100001 and 200000, which is exactly what we were trying to avoid.

To avoid duplicate sequence numbers, complete the following tasks in this order:

1. Execute **ttRepAdmin** `-duplicate`.
2. Drop and re-create any sequences, so that they once again have ranges that do not overlap.
3. Restart the application.

   By using this simple technique, sequences can be used in a replicated setting to insure uniqueness (but not density nor temporal monotonic increase) of a key value.

XLA topics

This section includes the following topics:

Use persistent XLA

There are two types of XLA supported by TimesTen: persistent XLA (when you connect by using the **ttXlaPersistOpen** function), and non-persistent XLA (when you connect by using the **ttXlaOpenTimesTen** function). In almost every respect, persistent XLA is superior to nonpersistent XLA.

Note: All discussion of XLA in this chapter refers to persistent XLA.

Always monitor XLA

Like replication, XLA requires careful monitoring.

The importance of monitoring XLA processes

Each XLA connection keeps a bookmark in the transaction log. If one of these XLA processes dies, its bookmark stays in the log, and that log file and all future log files cannot be deleted by a checkpoint operation until the bookmark is advanced or is deleted.

Thus a persistent XLA process that dies can cause log files to accumulate, which can cause the disk to become full, which can then cause data store failure when additional log files cannot be written. Recovering from a full disk is described in "Recovering from a full disk" on page 30.

TTClasses provides a useful command-line utility called `ttXlaAdmin`, which lists the XLA bookmarks in a data store (showing how many log files are being held by each), and allows a user to delete them, if desired. For more information on TTClasses see *Oracle TimesTen In-Memory Database TTClasses Guide*.

Use ttXlaAcknowledge to acknowledge XLA updates

XLA is made persistent by keeping a bookmark in the transaction log for what has been read (and what has not been read) by a particular XLA reader. Note that this bookmark is not advanced when an XLA reader receives a set of XLA records. This is intentional, in case the XLA reader dies before it has a chance to act on those records.

In order to move the bookmark forward, the XLA reader must explicitly acknowledge a batch of XLA records. This is done by using the **ttXlaAcknowledge** XLA function. If a user program does not call this function regularly, then the XLA bookmark will never move forward, and the disk space exhaustion problems described in the previous item will occur. However, this function should not be called too frequently, as described below in the section on XLA performance.

The TTClasses method `TTXlaPersistConnection::ackUpdates()` is a wrapper around **ttXlaAcknowledge**. See the TTClasses sample programs `xlaSimple` for a demonstration of using `ackUpdates`. For more information about TTClasses, see *Oracle TimesTen In-Memory Database TTClasses Guide*.

Maximizing XLA Performance

XLA is a flexible, powerful framework for receiving notifications about changes to a table or multiple tables within a data store. However, performance of your XLA application can be degraded if you do not take into consideration the following topic.

Use ttXlaAcknowledge, but not too often

The **ttXlaAcknowledge** XLA function is a fundamental part of any XLA application: it moves the XLA bookmark forward, allowing transaction logs to be purged by checkpointing.

However, **ttXlaAcknowledge** is relatively expensive, so it should not be called every time that **ttXlaNextUpdate** (or **ttXlaNextUpdateWait**) returns a batch of XLA records.

Knowing how often to call this function depends upon balancing application business logic with recovery and performance requirements. For applications with relatively low throughput requirements, it might be alright to call **ttXlaAcknowledge** every 10 times that **ttXlaNextUpdate**[`Wait`] returns records, but when maximum throughput is necessary, a 1:100 ratio between these two functions might be more advisable.

Index