# *Oracle TimesTen In-Memory Database Operations Guide*

# *Release 6.0*

**B25269-03**


For last-minute updates, see the TimesTen release notes.

March 2006

Printed in the United States of America

# Contents

## 3 Working with the TimesTen Client and Server

## 4 Working with the Oracle TimesTen Data Manager Daemon

## 5 Using the ttIsql Utility



## 6 Working with Data in a TimesTen Data Store

## 9 The TimesTen Query Optimizer



## 10 UNIX Configuration Files



## Index

# *About this Guide*

TimesTen® is a high-performance, in-memory data manager that supports the ODBC and JDBC interfaces.

This guide provides:

- Background information to help you understand how TimesTen works.
- Step-by-step instruction and examples that show how to perform the most commonly needed tasks.

To work with this guide, you should understand how database systems work and have some knowledge of SQL (Structured Query Language).

## 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 JDBC development information, 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 1.2, refer to the Java 2 Platform API specification at: `http://java.sun.com/products/jdk/1.2/docs/api/index.html`
- If you are working with JDK 1.3, refer to the Java 2 Platform API specification at: `http://java.sun.com/j2se/1.3/docs/api/index.html`
- If you are working with JDK 1.4, refer to the Java 2 Platform API specification at: `http://java.sun.com/j2se/1.4/docs/api/index.html`
- Siple, Matthew. *The Complete Guide to Java Database Programming: JDBC, ODBC and SQL*. McGraw-Hill. 1997.

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 the appropriate ODBC manual for your platform:


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


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

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

- Sanders, Roger E. *ODBC 3.5 Developer's Guide* (McGraw-Hill Series On Data Warehousing and Data Management); McGraw-Hill. 1999
- Signore, Robert / Stegman, Michael O. / et al. *ODBC Solution: Open Database Connectivity in Distributed Environments*: *Mcgraw-hill Series On Computer Communications*; McGraw Hill. 1995

For a review of SQL, see:

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

- Groff, James R. / Weinberg, Paul N. *SQL: The Complete Reference.* McGraw-Hill. /1999

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`

## Installing TimesTen

TimesTen Release 6.0 includes the TimesTen Data Manager for 32-bit and 64-bit platforms. See the *Oracle TimesTen In-Memory Database Installation Guide* for a description of supported platforms.

In addition to the Data Manager, TimesTen Release 6.0 also includes TimesTen Client and Server components. You can install the TimesTen Data Manager stand-alone or in a client/server environment.

For a list of the The TimesTen default installation directories, see the *Oracle TimesTen In-Memory Database Installation Guide*.

## 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.<br><br>For example, the `.odbc.ini`.`ttconnect.ini` file. |
| *`italic code font`* | A variable in a code example that you must replace.<br><br>For example:<br>`Driver=`*`install_dir`*`/lib/libtten.sl`<br>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, 50 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. |

## Finding the information you need

The table below provides a brief overview of the TimesTen development process. It will help you get started with your application and find relevant information as you progress.

| To learn how to | See |
|---|---|
| Install TimesTen | *Oracle TimesTen In-Memory Database Installation Guide* |
| Create and use a TimesTen data store | Chapter 1, "QuickStart" |
| Use the **ttIsql** command-line utility to execute SQL | Chapter 5, "Using the ttIsql Utility" |
| Manage TimesTen resources | Chapter 4, "Working with the Oracle TimesTen Data Manager Daemon" |
| Configure TimesTen on UNIX systems | Chapter 10, "UNIX Configuration Files" |
| Troubleshoot problems running the TimesTen demos | The *TimesTen Troubleshooting Procedures Guide*: http://www.oracle.com/technology/documentation/timesten_doc.html |

## 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

# QuickStart

This tutorial provides a basic grounding in development using TimesTen. The topics covered in this tutorial include:



## Overview

This tutorial walks you through the process of using the basic functions of TimesTen. The tutorial will teach you how to:

- Define a Data Source Name (DSN).
- Connect to a data store.
- Execute SQL statements using the TimesTen **ttIsql** utility.
- Work with other TimesTen utilities.

In general, you can set your own pace for completing the tutorial. However, TimesTen recommends completing an entire lesson in one sitting since each lesson builds on a sequence of related tasks. After you have finished the tutorial, you will have mastered the basics of using TimesTen.

### Before you begin

Before starting the lesson plan in the basic tutorial, you need to complete the installation of the software. Be sure that all software is installed according to the instructions found in the *Oracle TimesTen In-Memory Database Installation Guide*.

### Lessons

You are going to build a database containing data regarding information on a fictitious group of customers who purchase dictionaries. You will define a DSN, create a data store, build tables and populate the tables with data supplied within the demo directory. You will then work with the data store using various TimesTen utilities.

## Lesson 1: Defining a data source name (DSN)

### Data sources and data stores

A data store is a collection of TimesTen tables and indexes. The name of a data store is identified by the path name of its checkpoint and log files, which you enter when creating a DSN.

A data source is an ODBC entity that defines configuration information used to connect to a data store. The connection's configuration is determined by the settings of the data source's attributes.

A DSN is a logical name for a data source. The ODBC driver uses a DSN to connect to a data store. DSNs are case insensitive. The DSN uses the data store path name to map to the data store. More than one DSN can be mapped to the same data store to create different connection configurations to that data store.


### Defining a DSN on Windows

Use the ODBC TimesTen Setup dialog to define a DSN:

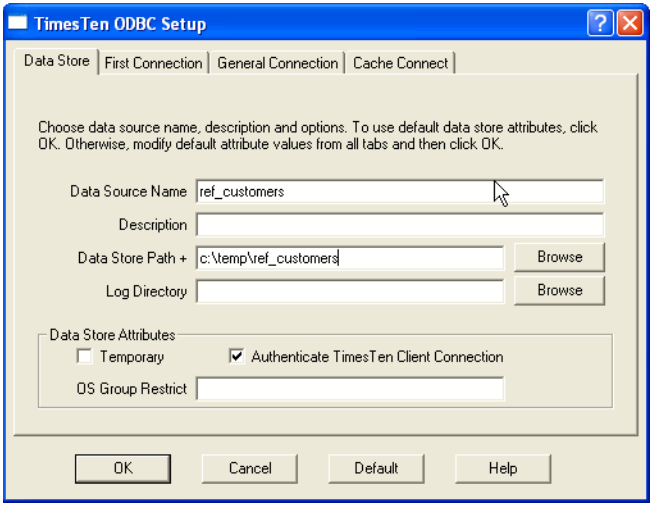
1. From the desktop, choose **Start > Control Panel > Administrative Tools > Data Sources (ODBC)**.
2. Double click **Data Sources (ODBC)**.
3. Click the **User DSN** or **System DSN** tab.
4. Click **Add**. The Create New Data Source dialog appears.
5. Choose **Oracle TimesTen Data Manager 6.0**.
6. Click **Finish**. The ODBC TimesTen Setup dialog appears.
7. Click **Data Store**.
8. In the **Data Source Name** field, type: `ref_customers`.

---

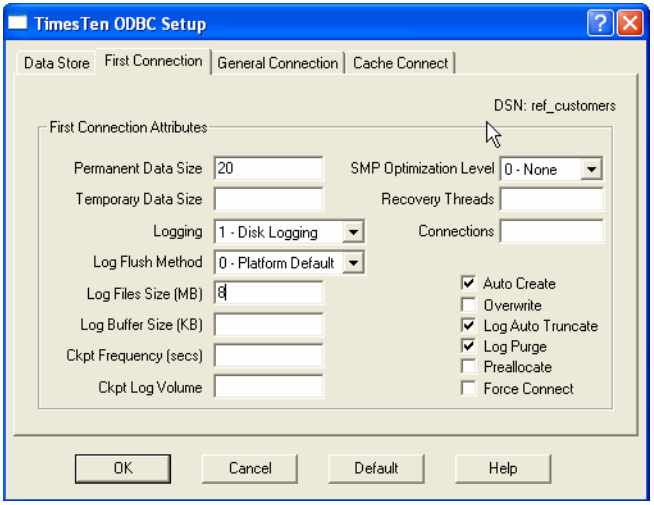
**Note:** The Data Source Name and the data store name do *not* need to be the same. They are presented here as such for simplicity.

---

9. In the **Data Store Path and Name** field, type `c:\temp\ref_customers`.



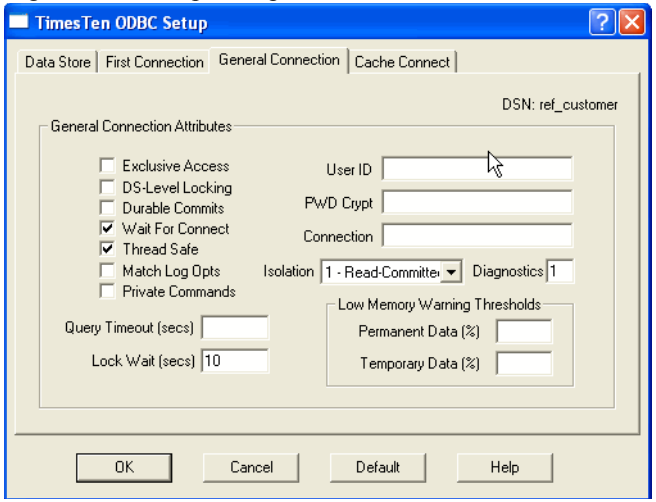

10. Click the **First Connection** tab.
11. In the **Permanent Data Size** field, type `20`. Size specifies the initial size, in megabytes, of the data store.
12. In the **Log Files Size (MB)** field, type `8`.




**Note:** Be sure **AutoCreate** is checked. Lesson 2 uses this attribute. On connection the data store will be created if it does not exist.

13. Click **General Connection**.

14. Make sure that **DurableCommits** is unchecked. This minimizes writing to the log buffer and improves performance.




15. Click **OK**.
16. Click **OK** on the ODBC Data Source Administrator.

    For details on the TimesTen attributes, refer to Chapter 1, "Data Store Attributes in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

## Defining a DSN on UNIX

UNIX

On UNIX, you must create a configuration file in your home directory prior to running TimesTen. For details on the UNIX configuration files, see Chapter 10, "UNIX Configuration Files." A sample file containing definitions for the DSNs required by TimesTen is provided in:

`/var/timesten/`*`TTinstance`*`/sys.odbc.ini`.

If you have an `.odbc.ini` file in your home directory, merge the contents of the `sys.odbc.ini` file into your `.odbc.ini` file. If a DSN is defined in both your user `.odbc.ini` file and the system `.odbc.ini` file, TimesTen uses the definition in your user `.odbc.ini` file.

After the `.odbc.ini` file has been created in your home directory, create the entry:

```
[ODBC Data Sources]
ref_customers=TimesTen 6.0 Driver

[ref_customers]
DataStore=/tmp/ref_customers
DurableCommits=0
PermSize=16
```

```
LogFileSize=8
```

Now that you have defined a DSN, the next step is to connect to the data store. For further information on creating a data store, refer to the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

# Lesson 2: Connecting to a data store

When the DSN was defined in Lesson 1, the **AutoCreate** attribute was set to allow for creation of the data store at connection. You are now ready to connect to the data store. You will be using the TimesTen utility **ttIsql**. The **ttIsql** utility allows you to run SQL interactively from the command line.


## Using ttIsql on Windows

Perform the following steps to connect to the data store:

1. Choose **Start > Run.**
2. Type **cmd** in the Run field. The Command Prompt dialog appears.
3. At the prompt, type **ttIsql**. This command returns the message:

```
ttIsql (c) 1996-2005, Oracle. All rights reserved. Type ? or help,
type "exit" to quit ttIsql. All commands must end with a semicolon
character.

Command>
```

4. At the **ttIsql** command prompt, type
```
connect "DSN=ref_customers";
```
5. Press **Enter**. The message `(Default setting AutoCommit=1)` returns.

**Note:** If **ttIsql** is not a recognized command in the Command Prompt window, run the `install_dir\bin\ttVars.bat` file to add it to your System Path.


## Using ttIsql on UNIX

Perform the following steps to connect to a data store using UNIX:

1. Type **ttIsql** at the command prompt.
2. The following message is returned:

```
ttIsql (c) 1996-2005, Oracle. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.
All commands must end with a semi-colon character.
```

3. At the **ttIsql** prompt, type:

```
connect "DSN=ref_customers";
```

4. Press **Enter**. The following message is returned:

```
(Default setting AutoCommit=1)
```

Now that the connection has been made the next step is to create the tables that you are going to populate. To obtain help with **ttIsql,** type `?`, or `help`. For further information on **ttIsql**, refer to the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

## Lesson 3: Creating tables

Creating tables in TimesTen requires some prior knowledge of SQL and planning. While this tutorial walks you through creating the tables in this lesson, it is not a substitute for understanding database design principles.

You will build three tables: the first contains customer information, the second contains reference product information, and the third tracks customer orders. Later you will populate the tables with data from the `install_dir/demo` directory.

First you need a table called *customer* with the following columns:

| Column Name | Data type | Nullable? | Key |
|---|---|---|---|
| cust_number | Integer | not null | primary key |
| first_name | Char (12) | not null | |
| last_name | Char (12) | not null | |
| address | Varchar (100) | not null | |

To create the customer information table, execute the following SQL statement at the **ttIsql** prompt:

```
CREATE TABLE customer
(cust_number integer not null primary key,
first_name char (12) not null,
last_name char (12) not null,
address varchar (100) not null);
```

To view what you have just created, execute the `describe` command that is a part of **ttIsql**. The `describe` command returns the structure of each table, including the table name, the column names and data types.

To view the column definitions for the table you created, execute the following **ttIsql** statement:

```
describe customer;
```

The `describe` command returns the following definitions:

```
Table owner.CUSTOMER
Columns:
       *CUST_NUMBER           INTEGER NOT NULL
        FIRST_NAME            CHAR(12) NOT NULL
        LAST_NAME             CHAR(12) NOT NULL
        ADDRESS               VARCHAR(100) NOT NULL
1 table found
(primary key columns are indicated with *)
```

After you have built the *Customer* table, create the two other tables needed to perform queries in later lessons. Use the same syntax as above to create the columns of the new tables.

Create the following table called `ref_products`:

| Column | Data type | Null? | Keys |
|---|---|---|---|
| prod_number | char (10) | not null | primary key |
| prod_name | varchar (100) | not null | |
| price | decimal (6,2) | | |

Create the following table called *Orders*:

| Column | Data type | Null? | Keys |
|---|---|---|---|
| order_number | integer | not null | |
| cust_number | integer | not null | |
| prod_number | char (10) | not null | |
| order_date | date | not null | |

To be sure the columns you have created are correct, execute the `describe` command as you did above. The `describe` command should return the following definition for the *ref_products* table:

```
Table owner.REF_PRODUCTS
Columns:
      *PROD_NUMBER   CHAR(10) NOT NULL
       PROD_NAME     VARCHAR(100) NOT NULL
       PRICE         DECIMAL(6,2) NOT NULL
1 table found
```

and the following definition for the *Orders* table:

```
Table owner.ORDERS
Columns:
```

```
      ORDER_NUMBER    INTEGER NOT NULL
      CUST_NUMBER     INTEGER NOT NULL
      PROD_NUMBER     CHAR(10) NOT NULL
      ORDER_DATE      DATE NOT NULL
1 table found
```

To see the full scheme of this data store, use the **ttSchema** utility from a **ttIsql** session. Enter the following at the**ttIsql** prompt:

```
host ttSchema ref_customers
```

Type `exit` at the **ttIsql** prompt. TimesTen "checkpoints" the data store, saving the information to disk, making it ready for the next connection.

Now that you have created the tables, you are ready to populate them with data. There are data files located in the `install_dir/demo` which you will use for population. The next lesson will walk you through populating the tables with a TimesTen utility.

## Lesson 4: Populating the data store using ttBulkCp

The **ttBulkCp** utility copies data between TimesTen tables and ASCII files. Files containing data to populate the tables you created in Lesson 3 are located in the demo directory. The titles are `customer.dat`, `orders.dat` and `ref_products.dat`. For easy identification, the data files have the same base name as the table names.

To populate the *Customer* table, perform the following steps:

Execute the following statement at the command prompt:

```
ttBulkCp -i -d warn DSN=ref_customers owner.customer customer.dat
```

**Note:** You must specify the path or directory of the `.dat` file. If the full path of the file contains spaces the path and file name must be contained in double quotes. The owner name is given when a `describe` command is executed from **ttIsql** as in Lesson 3. For a full explanation of the arguments given, refer to the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

The following message appears:

```
customer.dat:
   25 rows inserted
   25 rows total
```

To view the data loaded into the *Customer* table, perform the following steps:

1. Connect to the DSN with **ttIsql** as described in Lesson 2.
2. Execute the following SQL statement at the **ttIsql** prompt:

```
SELECT * FROM customer;
```

The results of the SELECT are as follows:

```
< 3700, Peter     , Burchard    , 882 Osborne Avenue, Boston, MA 02122 >
< 1121, Saul      , Mendoza     , 721 Stardust Street, Mountain View, CA 94043 >
< 1278, Mary      , Behr        , 2233 Emerson Road, Vancouver, WA 98663 >
< 1868, Paul      , Tu          , 308 Bowman Court, Palo Alto, CA 94309 >
< 3645, John      , Silva       , 3329 Taffy Lane, Atlanta, GA 30314 >
< 1935, Sandra    , Lao         , 115 Spangler Avenue, San Jose, CA 95112 >
< 1002, Marco     , Mueller     , 40 East 5th Avenue, New York, NY 10009 >
< 2364, Karen     , Johnson     , 3971 Hill Road, Chicago, IL 60608 >
< 2655, Linda     , Garcia      , 7599 Clark Road, Denver, CO 80210 >
< 1077, Gautam    , Mudunuri    , 16 Welsley Avenue, Fremont, CA 94555 >
< 3864, Ruth      , Silver      , 88 West 65th Street, New York, NY 10009 >
< 1010, Fatima    , Borba       , 6868 Bascom Avenue, San Jose, CA 95128 >
< 2300, Pavel     , Popov       , 233 Loredo Street, Dallas, TX 75210 >
< 1001, Steven    , McPhee      , 72 Vine Street, San Jose, CA 95125 >
< 3525, Anthony   , Bianchi     , 122 Fuller Avenue, Patchogue, NY 11772 >
< 2826, Mary      , Anderson    , 6363 Bjorn Road, Minneapolis, MN 55417 >
< 2435, Juanita   , Dawes       , 733 Valdosta Avenue, Baton Rouge, LA 70816 >
< 1224, Abdul     , Aziz        , 6793 Bird Avenue, San Jose, CA 95126 >
< 3611, Katherine , McKenzie    , 54 East 21st Avenue, New York, NY 10009 >
< 1900, Patricia  , Levesque    , 658 Aristotle Road, Palo Alto, CA 94305 >
< 3290, Paula     , Rossi       , 21 West 54th Street, New York, NY 10009 >
< 1665, David     , Singh       , 4001 West Hedding, San Jose, CA 95216 >
< 3098, Cynthia   , Stewart     , 333 East Palm Street, Miami, FL 33150 >
< 1133, Kerri     , Haas        , 68 East San Fernando, San Jose, CA 95113 >
< 2555, Bo        , Smith       , 124 North 1st Street, Dallas, TX 75210 >
25 rows found.
```

With the same syntax as above, use the **ttBulkCp** utility to populate the *ref_products* and *Orders* tables with the corresponding `.dat` files located in the demo directory.

Execute a **SELECT** statement as above. The results of the **SELECT** for the *Orders* table are as follows:

```
Command> select * from orders;
< 6853036, 3700, 0028616731, 1996-04-05 >
< 6853041, 3700, 0198612710, 1997-01-12 >
< 6853169, 1121, 0003750299, 1997-08-01 >
< 6853174, 1121, 0789428741, 1998-02-02 >
< 6853179, 1121, 0198612583, 1998-10-25 >
< 6853302, 1278, 0198612605, 1998-08-22 >
< 6853435, 1868, 0198613202, 1999-04-15 >
< 6853568, 3645, 0028616731, 1998-06-28 >
< 6853701, 1935, 0395671612, 1998-02-12 >
< 6853834, 1002, 0395448956, 1998-05-07 >
< 6853967, 2364, 0877797099, 1997-08-30 >
< 6853967, 2364, 0789435578, 1997-08-30 >
```

```
< 6853967, 2364, 0877799113, 1997-08-30 >
< 6854100, 2655, 0440218616, 1996-07-22 >
< 6854105, 2655, 0877797099, 1997-08-30 >
< 6854233, 1077, 0789435578, 1997-12-01 >
< 6854366, 3864, 0877799113, 1998-09-20 >
< 6854499, 1010, 0028616731, 1998-06-06 >
< 6854632, 2300, 0877799113, 1999-10-15 >
< 6854765, 1001, 0198612605, 1999-11-25 >
< 6854765, 1001, 0789435578, 1999-11-25 >
< 6854765, 1001, 0877799113, 1999-11-25 >
< 6854770, 1001, 0198612710, 1999-12-07 >
< 6854898, 3525, 0877799113, 1998-08-08 >
< 6855031, 2826, 0028616731, 1996-03-01 >
< 6855036, 2826, 0877799113, 1996-03-08 >
< 6855164, 2435, 0385312547, 1997-02-05 >
< 6855297, 1224, 0877799113, 1996-01-03 >
< 6855297, 1224, 0877797099, 1996-01-03 >
< 6855297, 1224, 0789435578, 1996-01-03 >
< 6855297, 1224, 0198612605, 1996-01-03 >
< 6855430, 3611, 0877797099, 1997-06-24 >
< 6855435, 3611, 0877799113, 1997-12-05 >
< 6855440, 3611, 0198612710, 1998-03-13 >
< 6855445, 3611, 0198612583, 1998-11-11 >
< 6855563, 1900, 0194314219, 1998-05-07 >
< 6855696, 3290, 0385312547, 1998-08-22 >
< 6855701, 3290, 0028616731, 1999-03-01 >
< 6855829, 1665, 0440218616, 1998-08-30 >
< 6855962, 3098, 0877799113, 1997-04-07 >
< 6855967, 3098, 0789435578, 1998-04-14 >
< 6856095, 1133, 0877799113, 1996-05-02 >
< 6856228, 2555, 0194314219, 1999-06-17 >
43 rows found.
```

Execute a SELECT statement as above for the third table. The results of the SELECT for the *ref_products* table are as follows:

```
Command> select * from ref_products;
< 0194314219, Oxford Advanced Learner's Dictionary of Current English, 26.95 >
< 0877799113, The Merriam Webster Dictionary, 5.99 >
< 0789435578, Dk Illustrated Oxford Dictionary, 50.00 >
< 0877797099, Merriam Webster's Collegiate Dictionary (10th Edition), 24.95 >
< 0395671612, The American Heritage College Dictionary, 24.00 >
< 0198613202, The Concise Oxford Dictionary of Current English, 29.95 >
< 0385312547, The American Heritage Dictionary, 12.95 >
< 0003750299, Collins Cobuild English Dictionary, 55.00 >
< 0395448956, The American Heritage Dictionary of the English Language, 50.00 >
< 0198612710, The New Shorter Oxford English Dictionary (2 Vol.), 135.00 >
< 0198612583, The Compact Oxford English Dictionary, 375.00 >
< 0198612605, Oxford English Dictionary: CD-Rom for Windows (20 Vol./1 CD), 395.
```

```
00 >
< 0789428741, Ultimate Visual Dictionary, 16.95 >
< 0028616731, Webster's New World College Dictionary, 23.95 >
< 0440218616, The American Heritage Dictionary (Paperback), 6.50 >
15 rows found.
```

After the tables have been populated, you can work with the data store as you would with any other database. In the next lesson you will work with the data store with various utilities. For further information on the **ttBulkCp** utility, refer to the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

# Lesson 5: Working with a data store

## Lesson 5A: Inserts

You are now ready to work with the data store you created. Just as with any database, you can execute SQL statements to insert rows into tables. For example, suppose new customers need to be added to the data store. The first new customer to be added is Josephine Rogers. You use INSERT statements and VALUES clauses to add her information to the customer table.

The INSERT statement allows you to add rows to the data store. Along with the INSERT statement, use a VALUES clause to populate the columns with the desired information.

To insert a row containing new customer information into your data store, execute the following SQL statement at the **ttIsql** prompt:

```
INSERT INTO customer
VALUES (1365,'Josephine','Rogers','2100 Augustine Drive, Santa
Clara, CA 95054');
```

You must type the data values in the same order as the column names in the original CREATE TABLE statement in Lesson 3 (the customer number first, first name second, etc.). The string values must use single quotes.

To view the new row inserted into the *Customer* table, execute the following SQL statement:

```
SELECT * FROM customer WHERE cust_number=1365;
```

Use a separate INSERT statement for each row when you add the following customers:

| cust_ number | first_ name | last_ name | address |
|---|---|---|---|
| 3750 | Michelle | Ioni | 12 Kipling Street, Denver, CO 80215 |
| 3836 | Dante | Cremona | 202 Tampa Way, Fremont, CA 94539 |
| 3945 | Mark | Edwards | 1 Market Street, San Francisco, CA 94126 |

To view the rows inserted in the data store, execute the following SELECT statement:

```
SELECT * FROM customer WHERE cust_number>=3750;
```

The SELECT statement returns the following results:

```
< 3864,  Ruth,     Silver,  88 West 65th Street,  New York,      NY 10009 >
< 3750,  Michelle, Ioni,    12 Kipling Street,    Denver,        CO 80215 >
< 3836,  Dante,    Cremona, 202 Tampa Way,        Fremont,       CA 94539 >
< 3945,  Mark,     Edwards, 1 Market Street,      San Francisco, CA 94126 >
```

## Lesson 5B: Selects

When you populated the tables in Lesson 4, you noticed that for the `orders` table some of the customer numbers and order numbers were repeated. You can create a SELECT statement to give a sum total of the repeated order numbers. To make the report readable, you can use clauses to sort the results.

The aggregate syntax SUM, which gives the total of the values in the numeric expression, is used to add the amount of those repeated order numbers. The WHERE clause uses JOINS to link information from the three tables. And, as much as WHERE qualifies individual rows, the GROUP BY clause groups common rows together. Further, the ORDER BY clause makes the results more readable. It allows you to sort by any set of columns in the SELECT list. In this example, 4 is order date, 1 is last name, and 2 is first name.

To find the total of the customers' orders by date, execute the following SELECT statement:

```
SELECT last_name, first_name, SUM (price),order_date
FROM customer c, ref_products r, orders o
WHERE c.cust_number=o.cust_number and r.prod_number=o.prod_number
GROUP BY last_name, first_name, order_date
ORDER BY 4,1,2;
```

The results of the query are as follows:

```
< Aziz      , Abdul    , 475.94, 1996-01-03 >
< Anderson  , Mary     , 23.95,  1996-03-01 >
< Anderson  , Mary     , 5.99,   1996-03-08 >
< Burchard  , Peter    , 23.95,  1996-04-05 >
< Haas      , Kerri    , 5.99,   1996-05-02 >
< Garcia    , Linda    , 6.50,   1996-07-22 >
< Burchard  , Peter    , 135.00, 1997-01-12 >
< Dawes     , Juanita  , 12.95,  1997-02-05 >
< Stewart   , Cynthia  , 5.99,   1997-04-07 >
< McKenzie  , Katherine, 24.95,  1997-06-24 >
< Mendoza   , Saul     , 55.00,  1997-08-01 >
< Garcia    , Linda    , 24.95,  1997-08-30 >
< Johnson   , Karen    , 80.94,  1997-08-30 >
< Mudunuri  , Gautam   , 50.00,  1997-12-01 >
< McKenzie  , Katherine , 5.99,   1997-12-05 >
< Mendoza   , Saul     , 16.95,  1998-02-02 >
< Lao       , Sandra   , 24.00,  1998-02-12 >
< McKenzie  , Katherine , 135.00, 1998-03-13 >
< Stewart   , Cynthia  , 50.00,  1998-04-14 >
< Levesque  , Patricia , 26.95,  1998-05-07 >
< Mueller   , Marco    , 50.00,  1998-05-07 >
< Borba     , Fatima   , 23.95,  1998-06-06 >
< Silva     , John     , 23.95,  1998-06-28 >
< Bianchi   , Anthony  , 5.99,   1998-08-08 >
< Behr      , Mary     , 395.00, 1998-08-22 >
< Rossi     , Paula    , 12.95,  1998-08-22 >
< Singh     , David    , 6.50,   1998-08-30 >
< Silver    , Ruth     , 5.99,   1998-09-20 >
< Mendoza   , Saul     , 375.00, 1998-10-25 >
< McKenzie  , Katherine , 375.00, 1998-11-11 >
< Rossi     , Paula    , 23.95,  1999-03-01 >
< Tu        , Paul     , 29.95,  1999-04-15 >
< Smith     , Bo       , 26.95,  1999-06-17 >
< Popov     , Pavel    , 5.99,   1999-10-15 >
< McPhee    , Steven   , 450.99, 1999-11-25 >
< McPhee    , Steven   , 135.00, 1999-12-07 >
```

## Lesson 5C: Updates

While INSERT adds new rows to a table, UPDATE changes existing rows. Use UPDATE to change values in single rows, groups of rows, or all the rows in a table.

The UPDATE specifies the row or rows you want to change. For example, we want to discount those reference products costing $50.00 or more by 10%. To begin, perform a SELECT to view the products costing $50.00 or more.

At the command prompt, type the following:

```
SELECT * FROM ref_products
WHERE price >= 50.00;
```

The following list returns:

```
< 0789435578, Dk Illustrated Oxford Dictionary,                          50.00 >
< 0003750299, Collins Cobuild English Dictionary,                        55.00 >
< 0395448956, The American Heritage Dictionary of the English Language,  50.00 >
< 0198612710, The New Shorter Oxford English Dictionary (2 Vol.),       135.00 >
< 0198612583, The Compact Oxford English Dictionary,                    375.00 >
< 0198612605, Oxford English Dictionary: CD-Rom for Windows(20Vol./1CD),395.00 >
```

The UPDATE is followed by a SET clause which specifies the column and changed values. To discount the price of the six products above by 10%, multiply the price by a value of.90.

At the command prompt, type the following:

```
UPDATE ref_products
SET price = price * .90
WHERE price >= 50.00;
```

The following message returns:

```
6 rows updated
```

To view the new prices, execute the following at the command prompt:

```
SELECT * FROM ref_products
WHERE price >= 45.00;
```

The following list returns:

```
< 0789435578, Dk Illustrated Oxford Dictionary, 45.00 >
< 0003750299, Collins Cobuild English Dictionary, 49.50 >
< 0395448956, The American Heritage Dictionary of the English Language, 45.00 >
< 0198612710, The New Shorter Oxford English Dictionary (2 Vol.), 121.50 >
< 0198612583, The Compact Oxford English Dictionary, 337.50 >
< 0198612605, Oxford English Dictionary: CD-Rom for Windows (20 Vol./1 CD), 355.
50 >
```

## Lesson 5D: Deletes

It is just as important to be able to remove rows as it is to be able to add or change them. Like INSERT and UPDATE, CREATE VIEW works for single operations as well as multiple-row operations.

For example, suppose our customer Steven McPhee has contacted us and wants to cancel an order placed in November 1999. Mr. McPhee's customer number is 1001.

To view all of Steven Mcphee's orders, type the following at the command prompt:

```
SELECT * FROM orders
WHERE cust_number = 1001;
```

The following list returns:

```
< 6854765, 1001, 0198612605, 1999-11-25 >
< 6854765, 1001, 0789435578, 1999-11-25 >
< 6854765, 1001, 0877799113, 1999-11-25 >
< 6854770, 1001, 0198612710, 1999-12-07 >
```

There are two orders for Mr. McPhee. Since order number 6854765 is the order placed in November 1999, this is the one which will be deleted from the `orders` table.

To delete order number 6854765, type the following at the command prompt:

```
DELETE from orders
WHERE order_number = 6854765;
```

The following message returns:

```
(3 rows affected)
```

To view Steven McPhee's current orders, type the following at the command prompt:

```
SELECT * FROM orders
WHERE cust_number = 1001;
```

The following order returns:

```
< 6854770, 1001, 0198612710, 1999-12-07 >
```

The SELECT statement shows that Steven McPhee now has only one order on record.

Now that an individual order has been deleted, you are ready to update the data store, and delete those records which were placed prior to 1997. First you need to find those orders from the following SELECT statement:

```
SELECT * FROM orders
WHERE order_date <= '1996-12-31';
```

The following orders return:

```
< 6853036, 3700, 0028616731, 1996-04-05 >
< 6854100, 2655, 0440218616, 1996-07-22 >
< 6855031, 2826, 0028616731, 1996-03-01 >
< 6855036, 2826, 0877799113, 1996-03-08 >
< 6855297, 1224, 0877799113, 1996-01-03 >
< 6855297, 1224, 0877797099, 1996-01-03 >
< 6855297, 1224, 0789435578, 1996-01-03 >
< 6855297, 1224, 0198612605, 1996-01-03 >
< 6856095, 1133, 0877799113, 1996-05-02 >
```

As you can see, there are nine orders placed prior to 1997. These are the orders which you need to delete.

The CREATE VIEW statement uses the same WHERE clause you used in the query above. To delete those orders placed before January 1, 1997, execute the following at the command prompt:

```
DELETE FROM orders
WHERE order_date <= '1996-12-31';
```

The following message returns:

```
9 rows deleted
```

The nine orders have been deleted from the data store.

Now that the data store has been updated, you can perform a backup. The backup takes a snapshot of the data store for a possible restore. For further information on working with a data store, please refer to the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*, and the *TimesTen C or Java Developer's Guide*.

## Lesson 6: Data store backup

To take a snapshot of a data store in order to later use that data store in the exact same state, use the **ttBackup** utility (the **ttRestore** utility is used for restoring the data store and is covered in Lesson 8).

To backup the data store, perform the following:

1. Type `exit` at the **ttIsql** prompt.
2. Execute the following statement at the prompt:

   Unix: `ttBackup -dir /tmp "DSN=ref_customers"`

   Windows: `ttBackup -dir \temp "DSN=ref_customers"`

The following message appears:

```
Backup started
Backup complete
```

This message informs you that the backup is complete and was successful.

For further information on using the **ttBackup** utility, refer to the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

You have now completed a backup and are ready to destroy a data store in the next lesson.

# Lesson 7: Destroying a data store

If a data store becomes corrupted, or needs to be deleted, it can be destroyed. The **ttDestroy** utility destroys a data store including all checkpoint files, transaction logs and daemon catalog entries (though not the DSNs).

Before using the **ttDestroy** utility check the name of all data stores within TimesTen. Use the **ttStatus** utility to display this information. The **ttStatus** utility displays information that describes the current state of TimesTen. This information includes the state of the TimesTen daemon process and all subdaemon processes; the names of all existing TimesTen data stores; the number of users currently connected to each TimesTen data store; and various other information.

## Using ttStatus

Execute the following at the command prompt:

```
ttStatus
```

The following statement returns:

```
TimesTen status report as of Thu Apr 22 12:44:03 2005

Daemon pid 2200 port 15100 instance tt51
TimesTen server pid 4092 started on port 15102
------------------------------------------------

Data store c:\temp\ref_customers
There are no connections to the data store
Replication policy  : Manual
Oracle agent policy : Manual
------------------------------------------------

End of report
```

You will use this utility to verify that the data store has been destroyed.

### Using ttDestroy

To destroy the data store, execute the following at the command prompt:

UNIX: `ttDestroy /tmp/ref_customers`

Windows: `ttDestroy \temp\ref_customers`

Execute the **ttStatus** utility again.

The following statement returns:

```
TimesTen status report as of Thu Apr 22 12:48:03 2005
Daemon pid 2200 port 15100 instance tt51
TimesTen server pid 4092 started on port 15102
----------------------------------------------
End of report
```

Once the data store has been destroyed, you can restore it from the backup which was made in Lesson 6. For further information on the **ttDestroy** utility, refer to the *TimesTen C Developer's Guide* or the *TimesTen Java Developer's Guide*.

## Lesson 8: Restoring a data store

To restore your data store, you will use the **ttRestore** utility. The **ttRestore** utility restores a data store that has been backed up using the **ttBackup** utility. If the data store has not been destroyed, **ttRestore** will not overwrite any current data store.

To restore the `ref_customers` data store, execute the following at the command prompt:

UNIX: `ttRestore -dir /tmp "DSN=ref_customers"`

Windows: `ttRestore -dir C:\temp "DSN=ref_customers"`

---

**Note:** The required information when using **ttRestore** is the directory where the backup files are stored and the ODBC connection string for the data store. Refer to "Lesson 6: Data store backup" on page 24 for the directory where the files are stored.

---

The following message appears:

```
Restore started...
```

When the restore is complete, the following message appears:

```
Restore complete
```

Perform the following steps to verify the restore:

1. At the command prompt, type **ttIsql**.
2. At the **ttIsql** command prompt, connect to the data store by typing:

```
connect "DSN=ref_customers";
```

3. Execute the following SQL statement at the **ttIsql** prompt:

```
SELECT * FROM customer;
```

The results of the SELECT are as follows:

```
< 3700, Peter,     Burchard,  882 Osborne Avenue, Boston, MA 02122 >
< 1121, Saul,      Mendoza,   721 Stardust Street, Mountain View, CA 94043 >
< 1278, Mary,      Behr,      2233 Emerson Road, Vancouver, WA 98663 >
< 1868, Paul,      Tu,        308 Bowman Court, Palo Alto, CA 94309 >
< 3645, John,      Silva,     3329 Taffy Lane, Atlanta, GA 30314 >
< 1935, Sandra,    Lao,       115 Spangler Avenue, San Jose, CA 95112 >
< 1002, Marco,     Mueller,   40 East 5th Avenue, New York, NY 10009 >
< 2364, Karen,     Johnson,   3971 Hill Road, Chicago, IL 60608 >
< 2655, Linda,     Garcia,    7599 Clark Road, Denver, CO 80210 >
< 1077, Gautam,    Mudunuri,  16 Welsley Avenue, Fremont, CA 94555 >
< 3864, Ruth,      Silver,    88 West 65th Street, New York, NY 10009 >
< 1010, Fatima,    Borba,     6868 Bascom Avenue, San Jose, CA 95128 >
< 2300, Pavel,     Popov,     233 Loredo Street, Dallas, TX 75210 >
< 1001, Steven,    McPhee,    72 Vine Street, San Jose, CA 95125 >
< 3525, Anthony,   Bianchi,   122 Fuller Avenue, Patchogue, NY 11772 >
< 2826, Mary,      Anderson,  6363 Bjorn Road, Minneapolis, MN 55417 >
< 2435, Juanita,   Dawes,     733 Valdosta Avenue, Baton Rouge, LA 70816 >
< 1224, Abdul,     Aziz,      6793 Bird Avenue, San Jose, CA 95126 >
< 3611, Katherine, McKenzie,  54 East 21st Avenue, New York, NY 10009 >
< 1900, Patricia,  Levesque,  658 Aristotle Road, Palo Alto, CA 94305 >
< 3290, Paula,     Rossi,     21 West 54th Street, New York, NY 10009 >
< 1665, David,     Singh,     4001 West Hedding, San Jose, CA 95216 >
< 3098, Cynthia,   Stewart,   333 East Palm Street, Miami, FL 33150 >
< 1133, Kerri,     Haas,      68 East San Fernando, San Jose, CA 95113 >
< 2555, Bo,        Smith,     124 North 1st Street, Dallas, TX 75210 >
< 1365, Josephine, Rogers,    2100 Augustine Drive, Santa Clara, CA 95054 >
< 3750, Michelle,  Ioni,      12 Kipling Street, Denver, CO 80215 >
< 3836, Dante,     Cremona,   202 Tampa Way, Fremont, CA 94539  >
< 3945, Mark,      Edwards,   1 Market Street, San Francisco, CA 94126  >
```

You have now completed the QuickStart tutorial and are ready to move on to planning and implementing your own data store. Your TimesTen documentation set is a complete reference to utilities and SQL.

## Other considerations

If you are about to put a real-world data store online, you have several critical issues, like indexing and integrity to consider. TimesTen assumes you will consider these issues as you develop your design, and after completing the tutorial. Contact TimesTen Customer Support if you have any further questions.

# 2

# *Creating TimesTen Data Stores*

A TimesTen data store is a collection of tables and indexes that can be accessed and manipulated through SQL. This chapter describes how to set up a TimesTen data store. It begins with an overview of the things you should consider when setting up a data store and then describes each task in detail.

Once you have created a data store, you can:

- Use the **ttIsql** utility to connect to the data store and execute a SQL file or start an interactive SQL session, as described in "Batch mode vs. interactive mode" on page 90.
- Execute an application that uses the data store, as described *Oracle TimesTen In-Memory Database Java Developer's and Reference Guide* and *Oracle TimesTen In-Memory Database C Developer's and Reference Guide*.

The main topics are:

# TimesTen ODBC and JDBC drivers

This section describes some basic concepts that will help you define TimesTen data stores. The main topics are:

- TimesTen ODBC drivers
- TimesTen JDBC driver

C applications interact with TimesTen either by linking directly with a TimesTen ODBC driver, or by linking with an ODBC driver manager. Java applications access the ODBC driver through a JDBC library. Consider the following points:

- An application that uses an ODBC driver manager dynamically loads an ODBC driver at runtime. A single application can use more than one ODBC driver within the same application, even if the drivers are for different RDBMS products. The downside to this flexibility is that the driver manager adds additional runtime overhead.
- An application that links directly with an ODBC driver can use only the driver with which it is linked. This option offers less flexibility but better performance than linking with a driver manager.
- An application that is using an ODBC driver manager cannot use XLA.


On Windows, TimesTen installs the ODBC driver manager if it is not already present on the machine.


On UNIX, ODBC driver managers are available from third-party vendors. ODBC driver managers are not supplied with UNIX systems. This guide refers to the use of a driver manager for Windows systems only.

For more information on how to compile an application that uses the TimesTen data manager, see *Oracle TimesTen In-Memory Database Java Developer's and Reference Guide* and *Oracle TimesTen In-Memory Database C Developer's and Reference Guide*.

## TimesTen ODBC drivers

TimesTen includes two versions of the Data Manager ODBC driver: a *production* version and a *debug* version.

- Use the *production* version of the TimesTen Data Manager driver for most application development and for all deployment.
- Use the *debug* version of the TimesTen Data Manager driver only if you encounter problems with TimesTen itself. This version performs additional internal error checking and is slower than the production version. (On UNIX, the TimesTen debug libraries are compiled with the `-g` option to display additional debug information.)

TimesTen also includes the TimesTen Client ODBC driver for use with client/server applications.


On Windows, the production version of the TimesTen Data Manager is installed by default. To install the debug version, choose **Custom** setup. To install the TimesTen Client driver, choose either **Typical** or **Custom** setup. The following table lists the ODBC drivers for Windows:

| Platform | Version | Name |
|---|---|---|
| Windows | Production | TimesTen Data Manager 6.0 Driver. |
| Windows | Debug | TimesTen Data Manager 6.0 Debug Driver. |
| Windows | Client | TimesTen Client 6.0 Driver |


On UNIX, depending on the options selected at install time, TimesTen installs the Client driver and/or both the production version and the debug version of the TimesTen Data Manager ODBC driver. The following table lists the TimesTen ODBC drivers for UNIX platforms, assuming TimesTen is installed in the default directory by user `root`.

| Platform | Version | Location and Name |
|---|---|---|
| HP-UX | Production | `/opt/TimesTen/`*`TTinstance`*`/lib/libtten.sl`<br>TimesTen Data Manager 6.0 Driver. |
| HP-UX | Debug | `/opt/TimesTen/`*`TTinstance`*`/lib/libttenD.sl`<br>TimesTen Data Manager 6.0 Debug Driver. |
| HP-UX | Client | `/opt/TimesTen/`*`TTinstance`*`/lib/libttclient.sl`<br>TimesTen Client 6.0 Driver. |
| Solaris<br>Linux<br>Tru64 | Production | `/opt/TimesTen/`*`TTinstance`*`/lib/libtten.so`<br>TimesTen Data Manager 6.0 Driver. |
| Solaris<br>Linux<br>Tru64 | Debug | `/opt/TimesTen/`*`TTinstance`*`/lib/libttenD.so`<br>TimesTen Data Manager 6.0 Debug Driver. |
| Solaris<br>Linux<br>Tru64 | Client | `/opt/TimesTen/`*`TTinstance`*`/lib/libttclient.so`<br>TimesTen Client 6.0 Driver. |
| AIX | Production | `/usr/lpp/TimesTen/`*`TTinstance`*`/lib/libtten.a`<br>TimesTen Data Manager 6.0 Driver. |

| Platform | Version | Location and Name |
|---|---|---|
| AIX | Debug | `/usr/lpp/TimesTen/`*`TTinstance`*`/lib/libttenD.a`<br>TimesTen Data Manager 6.0 Debug Driver. |
| AIX | Client | `/usr/lpp/TimesTen/`*`TTinstance`*`/lib/libttclient.a`<br>TimesTen Client 6.0 Driver. |

## TimesTen JDBC driver

The TimesTen JDBC driver uses the ODBC driver to access the TimesTen data stores. For each JDBC method, the driver executes a set of ODBC functions to perform the appropriate operation. Since the JDBC driver depends on ODBC for all data store operations, the first step in using JDBC is to define a TimesTen data store and the ODBC driver that will access it on behalf of JDBC.

JDBC is installed with the TimesTen Data Manager. JDBC allows Java applications to issue SQL statements to TimesTen and process the results. JDBC is the primary interface for data access in the Java programming language.

The JDBC API is implemented using a driver manager that can support multiple drivers connecting to different databases. The TimesTen JDBC driver is implemented using native methods to bridge to the TimesTen native API.

For a list of the functions supported by TimesTen, see the *Oracle TimesTen In-Memory Database Java Developer's and Reference Guide*.

### JDBC driver manager

WINDOWS

The JDBC driver manager (DriverManager class) keeps track of all the JDBC drivers that have been loaded and are available to the Java Application. The application may load several drivers and access each driver independently. For example, both the TimesTen JDBC Client/Server driver and the TimesTen JDBC direct driver can be loaded onto a machine. Then Java applications can access data stores either on the local machine or a remote machine.

# Data source names

TimesTen data stores are accessed through Data Source Names (DSNs). A DSN is a character-string name that identifies a TimesTen data store and a collection of connection attributes that are to be used when connecting to the data store. On Windows, the DSN also specifies the ODBC driver to be used to access the data store.

Each DSN uniquely identifies a data store. However, a data store can be referenced by multiple DSNs. Each of these DSNs can specify a different set of connection attributes. This allows you to give convenient names to different connection configurations for a single data store.

---

**Note:** According to the ODBC standard, when an attribute occurs multiple times in a connection string, the first value specified is used, not the last value.

---

A DSN has the following characteristics:

- Its maximum length is 32 characters.
- It is composed of ASCII characters except for the following: []{},;?*=!@\
- It cannot contain spaces.

The rest of this section includes the following topics:



## User and system DSNs

DSNs are resolved using a two-tiered naming system, consisting of user DSNs and system DSNs:

- A **user DSN** can be used only by the user who created the DSN. On Windows, user DSNs are defined from the **User DSN** tab of the ODBC Data Source Administrator. On UNIX, user DSNs are defined in the file `$HOME/.odbc.ini` or in a file named by the ODBCINI environment variable. This file is referred to as the "user ODBC.INI file." Although a user DSN is private to the user who created it, it is only the DSN (the character-string name and its attributes) that is private. The underlying data store can be referenced by other users' user DSNs or by system DSNs.
- A **system DSN** can be used by any user on the machine on which the system DSN is defined. On Windows, system DSNs are defined from the **System DSN** tab of the ODBC Data Source Administrator. On UNIX, system DSNs are defined in the file `/var/TimesTen/sys.odbc.ini` if TimesTen is installed

by user `root`, or *`install_dir`*`/info/sys.odbc` if TimesTen is installed by a non-root user. This file is referred to as the "system ODBC.INI file."

When looking for a specific DSN, TimesTen first looks for a user DSN with the specified name. If no matching user DSN is found, TimesTen looks for a system DSN with the specified name. If a user DSN and a system DSN with the same name exist, TimesTen uses the user DSN. On UNIX, if there are multiple DSNs with the same name in the same ODBC.INI file, TimesTen uses the first one in the file.

## Data Manager and Client DSNs

DSNs that use the TimesTen Data Manager (either the production version or the debug version) are called "Data Manager DSNs." DSNs that use the TimesTen Client are called "Client DSNs."

As described in Chapter 4, "How Applications Connect to Data Stores" in the *Oracle TimesTen In-Memory Database Architectural Overview*, Data Manager DSNs define what is referred to as a *direct driver* connection to the data store. A Data Manager DSN refers to a data store using a path name. A data store path name is a path name that specifies the location of the data store, for example: `C:\data\chns\AdminDS` or `/home/chns/AdminDS`. This name is not a file name. The actual files used by the data store have file suffixes, for example: `C:\data\chns\AdminDS.ds0` or `/home/chns/AdminDS.log2`. A Data Manager DSN that refers to a given TimesTen data store must be defined on the same system on which the data store resides. In addition, TimesTen creates *`dsName`*`.resn` files for each data store. These files are used internally by TimesTen for the creation of logs.

---

**Note:** If multiple Data Manager DSNs refer to the same data store, they must all use exactly the same data store path name, even if some other path name identifies the same location. For example, you cannot use a symbolic link to refer to the data store in one DSN and the actual path name in another DSN. On UNIX, you cannot place the data store on an NFS-mounted file system. On Windows, you cannot use a mapped drive letter in the data store path name.

---

A Client DSN refers to a TimesTen data store indirectly by specifying a `hostname, DSN` pair, where the hostname represents a machine on which TimesTen Server Daemon is running and the DSN refers to a system DSN that is defined on that host. We refer to this host as the "server machine" and the DSN as a "Server DSN."

UNIX

On UNIX, all user DSNs (Client DSNs and/or Data Manager DSNs) created by a specific user are defined in the same user ODBC.INI file. Similarly, all system DSNs are defined in the same system ODBC.INI file.

The following table indicates the types of DSN supported by TimesTen, whether to create a user or system DSN and the location of the DSN.

| DSN type | User or System DSN? | Location of DSN |
|---|---|---|
| Data Manager DSN | Can be a user or system DSN | Located on the machine where the data store resides. |
| Client DSN | Can be a user or system DSN | Located on any local or remote machine. |
| Server DSN | Must be a system DSN | Located on the machine where the data store resides. |

The remainder of this chapter describes Data Manager DSNs and the connection attributes that can be defined for them. For more information about Client DSNs and Server DSNs, see Chapter 3, "Working with the TimesTen Client and Server."

## Connection attributes for Data Manager DSNs

There are four types of TimesTen Data Manager attributes:

- ***Data store attributes*** are associated with a data store when it is created and cannot be modified by subsequent connections.
- ***First connection attributes*** are set when a connection is made to an idle data store (a data store with no connections) and persists for that connection and all subsequent connections until the last connection to the data store is closed.
- ***General connection attributes*** are set by each connection and persist for the duration of the connection. Different concurrent connections may use different values.
- ***Cache Connect attributes*** allow you to enter the Oracle Service Identifier for the Oracle instance from which data will be loaded into TimesTen.

---

**Note:** See Chapter 3, "Working with the TimesTen Client and Server" for a description of the connection attributes that can be used with the TimesTen Client ODBC driver.

---

For a complete description of each attribute, see Chapter 1, "Data Store Attributes" in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

WINDOWS

On Windows, you specify data store attributes in the ODBC Data Source Administrator.


On UNIX, you specify data store attributes in the ODBC.INI file. Attributes that do not appear in the ODBC.INI file assume their default value.

## Exclusive and shared connections

When in use, a TimesTen data store resides in virtual memory. Applications can access a data store in either exclusive or shared mode. See **ExclAccess** in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

- In **Exclusive mode**, only one application can connect to the data store at a time, and that application can open only one connection to the data store at a time. The data store resides within the address space of the connected application, and therefore cannot be access by any other process.
- In **Shared mode**, multiple processes and threads can connect to the data store. The data store resides in a region of shared memory, as provided by the underlying operating system.

Applications using Exclusive mode may see better performance than those using Shared mode, as TimesTen avoids certain overheads (fine-grain locking and latching, for example) in Exclusive mode.

## Thread programming with TimesTen

TimesTen supports multi-threaded application access to data stores. When a connection is made to a data store, any thread may issue operations on the connection.

To permit multiple connections to the same data store, multi-threaded applications must set the **ThreadSafe=1** connection attribute.

Typically, a thread issues operations on its own connection and therefore in a separate transaction from all other threads. In environments where threads are created and destroyed rapidly, better performance may be obtained by maintaining a pool of connections. Threads can allocate connections from this pool on demand to avoid the connect and disconnect overhead.

TimesTen allows multiple threads to issue requests on the same connection and therefore the same transaction. These requests are serialized by TimesTen, although the application may require additional serialization of its own.

TimesTen also allows a thread to issue requests against multiple connections, managing activities in several separate and concurrent transactions on the same or different data stores.

# Creating a DSN on Windows

WINDOWS

This section describes how to set up a TimesTen data store on Windows. Before you begin, read the "TimesTen ODBC and JDBC drivers" on page 30 to find out what you'll need to consider as you set up the data store.

For additional examples of setting up a data store, see "DSN examples" on page 42.

This section includes the following topics:



## Specify the ODBC driver

Specify the ODBC driver in the ODBC Data Source Administrator.

---

**Note:** JDBC users need to specify the ODBC driver to be used by the JDBC driver, as described in "TimesTen JDBC driver" on page 32.

---

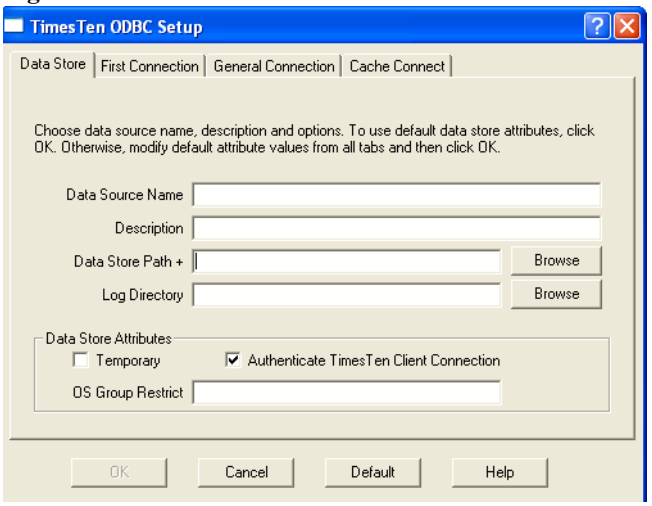
1. On the Windows Desktop, choose **Start > Settings > Control Panel > Administrative Tools > Data Sources (ODBC)**.
   This opens the ODBC Data Source Administrator.
2. Choose **User DSN** if you want to create a user DSN or **System DSN** if you want to create a system DSN. You must have administrator privileges to create a System DSN. For a description of user and system DSNs, see "User and system DSNs" on page 33.
3. Do one of the following:
   - Select an existing data source and click **Configure**.
   - Click **Add,** choose the appropriate TimesTen driver from the list. Click **Finish**. This displays the TimesTen ODBC Setup dialog.

     For a list of TimesTen ODBC drivers, see "TimesTen ODBC drivers" on page 30.

## Specify the DSN

On the Data Store tab of the TimesTen ODBC Setup dialog, specify a data source name and a data store pathname. The Data Store Path Name cannot reference a mapped drive. See Figure 2.1.

**Figure 2.1 Data Store**




For an explanation of DSNs and data store pathnames, see "Data source names" on page 33. The description field is optional.

## Specify the connection attributes

Indicate the desired connection attributes under the **First Connection**, **General Connection** tabs of the TimesTen ODBC Setup dialog. See Figure 2.2 and Figure 2.3.

For a description of the connection attributes, see Chapter 1, "Data Store Attributes" in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

**Figure 2.2 First Connection Attributes**

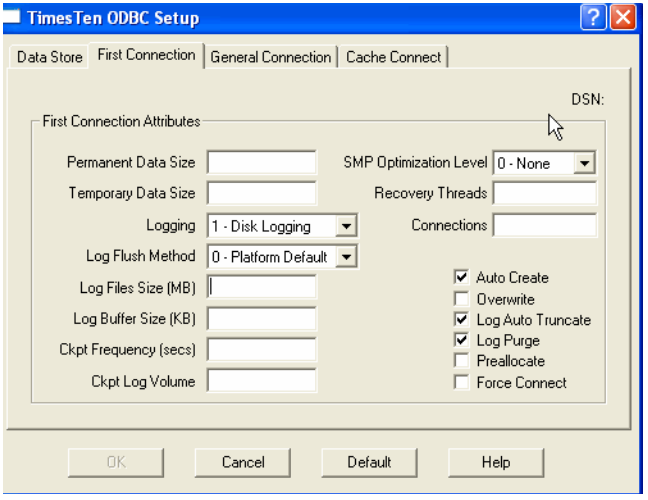
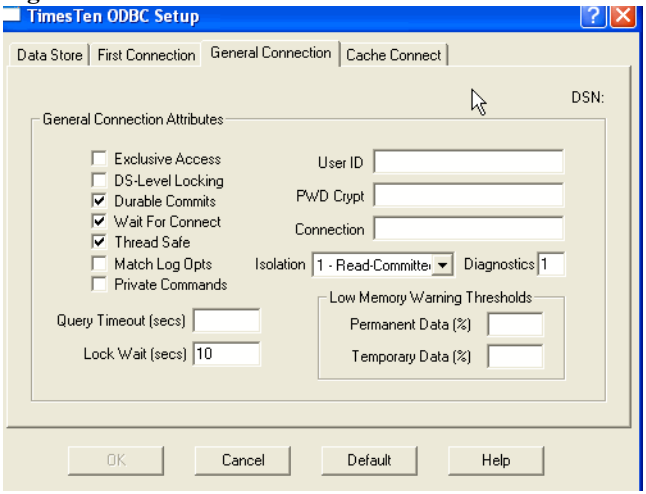


**Figure 2.3 General Connection Attributes**




Click **OK** when you are finished.

# Creating a DSN on UNIX


This section describes how to set up a TimesTen data store on UNIX. Before you begin, be sure to read the "TimesTen ODBC and JDBC drivers" on page 30 to find out what you'll need to consider as you set up the data store.

For examples of defining a data store, see "DSN examples" on page 42.

This section includes the following topics:



## Create a user ODBC.INI file

On UNIX, user DSNs are defined in the file `$HOME/.odbc.ini` or in a file named by the ODBCINI environment variable. This file is referred to as the "user ODBC.INI file." System DSNs are defined in the file `/var/TimesTen/sys.odbc.ini` if TimesTen is installed by user `root`, or *`install_dir`*`/info/sys.odbc` if TimesTen is installed by a non-root user. This file is referred to as the "system ODBC.INI file."

This file is referred to as the "system ODBC.INI file." The syntax for the user ODBC.INI file and the system ODBC.INI file is the same.

The system ODBC.INI file is created when TimesTen is installed on the machine. Users must create their own user ODBC.INI file.

For more information about configuration files, see Chapter 10, "UNIX Configuration Files" in the *Oracle TimesTen In-Memory Database Operations Guide*.

## Specify the DSN

Specify the data source name in the ODBC.INI file. The DSN appears inside square brackets at the top of the DSN definition on a line by itself. For example:

```
[AdminDS]
```

## Specify the ODBC driver

---

**Note:** JDBC users need to specify the ODBC driver to be used by the JDBC driver, as described in "TimesTen JDBC driver" on page 32.

---

To set the TimesTen driver, specify the `DRIVER` parameter in the ODBC.INI file. For example:

```
[AdminDS]
DRIVER=install_dir/lib/libtten.so
```

For a list of TimesTen ODBC drivers, see "TimesTen ODBC drivers" on page 30.

## Specify the data store path name

Specify the data store path name in the ODBC.INI file. For example:

```
DataStore=/users/robin/FixedDs
```

where FixedDs is the prefix for data store files. For more information, see "Data source names" on page 33.

---

**Note:** On UNIX, the data store path name cannot reference an NFS-mounted file system.

---

### Set data store attributes

Specify data store attributes in your ODBC.INI file. Attributes that do not appear in the ODBC.INI file assume their default value. See Chapter 1, "Data Store Attributes" in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*. For examples, see "DSN examples" on page 42.

### Using environment variables in data store path names

You can use environment variables in the specification of the data store path name and log file path name. For example, you can specify `$HOME/AdminDS` for the location of the data store.

Environment variables can be expressed either as `$varname` or `$(varname)` (the parentheses are optional). A backslash character (\) in the data store path name quotes the next character.

---

**Note:** Environment variable expansion uses the environment of the process connecting to the data store. Different processes may have different values for the same environment variables and may therefore expand the data store path name differently. Environment variables can only be used in the user ODBC.INI file. They cannot be specified in the system ODBC.INI file.

---

# DSN examples

This section provides additional examples of how to set up a data store:

- Setting up a temporary data store
- Creating multiple DSNs to a single data store
- Connecting to a data store without using a DSN

For each example, the Windows ODBC Data Source Administrator settings are followed by the corresponding ODBC.INI entries for UNIX.

## Setting up a temporary data store

**Example 2.1** This example illustrates how to set up a temporary data store.

WINDOWS On Windows, you can use the settings in the TimesTen ODBC Setup dialog to set up a temporary data store. See Figure 2.4 and Figure 2.5.

**Figure 2.4 Data Store**

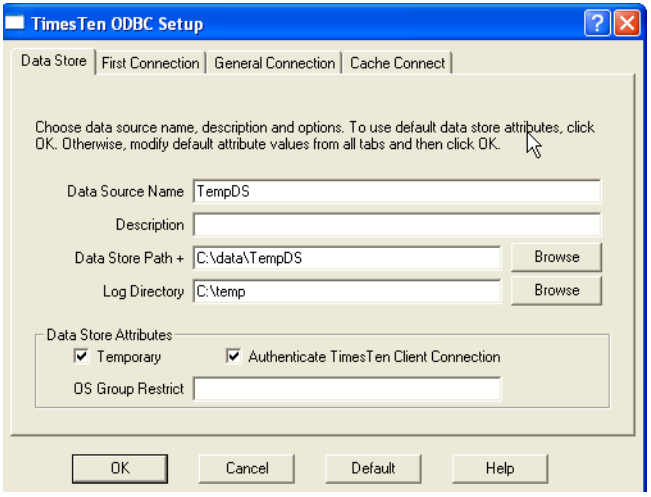
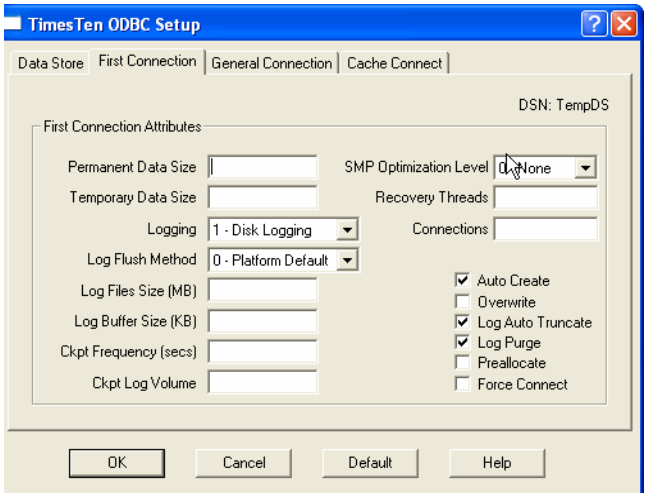


**Figure 2.5 First Connection Attributes**




UNIX

To set up a temporary data store on UNIX, create the following entries in your ODBC.INI file. For a list of drivers for all UNIX platforms, see the table on page 31.

The text in square brackets is the data source name.

```
[TempDs]
Driver=install_dir/lib/libtten.so
DataStore=/users/robin/TempDs
#this is a temporary data store
Temporary=1
#create data store if it is not found
AutoCreate=1
#log data store updates to disk
Logging=1
LogPurge=1
```

**Note:** A temporary data store cannot be backed up.

## Creating multiple DSNs to a single data store

You can create two or more DSNs that refer to the same data store but have different connection attributes. For example, an administrator might want to connect to the data store with different settings than other users.

**Example 2.2** This example creates two DSNs, `AdminDs` and `OperationalDS`. `AdminDs` is used to perform administrative operations on the data store such as updating statistics and creating indexes. The second data source name, `OperationalDs`, is used for normal access to the data store.

WINDOWS

For Windows, use the ODBC Data Source Administrator to define two DSNs as shown in Figure 2.6 and Figure 2.7.

**Figure 2.6 Data Store**

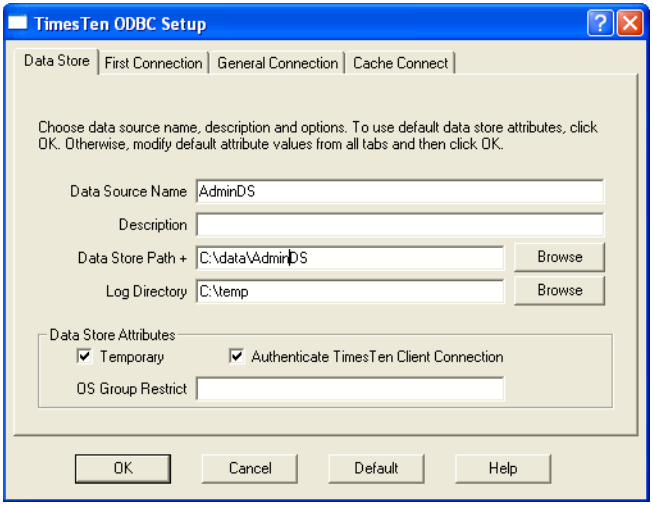
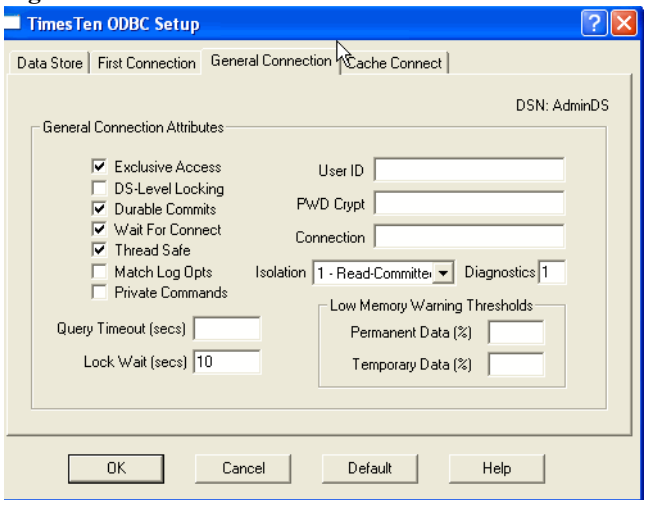



**Figure 2.7 First Connection Attributes**




UNIX

This example shows how to specify a data store with two DSNs on UNIX. It uses the TimesTen Data Manager ODBC driver for Solaris.

The text in square brackets is the data source name.

```
[AdminDs]
Driver=install_dir/lib/libtten.so
Datastore=/users/robin/TTDS
#connect to data store in exclusive mode
ExclAccess=1
```

```
#do not force log to disk on transaction commit
DurableCommits=0

[OperationalDs]
Driver=install_dir/lib/libtten.so
DataStore=/users/robin/TTDS
#create data store if it's not found
AutoCreate=1
#do not wait if cannot connect to data store
WaitForConnect=0
#remove old log files at connect and checkpoint
LogPurge=1
```

## Connecting to a data store without using a DSN

Using the **ttIsql** utility, you can connect to a data store without a predefined Data Source Name by specifying:

- The name or path name of driver using the Driver attribute, and
- The data store path and filename prefix using DataStore attribute

On Microsoft Windows systems, the value of the Driver attribute should be the name of the TimesTen ODBC Driver. For example, TimesTen Data Manager 6.0.

On UNIX systems, the value of the Driver attribute should be the pathname of the TimesTen ODBC Driver shared library file. The file resides in the *install_dir*/lib directory.

**Example 2.3** C:\ ttIsql

ttIsql <c> 1996-2005, Oracle. All rights reserved.

```
Type ? or "help" for help, type "exit" to quit ttIsql.
All commands must end with a semicolon character.

Command> connect "Driver=TimesTen Data Manager5.1;
           DataStore=C:\sales\admin";
```

# Specifying the size of a data store

This section includes the following topics:

- Temporary and permanent memory
- Changing data store size
- Receiving out-of-memory warnings

## Temporary and permanent memory

TimesTen manages data store space using two separate memory partitions within a single contiguous memory space. One partition contains permanent data and the other contains temporary data.

- Permanent data includes the tables and indexes that make up a TimesTen data store. When a data store is loaded into memory, the contents of the permanent data partition are read from files stored on disk. The permanent data partition is written to disk during checkpoint operations.
- Temporary data includes locks, cursors, compiled commands, and other structures needed for command execution and query evaluation. The temporary data partition is created when a data store is loaded into memory and is destroyed when it is unloaded.

The connection attributes that control the size of the data store when it is in memory are PermSize and TempSize. The PermSize attribute specifies the size of the permanent data partition and the TempSize attribute specifies the size of the temporary data partition.

See Chapter 1, "Data Store Attributes" in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide* for further description of these attributes.

## Changing data store size

The sizes of the permanent and temporary data partitions are set when a data store is loaded into memory and cannot be changed while the data store is in memory. To change the size of either partition, you must unload the data store from memory and then reconnect using different values for the PermSize or TempSize attributes. You can use the **ttStatus** utility to find processes connected to the data store and stop them. Once you have made the change in data store size, reload it into memory.

If the data store is configured for replication, you must also stop the replication agent. You must reconfigure the data store sizes for all replicas of the data store. Once you have made the change in data store size, read it into memory and restart the replication agent and Cache Connect if it was connected.

- The permanent data partition can be increased in size, but it cannot be decreased.

- The temporary data partition can be either increased or decreased in size for data stores that do not participate in replication.

To unload a data store from memory, you must close all active connections to the data store and the RAM policy of the data store must be set to manual or inUse. See “Specifying a RAM policy” on page 48 for more information about RAM policy settings.

You must also make sure that you have a shared memory segment that is large enough to hold the data store. In general, the minimum size of this shared memory segment should be:

```
PermSize + TempSize + LogBuffSize +7MB overhead
```

For more details, see "Installation prerequisites" in the TimesTen Installation Guide and the descriptions of the attributes **TempSize** and **PermSize** in the TimesTen Reference Guide.

## Receiving out-of-memory warnings

TimesTen provides two General Connection attributes that determine when a low memory warning should be issued: **PermWarnThreshold** and **SqlQueryTimeout**. Both attributes take a percentage value.

To receive out-of memory warnings, applications must call the built-in procedure **ttWarnOnLowMemory**.

These attributes also set the threshold for SNMP warning. See the chapter, “Diagnostics through SNMP Traps” in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

## Specifying a RAM policy

TimesTen allows you to specify a RAM policy that determines when data stores are loaded and unloaded from main memory. To set the RAM policy, use the **ttAdmin** utility.

For each data store you can have a different RAM policy. The policy options are:

- *In Use.* The data store is loaded into memory when the first connection to the data store is opened, and it remains in memory as long as it has at least one active connection. When the last connection to the data store is closed, the data store is unloaded from memory. This is the default policy.
- *InUse with RamGrace.* The data store is loaded into memory when the first connection to the data store is opened, and it remains in memory as long as it has at least one active connection. When the last connection to the data store is closed, the data store remains in memory for a "grace period." The data store is unloaded from memory only if no processes have connected to the data store for the duration of the grace period. The grace period can be set or reset at any time. It is only in effect and stays in effect until the next time the grace period is changed.
- *Always.*The data store stays in memory at all times. If the machine on which the data store resides is rebooted, the data store reloads into memory when the TimesTen daemon is started, generally at boot time.
- *Manual.* The data store is manually loaded and unloaded by system administrators using the **ttAdmin** utility.

## Configuring a diskless data store

TimesTen allows you to create and use data stores that do not create disk files of any kind. This form of operation is called "diskless operation," and can be used on machines that have no writable file systems. To enable diskless operation for your installation, you must:

- Inform the TimesTen daemon that it is to operate in diskless mode. You must set the `-diskless` option in the `ttendaemon.options` file. See Chapter 4, "Working with the Oracle TimesTen Data Manager Daemon" in the *Oracle TimesTen In-Memory Database Operations Guide*.
- Use the following data store connection attributes:
  - Set **Logging** to diskless mode (2) or use no logging (0).
  - Set DataStore to any path, even invalid or non existing ones. TimesTen uses the value of DataStore as a unique identifier.
  - Turn DurableCommits off (0)
  - Set Locklevel to data store-level locking (1)
  - Set **Temporary** to temporary data store mode (1).

---

**Note:** Diskless operations are only supported on UNIX systems.

---

TimesTen supports replication between diskless nodes and other nodes. For details about how to enable replication with diskless operation, see the *TimesTen Replication Guide*.

# Copying, migrating, backing up and restoring a data store

The TimesTen utilities for copying, backing up, restoring and migrating a data store allow you to:

- Copy an entire data store or a single table.
- Migrate a data store between releases of TimesTen or different hardware platforms.
- Take a snapshot of a data store and then restore it.
- Add rows of data to a table.
- Rename the owner of tables in a data store

***To migrate a data store between releases of*** *TimesTen*, use the **ttMigrate** utility. This utility saves tables and indexes from a TimesTen data store into a binary file. The tables and indexes can then be restored into another TimesTen data store. This allows you to migrate data between TimesTen releases.

***To migrate a data store between hardware platforms***, use the **ttBulkCp** utility. This utility saves the rows of a table to an ASCII file. It allows you to copy a single table between data stores, including between data stores from different releases of TimesTen or between data stores on different hardware platforms.

***To add rows of data to an existing table***, use the **ttBulkCp** utility. You can save data to an ASCII file and use the **ttBulkCp** utility to load the data rows into a table in a TimesTen data store. The rows you are adding must contain the same number of columns as the table, and the data in each column must be of the type defined for that column. Because the **ttBulkCp** utility works on data stored in ASCII files, you can also use this utility to import data from other applications, provided the number of columns and data types are compatible with those in the table in the TimesTen data store and that the file found is compatible with **ttBulkCp**.

***To take a snapshot of a data store and later restore that data store in the exact same state***, use the **ttBackup** and **ttRestore** utilities or the **ttBackup** and **ttRestore** utilities C functions.

***To rename the owner of tables in a data store***, use the **ttMigrate** utility. When restoring tables, you can use the -rename option to rename the owner of tables. Backing up and restoring a data store

TimesTen's backup and restore facility allows you to create backups of TimesTen data stores and restore the data store at a later time. The primary use for the backup and restore facility is to allow the restoration of a recent state of a data store that has been lost.

Every data store backup contains the information needed to restore the data store as it existed at a the *backup point*; the time the backup began. Restoration of a data store from a given backup restores the modifications of all transactions that committed before the backup point.

In this release, TimesTen supports both *full* and *incremental* backups. An incremental backup moves the backup point of an existing backup forward in time by augmenting the backup with all of the log records created since its backup point.

TimesTen writes a data store backup to a location specified by a *backup path*, which consists of a *directory name* and an optional *basename*. You must specify the backup directory and basename when the backup is created. The basename defaults to the basename of the data store itself if you do not specify a basename.

---

**Note:** You must not manually change the contents of the backup directory. The addition, removal, or modification of any file in the backup directory, except for modifications made by ttBackup and ttRestore themselves, may compromise the integrity of the backup and restoration of the data store from the backup may not be possible.

---

TimesTen also allows *stream* backups. A stream backup writes the data store backup file to `stdout`.

A set of files containing backup information for a given data store, residing at a given backup path is referred to as a *backup instance*. A given backup instance must be explicitly enabled for incremental backups.

An incremental backup can only augment an existing *incremental-enabled* backup of the same data store. Restoring a data store from a backup causes all existing incremental-enabled backups of this data store to become incremental backups to become incremental-disabled.

TimesTen supports the creation of up to eight incremental-enabled backup instances for each data store. If you attempt to start an incremental backup in a ninth backup path, TimesTen returns an error. Incremental backups are supported only for permanent disk-logging data stores.

Information about incremental backups is not retained across data store recovery or restoration. Existing backup instances continue to be usable for restoring the data store, but incremental backups to those instances will have to be reenabled.

A backup operation is atomic: If it completes successfully, it will produce a backup that can be used to restore a data store to the state of its backup point. If it fails for any reason, it leaves the files of the existing backup (if any) intact and its backup point unchanged.

---

**Note:** For full backups, you must have enough disk space available to hold both the existing backup and the new backup, until the new backup succeeds.

---

The files of the existing backup may be modified by a failed full or incremental backup, but not in a way that compromises the ability to restore from them.

An incremental backup typically completes much faster than a full backup, as it has less data to copy. The performance gain of incremental backups over full backups comes at the cost of increased disk usage and longer restoration times. Use incremental backups in concert with full backups in order to achieve a balance between backup time, disk usage, and restoration time.

The backup types supported by TimesTen are:

| Backup Type | File or Stream | Full or Incremental | Incremental -enabled | Comments |
|---|---|---|---|---|
| fileFull | File | Full | No | Default |
| fileFullEnable | File | Full | Yes | |
| fileIncremental | File | Incremental. | Yes | Fails if incremental backup not possible. |
| fileIncrOrFull | File | Either | Yes | Performs fileIncremental if possible; fileFullEnable otherwise. |
| streamFull | Stream | Full | No | |
| incrementalStop | None | None | No | Takes no backup; just disables existing incremental-enabled backup. |

For details on using TimesTen's backup and restore facility, see these references in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

# 3

# *Working with the TimesTen Client and Server*

To access TimesTen data stores on remote machines, applications can use the TimesTen ODBC or JDBC Client driver and TimesTen Server Daemon. Using the TimesTen Client, applications written to use the TimesTen Data Manager can connect transparently to TimesTen data stores on any remote or local machine that has the TimesTen Server Daemon and Data Manager installed.

You can also install the TimesTen Client and TimesTen Server on the same machine and use them to access TimesTen data stores on the local machine. On Solaris and HP-UX, this is useful if you have a 32-bit client application that needs to access a 64-bit data store on the same machine. You can create a client/server connection between any combination of 32/64 bits RedHat Linux, Solaris, and HP-UX platforms.

The TimesTen Server is a process that runs on a server machine. The TimesTen ODBC or JDBC Client is a thin driver that communicates with the TimesTen Server.


On Windows, you can either link the Client applications with the driver manager or directly with the TimesTen Client driver.


On UNIX, you must link Client applications directly with the TimesTen Client driver.

For UNIX, there are Client/Server versions of some TimesTen utilities: ttIsqlCS, ttBulkCpCS, ttMigrateCS and ttSchemaCS.

For details on compiling TimesTen applications, see the *Oracle TimesTen In-Memory Database Java Developer's and Reference Guide* or the *Oracle TimesTen In-Memory Database C Developer's and Reference Guide*.

The main topics in this chapter are:

- Creating and configuring Client DSNs on UNIX
- Accessing a remote data store on UNIX
- Troubleshooting Client/Server problems

## Client/Server Communication

Each TimesTen Client connection requires one Server process. By default, a Server process is spawned at the time a Client requests a connection. By setting the `-serverPool` option in the `ttendaemon.options` file on the Server machine, you can pre-spawn a pool of Server processes. See "Prespawning TimesTen Server processes" on page 84," for details.

When using TimesTen Client/Server there are three ways the TimesTen Client can communicate with the TimesTen Server.

### TCP/IP Communication

By default, the TimesTen Client communicates with the TimesTen Server daemon using TCP/IP sockets. This is the only form of communication available when the TimesTen Client and Server are installed on different machines.

### Shared memory communication

If both the TimesTen Client and Server are installed on the same machine, applications using the TimesTen Client ODBC driver may improve performance by using a shared memory segment for inter-process communication (IPC). Using a shared memory segment allows for the best performance, but consumes more memory. To use a shared memory segment as communication, you must set the server options in the `ttendaemon.options` file. See "Using shared memory for Client/Server IPC" on page 85. You must also define the Network Address of the logical server as `ttShmHost`. See "Creating and configuring Client DSNs on Windows" on page 59 or "Creating and configuring Client DSNs on UNIX" on page 67.

---

**Note:** TimesTen supports a maximum of 16 different instances of the shared memory IPC-enabled server. If an application tries to connect to more than 16 different shared memory segments it receives the ODBC error "Cannot connect to more than 16 SHMIPC-enabled TimesTen Servers."

---

### UNIX domain socket communication

On certain UNIX platforms, if both the TimesTen Client and Server are installed on the same machine, you can use UNIX domain sockets for communication. Using a shared memory segment allows for the best performance, but greater memory usage. Using UNIX domain sockets allows for improved performance over TCP/IP, but with less memory consumption than a shared memory segment connection. See the section in this chapter that describes how to create a Client DSN for your platform. To use domain sockets, you must define the Network Address of the logical server as `ttLocalHost`. See "Creating and configuring Client DSNs on UNIX" on page 67.

## Configuring TimesTen Client and Server

Before configuring the TimesTen Client and Server, read the "TimesTen ODBC and JDBC drivers" and "Data source names" sections of Chapter 2, "Creating TimesTen Data Stores." To connect a TimesTen application to a TimesTen data store using TimesTen Client and Server:

1. Install the TimesTen Server on the machine on which the TimesTen data store resides. This machine is called the "server machine." For information on how to install the TimesTen Server, see the *Oracle TimesTen In-Memory Database Installation Guide*. During installation, choose to have the TimesTen Server automatically started.
2. Install the TimesTen Client on the machine where the Client application resides. This machine is called the "client machine." For information on how to install the TimesTen Client, see the *Oracle TimesTen In-Memory Database Installation Guide*.
3. For JDBC, install the Java Developer's Kit (JDK) on the client machine, where the Java application(s) will be running and set up the environment variables (CLASSPATH and LIBRARY PATH) on the client machine. See Chapter 1, "Configuring the Java Development Environment in the *Oracle TimesTen In-Memory Database Java Developer's and Reference Guide* for details.
4. On the server machine, create and configure a Server DSN corresponding to the TimesTen data store. See "Defining Server DSNs" on page 58.
5. On the client machine, create and configure a Client DSN corresponding to the Server DSN. See "Creating and configuring Client DSNs on UNIX" on page 67 and "Creating and configuring Client DSNs on Windows" on page 59.
6. Set TimesTen Client connection attributes. See Chapter 1, "Data Store Attributes" in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.
7. For C client applications, link as described in "Linking options" in the *Oracle TimesTen In-Memory Database C Developer's and Reference Guide*.

# Running the TimesTen Server Daemon

The TimesTen Server daemon is a subdaemon of the TimesTen daemon. If the TimesTen Server is installed on your machine, the TimesTen Server daemon starts automatically every time the TimesTen daemon is started and stops automatically every time the TimesTen daemon is stopped on your machine.

The TimesTen Server daemon handles request from applications linked with the TimesTen Client ODBC driver.

By default, the 32-bit version of TimesTen Server Daemon listens on TCP port number 16002 and the 64-bit version listens on 16003. System administrators can change the port number during installation to avoid conflicts or for security reasons. The port range is from 1 - 65535. To connect to the TimesTen Server Daemon, Client DSNs are required to specify the port number as part of the logical server name definition or in the connection string.

On Windows, the TimesTen Server daemon is run as user SYSTEM. On UNIX, the TimesTen Server daemon is run as the instance administrator (user `root`, unless the instance is installed as non-root).

For instructions on modifying TimesTen Server Daemon options, see "Modifying the TimesTen Server daemon options" in Chapter 4 of the *Oracle TimesTen In-Memory Database Operations Guide*.

## Server informational messages

The TimesTen Server records "connect," "disconnect" and various warning, error and informational entries in log files.


On Windows, these application messages can be accessed with the Event Viewer.

On UNIX, the TimesTen Server logs messages to a `syslog` facility. You can define the facility used by `syslog` by setting the `-facility` option in the `ttendaemon.options` file or specify an output file by using the `-f` option in the `ttendaemon.options` file. See "Modifying the informational message options on UNIX" on page 82. The `syslog` facility allows messages to be routed in a variety of ways, including recording them to one or more files. The disposition of messages is under the control of the configuration file `/etc/syslog.conf`. See the operating system documentation for `syslog.conf` or `syslogd` for information on how to configure this file. Message files can grow to be quite large. Prune them periodically to conserve disk space.

## Defining Server DSNs

Server DSNs correspond to data stores that are accessed by a TimesTen Server Daemon. You can add or configure a Server DSN while the TimesTen Server is running.

A Server DSN is a TimesTen data manager system DSN. For a description of DSNs and instructions on creating them, see "Creating a DSN on Windows" on page 37 or "Creating a DSN on UNIX" on page 40.

## TimesTen Client connection attributes

This section discusses the connection attributes used by the TimesTen Client driver. You can configure attributes as part of the Client DSN definition or you can configure connection attributes at runtime in the *connection* string that is passed to the ODBC **SQLDriverConnect** function or the *URL* string that is passed to the JDBC **DriverManager.getConnection** method.

The **Authenticate**, **DataStore,** and **ThreadSafe** Data Manager attributes are not allowed in the TimesTen Client connection string.

---

**Note:** The TimesTen Client allows TimesTen Data Manager attributes to be passed in as part of the connection or URL string. However, the Client ignores any Data Manager attributes specified in the ODBC.INI file as part of the TimesTen Client DSN definition.

---

For a complete description of the TimesTen Client connection attributes, see Chapter 1, "Data Store Attributes" in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

# Creating and configuring Client DSNs on Windows

WINDOWS

On Windows, use the ODBC Data Source Administrator to configure logical server names and to define Client DSNs.

This section includes the following topics:

- Creating and configuring a logical server name
- Creating a Client DSN on Windows
- Setting the timeout interval and authentication
- Deleting a server name
- Accessing a remote data store on Windows
- Testing connections

## Creating and configuring a logical server name

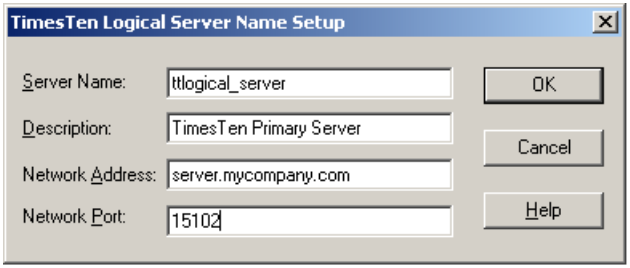
To create and configure a logical server name:

1. On the Windows Desktop, choose **Start > Settings > Control Panel > Administrative Tools > Data Sources (ODBC)**.
   This opens the ODBC Data Source Administrator.
2. Click **User DSN** or **System DSN**.
3. Select a TimesTen Client DSN and click **Configure**. If no Client DSN exists, click **Add**, select **TimesTen Client 6.0** and click **Finish**. This opens the TimesTen Client DSN Setup dialog.
4. Click **Servers**. This opens the TimesTen Logical Server List dialog.
5. Click **Add**. This opens the TimesTen Logical Server Name Setup dialog.
6. In the **Server Name** field, enter a logical server name.
7. In the **Description** field, enter an optional description for the server.
8. In the **Network Address** field, enter the host name or IP address of the server machine. The Network Address must be one of:

| Type of Connection | Network Address |
|---|---|
| Local Client/Server connection that uses shared memory for inter-process communication | `ttShmHost` |
| Remote Client/Server connection | The name of the machine where the TimesTen Server is running. For example, `server. mycompany.com` |

9. In the Network Port field, TimesTen displays the port number on which the TimesTen Logical Server Listens by default. If the TimesTen Server is listening on a different port, enter that port number in the **Network Port** field.

   For example:




10. Click **OK**, then click **Close** in the TimesTen Logical Server List dialog to finish creating the logical server name.

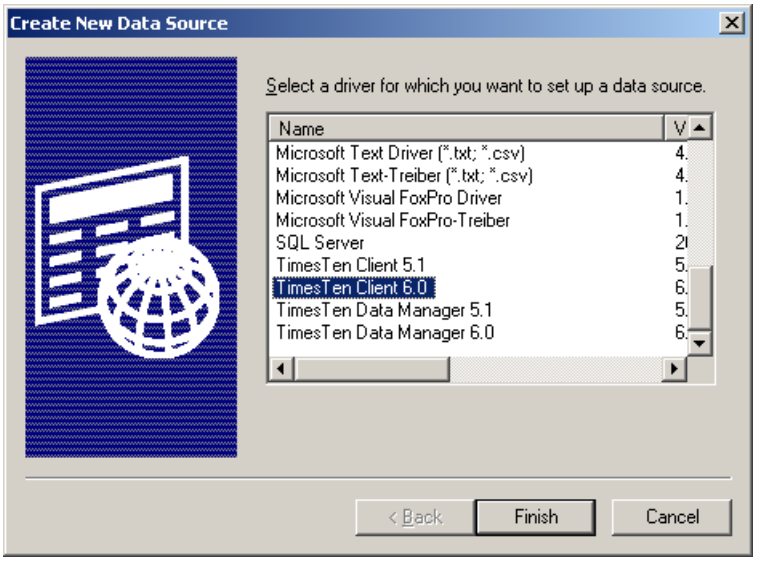
## Creating a Client DSN on Windows

WINDOWS

To define a TimesTen Client DSN:

1. On the Windows Desktop, choose **Start > Settings > Control Panel > Administrative Tools > Data Sources (ODBC)**.
   This opens the ODBC Data Source Administrator.
2. Choose either **User DSN** or **System DSN**. For a description of User DSNs and System DSNs see "Data source names" on page 33.

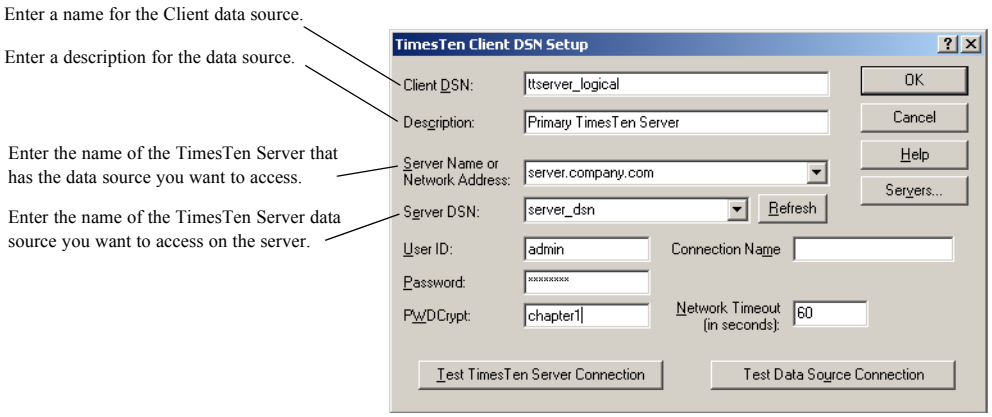
3. Click **Add**. This opens the Create New Data Source dialog.




4. Choose **TimesTen Client 6.0**. Click **Finish**. This opens the TimesTen Client DSN Setup dialog.
5. In the **Client DSN** field, enter a name for the Client DSN.
   - The name must be unique to the current list of defined DSNs on the machine where the Client application resides and can contain up to 32 characters. To avoid potential conflicts, you may want to use a consistent naming scheme that combines the logical server name with the name of the Server DSN. For example, a corporation might have Client DSNs named `Boston_Accounts` and `Chicago_Accounts` where `Boston` and `Chicago` are logical server names and `Accounts` is a Server DSN.
6. In the **Description** field, enter an optional description for the Client DSN.
7. In the **Server Name or Network Address** field, specify the logical server or network address of the server machine.
   - The name can be a host name, IP address or logical server name. The logical server names defined on the client machine can be found in the drop-down list. To define logical server names, click **Servers**.
   - If you do not specify a logical server name in this field, the TimesTen Client assumes that the TimesTen Server Daemon is running on the default TCP/IP port number. Therefore, if your Server is running on a port other than the default port and you do not specify a logical server name in this field, you must specify the port number in the ODBC connection string, using the TCP_Port attribute.

For more information on defining logical server names, see "Creating and configuring a logical server name" on page 59.

8. In the **Server DSN** field, enter the Server DSN corresponding to the data store that the Client application will access.
   - If you do not know the name of the Server DSN, click **Refresh** to obtain a list of Server DSNs that are defined on the machine specified in the **Server Name or Network Address** field. Select the Server DSN from the drop-down list.
   - You must have a network connection to the machine where the TimesTen Server is running.



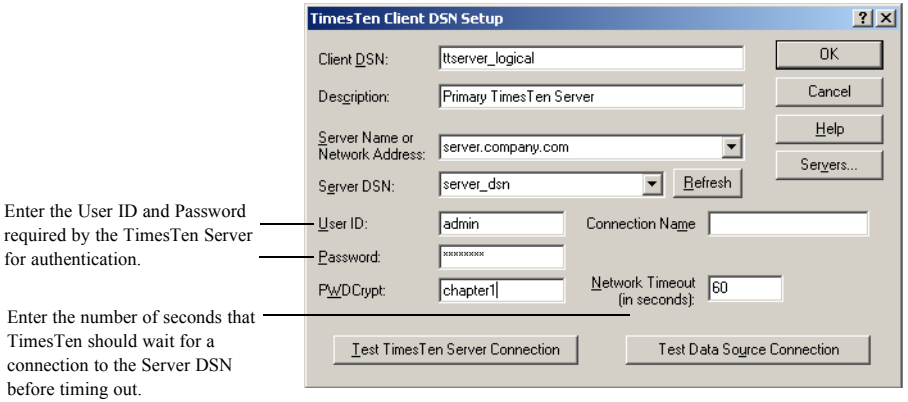

## Setting the timeout interval and authentication

For a description of the Timeout, UID and PWD attributes, see Chapter 1, "Data Store Attributes" in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

To set the timeout interval and authentication:

1. In the **User ID** field, you can enter a user name that is defined on the server machine. If the server DSN that corresponds to this client DSN is defined with Authenticate=1, you must provide a user name either in this field or in the connection string of every application that uses this client DSN. If the server

DSN is defined with **Authenticate**=0, then TimesTen ignores the value entered in this field, unless Access Control is enabled In that case the User ID is required.




2. In the **Password** field, you can enter the password that corresponds to the user ID. Alternatively, you can enter an encrypted password in the **PwdCrypt** field. If the server DSN that corresponds to this client DSN is defined with Authenticate=1, you must provide the password either in this field or in the connection string of every application that uses this client DSN. If the server DSN is defined with **Authenticate**=0, then TimesTen ignores the value entered in this field, unless Access Control is enabled In that case the Password is required.
3. In the **Timeout Interval** field, enter the interval time in seconds. You can enter any non-negative integer. A value of 0 indicates that Client/Server operations should not timeout. The default is 60 seconds. The maximum is 99,999 seconds.
4. Click **OK** to save the setup.

## Deleting a server name

To delete a server name:

1. On the Windows Desktop on the Client machine, choose **Start > Settings > Control Panel**.
2. Double click **ODBC**. This opens the ODBC Data Source Administrator.
3. Click either **User DSN** or **System DSN**.
4. Select a TimesTen Client DSN and click **Configure**. This opens the TimesTen Client DSN Setup dialog.
5. Click **Servers**. This opens the TimesTen Logical Server List dialog.
6. Select a server name from the **TimesTen Servers** list.
7. Click **Delete**.

## Accessing a remote data store on Windows

**Example 3.1** WINDOWS

In this example, the TimesTen Client machine is **client.mycompany.com**. The Client application is accessing the Server DSN on the remote server machine, `server.mycompany.com`. The logical name of the server is **ttserver_logical**.

1. On the server machine `server.mycompany.com`, use the ttStatus utility to verify that the TimesTen Server Daemon is running and to verify the port number it is listening on.
2. Using the procedure in "Defining Server DSNs" on page 58, verify that the Server DSN, **RunData60**, is defined as a System DSN on `server.mycompany.com`.
3. On the Client machine, `client.mycompany.com`, create a logical Server Name entry for the remote TimesTen Server. In the TimesTen Logical Server Name Setup dialog:
   - In the **Server Name** field, enter `ttserver_logical`.
   - In the **Network Address** field, enter `server.mycompany.com`.
   - In the **Network Port** field, enter `16002`. This is the default port number for TimesTen Release 6.0. This value should correspond to the value displayed by ttStatus in Step 1.

   See "Creating and configuring a logical server name" on page 59 for the procedure to open the TimesTen Server Name dialog and for more details.
4. On the Client machine, `client.mycompany.com`, create a Client DSN that corresponds to the remote Server DSN, **RunData60**. In the TimesTen Client DSN Setup dialog, enter the following values:
   - In the **Client DSN** field, enter `RunDataCS60`.
   - In the **Server Name or Network Address** field, enter `ttserver_logical`.
   - In the **Description** field, enter a description for the server. Entering data into this field is optional.
   - In the **Server DSN** field, enter `RunData60`.
5. Run the Client application from the machine `client.mycompany.com` using the Client DSN, **RunDataCS60**. The example below uses the **ttIsqlCS** program installed with TimesTen Client.

   ```
   ttIsqlCS connStr "DSN=RunDataCS60"
   ```

**Example 3.2** This example describes how to access a TimesTen Server that is listening on a port numbered other than the default port number.

Let us consider the Network Address of the TimesTen Server is `server.mycompany.com` and the Server is listening on Port `22222`. The following methods can be used to connect to a Server DS:

1. Define the logical server name `logical_server` with `server.mycompany.com` as the Network Address and `22222` as the Network Port. Define a Client DSN with logical_server as the Server name, Server_DSN as the Server DSN. And execute the command:

   ```
   ttIsqlCS -connStr "DSN=Client_DSN"
   ```

2. Alternatively, define the logical server name `logical_server` with `server.mycompany.com` as the Network Address and `the default port number` as the Network Port. Define a Client DSN with logical_server as the Server name, Server_DSN as the Server DSN. Overwrite the port number in the command:

   ```
   ttIsqlCS -connStr "DSN=Client_DSN; TCP_Port=22222"
   ```

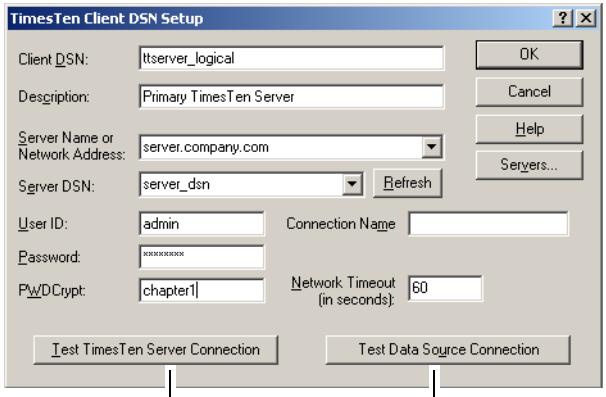
3. Alternatively, define the Server in the connection string. In this case you do not need to define a Client DSN, nor a logical server name.

   ```
   ttIsqlCS -connStr "TTC_Server=server.mycompany.com;
   TTC_Server_DSN=Server_DSN; TCP_Port=22222"
   ```

## Testing connections

To test Client application connections to TimesTen data stores:

1. On the Windows Desktop, choose **Start > Settings > Control Panel.**
2. Double click **ODBC**. This opens the ODBC Data Source Administrator.
3. Click **User DSN** or **System DSN**.
4. Select the TimesTen Client DSN whose connection you want to test and click **Configure**. This opens the TimesTen Client DSN Setup dialog.




Click here to attempt a connection to the TimesTen Server.

Click here to attempt a connection to the TimesTen data source on the TimesTen Server.

5. Click **Test TimesTen Server Connection** to test the connection to TimesTen Server.

   The ODBC Data Source Administrator attempts to connect to TimesTen Server Daemon and displays messages to indicate if it was successful. During this test TimesTen Client verifies that:

   - ODBC, Windows sockets and TimesTen Client are installed on the client machine.
   - The server specified in the **Server Name or Network Address** field of the TimesTen Client DSN Setup dialog is defined and the corresponding machine exists.
   - The TimesTen Server Daemon is running on the server machine.

6. Click **Test Data Source Connection** to test the connection to the Server DSN. The ODBC Data Source Administrator attempts to connect to the TimesTen Server DSN and displays messages to indicate whether it was successful.

   During this test, TimesTen Client verifies that:

   - The Server DSN specified in the **Server DSN** field is defined on the server machine.
   - A Client application can connect to the Server DSN.

# Creating and configuring Client DSNs on UNIX


On UNIX, you define logical server names by editing the TTCONNECT.INI file and you define client DSNs by editing the user ODBC.INI file (for user DNS) or the system ODBC.INI file (for system DSNs). For a description of user and system DSNs, see "Data source names" on page 33.

This section includes the following topics:



## Searching for a DSN

When TimesTen looks for a specific DSN, it looks in the following locations in this order:

1. The file referenced by the ODBCINI environment variable, if it is set
2. The `.odbc.ini` file in the user's home directory, if the ODBCINI environment variable is not set
3. The file referenced by the SYSODBCINI environment variable, if it is set
4. One of the following, if the SYSODBCINI environment variable is not set:
   - The `InstallDir/info/sys.odbc.ini` file for a nonroot installation
   - The `/var/TimesTen/InstanceName/sys.odbc.ini` file for a root installation
5. The `/var/TimesTen/sys.odbc.ini` file, if the SYSODBCINI environment variable is not set

## Creating and configuring a logical server name

You define logical server names in the `/var/TimesTen/`*`instance`*`/sys.ttconnect.ini` file or in a file named by the SYSTTCONNECTINI environment variable. This file is referred to as the TTCONNECT.INI file. The file contains a description, a network address and a port number.

The Network Address must be one of:

| Type of Connection | Network Address |
|---|---|
| Local Client/Server connection that uses UNIX domain sockets | `ttLocalHost` |

| Type of Connection | Network Address |
|---|---|
| Local Client/Server connection that uses shared memory for inter-process communication | `ttShmHost` |
| Remote Client/Server connection | The name of the machine where the TimesTen Server is running. For example, `server. mycompany.com` |

### Examples

**Example 3.3** Below is an example from a TTCONNECT.INI file that defines a logical server name, `ttserver_logical`, for a TimesTen Server daemon running on the machine `server.mycompany.com` and listening on `port 16002`. The instance name of the TimesTen installation is "tt60."

```
[ttserver_logical]
Description=TimesTen Server 6.0
Network_Address=server.mycompany.com
TCP_Port=16002
```

**Example 3.4** If both the client and server are on the same UNIX machine, applications using the TimesTen Client ODBC driver may improve performance by using UNIX domain sockets for communication.

The logical server name must also define the port number on which the TimesTen Server Daemon is listening so that multiple instances of the same version of TimesTen Server Daemon can be run on the same machine. To achieve this, the logical server name definition in TTCONNECT.INI file might look like:

```
[LocalHost_tt60]
Description=
 Local TimesTen Server TimesTen release 6.0 through domain
sockets
Network_Address=ttLocalHost
TCP_PORT=16002
```

**Example 3.5** If both the client and server are on the same machine, applications can use shared memory for inter-process communication. This may result in the best performance.

The logical server name must also define the port number on which the TimesTen Server Daemon is listening in order to make the initial connection. To achieve this, the logical server name definition in TTCONNECT.INI file might look like:

```
[ShmHost_tt60]
Description=
 Local TimesTen Server TimesTen release 6.0 through shared memory
Network_Address=ttShmHost
TCP_PORT=16002
```

## Creating a Client DSN

In the ODBC Data Sources section of the ODBC.INI file, add an entry for the Client DSN. Each entry in this section lists the data source and the name of the ODBC driver that the data source uses. Use the following format for data source entries.

```
[ODBC Data Sources]
data-source-name=name-of-ODBC-driver
```

For example, to add the **RunDataCS_tt60** data source and associate it with the TimesTen Client ODBC driver, make the following entry in the ODBC Data Sources section of the ODBC.INI file.

```
[ODBC Data Sources]
RunDataCS_tt60=TimesTen Client 6.0
```

After the ODBC Data Sources section, add an entry to specify the connection attributes for each data source you have defined. Each data source listed in the ODBC Data Sources section of the ODBC.INI file requires a data source specification section.

The following is an example specification of the TimesTen Client example DSN **RunData_tt60**.

```
[RunDataCS_tt60]
TTC_Server=ttserver_logical
TTC_Server_DSN=RunData_tt60
```

For a description of the Client DSN attributes used in the ODBC.INI file, see Chapter 1, "Data Store Attributes" in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

## Accessing a remote data store on UNIX

**Example 3.6**


In this example, the TimesTen Client application machine is a 32-bit Solaris machine, `client.mycompany.com`. The Client application is accessing the Server DSN `RunData_tt60` on the remote server machine, another 32-bit Solaris machine, `server.mycompany.com`. The logical name of the server is `ttserver_logical`. The instance name of the TimesTen installation is "tt60."

1. On the server machine `server.mycompany.com`, use the ttStatus utility to verify that the TimesTen Server is running and to verify the port number on which it is listening.
2. Verify that the Server DSN **RunData_tt60** exists in the system ODBC.INI file on `server.mycompany.com`.

   There should be an entry in the ODBC.INI file as follows:

   ```
   [RunData_tt60]
   Driver=/opt/TimesTen/tt60/lib/libtten.so
   DataStore=/var/TimesTen/tt60/server/RunData_tt60
   ```

3. Create a logical Server Name entry for the remote TimesTen Server in the TTCONNECT.INI file on `client.mycompany.com`.

   ```
   [ttserver_logical]
   # This value for TCP_Port should correspond to the
   # value reported by ttStatus when verifying that the
   # server is running
   Network_Address=server.mycompany.com
   TCP_Port=16002
   ```

   See "Creating and configuring Client DSNs on UNIX" on page 67 for information on the creating a TTCONNECT.INI file.

4. On the Client machine, `client.mycompany.com`, create a Client DSN corresponding to the remote Server DSN, **RunData_tt60**.

   There should be an entry in the ODBC.INI file as follows:

   ```
   [RunDataCS_tt60]
   TTC_SERVER=ttserver_logical
   TTC_SERVER_DSN=RunData_tt60
   ```

   See "User and system DSNs" on page 33 for information on the location of the proper ODBC.INI file.

5. Run the Client application from the machine `client.mycompany.com` using the Client DSN, **RunDataCS_tt60**. The example below uses the ttIsql program that is installed with TimesTen Client.

   ```
   ttIsqlCS -connStr "DSN=RunDataCS_tt60"
   ```

**Example 3.7**

This example describes how to access a TimesTen Server that is listening on a port numbered other than the default port number.

Let us consider the Network Address of the TimesTen Server is `server.mycompany.com` and the Server is listening on Port `22222`. The following methods can be used to connect to a Server DS:

1. Define the logical server name `logical_server` with `server.mycompany.com` as the Network Address and `22222` as the Network Port. Define a Client DSN with logical_server as the Server name, Server_DSN as the Server DSN. And execute the command:

   ```
   ttIsqlCS -connStr "DSN=Client_DSN"
   ```

2. Alternatively, define the logical server name `logical_server` with `server.mycompany.com` as the Network Address and `the default port number` as the Network Port. Define a Client DSN with logical_server as the Server name, Server_DSN as the Server DSN. Overwrite the port number in the command:

   ```
   ttIsqlCS -connStr "DSN=Client_DSN; TCP_Port=22222"
   ```

3. Alternatively, define the Server in the connection string. In this case you do not need to define a Client DSN, nor a logical server name.

   ```
   ttIsqlCS -connStr "TTC_Server=server.mycompany.com;
   TTC_Server_DSN=Server_DSN; TCP_Port=22222"
   ```

## Testing connections

To test Client application connections to TimesTen data stores:

1. Verify that the client machine can access the server machine.
2. Run `ping` from the client machine to see if a response is received from the server machine.
3. Verify that the TimesTen Server Daemon is running on the server machine.
   - Use `telnet` to connect to the port on which the TimesTen Server Daemon is listening. For example:

     ```
     telnet server.mycompany.com 16002
     ```

   - If you successfully connect to the TimesTen Server Daemon, you will see a message similar to:

     ```
     Connected to server.mycompany.com
     ```

   - If the server machine responds to a command, but TimesTen Server Daemon does not, the TimesTen Server Daemon may not be running. In the case of a failed connection, you will see a message similar to:

     ```
     telnet: Unable to connect to remote host:
     Connection refused
     ```

   - Use the ttStatus utility on the server machine to determine the status and port number of the TimesTen Server. Generally, the TimesTen Server Daemon is

started at installation time. If the TimesTen Server Daemon is not running, you must start it. For information on starting the TimesTen Server, see "Modifying the TimesTen Server daemon options" on page 84.

4. Verify that the Client application can connect to the data store. If you cannot establish a connection to the data store, check that the TTCONNECT.INI file contains the correct information.
5. If the information in the TTCONNECT.INI file is correct, check that a Server DSN corresponding to the data store has been defined properly in the system ODBC.INI file on the machine where the data store resides and where the TimesTen Server Daemon is running.

# Troubleshooting Client/Server problems

This section includes the following topics:



## Cannot connect to theTimesTen Server

You have not correctly identified the system where the TimesTen Server is running.

WINDOWS

On a Windows client machine, you select the TimesTen Server in the TimesTen Data Source Setup dialog that is displayed as part of the ODBC Data Source Administrator. To verify the TimesTen Server:

1. On the Windows Desktop, choose **Start > Settings > Control Panel**.
2. Double click the **ODBC** icon. This opens the ODBC Data Source Administrator.
3. Click the **System DSN** tab. This displays the System Data Sources list.
4. Select the TimesTen Client data source. This opens the TimesTen Client DSN Setup dialog.
5. Click **Servers**. This opens the TimesTen Logical Server List.
6. Select the TimesTen Server from the list. This opens the TimesTen Logical Server Name Setup dialog.
7. Verify that the values for the **Network Address** and **Port Number** are correct. If necessary, change the values.

---

**Note:** If you typed the hostname or network address directly into the Server Name field of the TimesTen Client DSN Setup, the Client tries to connect to the TimesTen Server using the default port.

---

If the Network Address and Port Number values are correct, the TimesTen Server may not be running. See "Starting and stopping the Oracle TimesTen Data Manager Service on Windows" on page 78 for information about starting the server manually. See "Testing connections" on page 65 for more information on identifying this problem.


On UNIX, specify the TimesTen Server with the **TTC_Server** connection attribute in the ODBC.INI file on the client machine. If the value specified for **TTC_Server** is an actual hostname or IP address, the client tries to connect to the TimesTen Server using the default port. In TimesTen, the default port is associated with the TimesTen release number. If the value specified for **TTC_Server** is a logical ServerName, this logical ServerName must be defined in the TTCONNECT.INI file. The TTCONNECT.INI entry for this ServerName needs to correctly define the hostname/IP address and port number on which the TimesTen Server is listening.

If the **Network Address** and **Port Number** values are correct, the TimesTen Server may not be running or did not start. See "Starting and stopping the daemon on UNIX" on page 79 for information about starting the server manually. See "Testing connections" on page 65 for more information on identifying this problem.

### TimesTen Server failed

Check the server's log file. On Windows, server log messages are stored in the Application Log of the Event Viewer. On UNIX, server log messages are stored in `syslog`. See "Creating and configuring Client DSNs on UNIX" on page 67."

The maximum number of concurrent IPC connections to a TimesTen Server allowed by TimesTen is 9,999. However, system limits can take precedence on the number of connections to a single DSN. Client/Server users can change the file descriptor limit to support a large number of connections. For an example, see "Installation prerequisites" in the *Oracle TimesTen In-Memory Database Installation Guide*.

### Cannot find TimesTen Server DSN


On UNIX, verify that the Server DSN is defined in the SYS.ODBC.INI file on the machine running the TimesTen Server.


On Windows, verify that the Server DSN is defined as a System DSN in the ODBC Data Source Administrator on the machine running the TimesTen Server. See "Creating and configuring a logical server name" on page 59."

### TimesTen Server failed to load DRIVER


This error only occurs on UNIX platforms. Open the SYS.ODBC.INI file on the machine running the TimesTen Server and locate the Server DSN you are trying to connect. Verify that the dynamic library specified in the DRIVER attribute for the Server DSN exists and is executable.

### Application times out when accessing TimesTen Server

The default TimeOut interval is 60 seconds.

UNIX To increase this interval on UNIX, change the value of the **TTC_Timeout** attribute in the ODBC.INI file.

WINDOWS To set the timeout interval on Windows, see the instructions in "Setting the timeout interval and authentication" on page 62.

## TimesTen Client loses connection with TimesTen Server

Check to see if the error was due to the Client timing out. Check the TimesTen Server's log to see why the Server may have severed connection with the Client. Use ping to determine if your network is up or try using `telnet` to connect to the TimesTen Server port number.

## Failed to attach to shared memory segment for IPC

While using shared memory segment (SHM) as IPC, the application may see the following error message from the TimesTen Client ODBC Driver if the application reaches the system-defined per-process file-descriptor-limit.

```
SQLState    = S1000,
Native Error = 0,
Message     = [TimesTen][TimesTen 6.0 CLIENT]Falied to
      attach to shared memory segment for IPC. System error: 24
```

This may happen during a connect operation to the Client DSN when the `shmat` system call fails because the application has more open file descriptors than the system-defined per-process file descriptor limit. To correct this problem, you must increase your system-defined per-process file descriptor limit. For additional information on file descriptor limits, see Chapter 8, "System Limits," in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

## Increasing the maximum Server connections on Windows XP

WINDOWS On Windows XP, by default, there can be no more than 47 simultaneous connections to the TimesTen Server. To avoid this limitation:

1. From the Start > Programs > Administrative Tools > Services menu, select **TimesTen Data Manager *release***.
2. On the LogOn tab, check the **Allow service to interact with desktop** checkbox and click **OK**.
3. Restart TimesTen Data Manager *release*

# 4

# *Working with the Oracle TimesTen Data Manager Daemon*

The Oracle TimesTen Data Manager daemon (Oracle TimesTen Data Manager service on Windows) is a process that runs at a high level of privilege. It starts automatically when TimesTen is installed by the root user and each time the operating system is booted. The daemon operates continually in the background.

The TimesTen daemon performs the following functions:

- Manages shared memory access.
- Coordinates recovery.
- Keeps management statistics on what data stores exist, which are in use, and which application processes are active.
- Coordinates deadlock detection.
- Manages RAM policy.
- Starts replication processes.
- Starts the TimesTen Server daemon

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 data stores communicate with the daemon transparently using TimesTen internal routines.

This chapter discusses interaction with the TimesTen daemon on various platforms. It includes the following topics:

## Starting and stopping the Oracle TimesTen Data Manager Service on Windows

WINDOWS

The Oracle TimesTen Data Manager service starts automatically when you install the Oracle TimesTen Data Manager on your Windows system. To manually start and stop the Oracle TimesTen Data Manager service, you can use the **ttDaemonAdmin** utility with the -start or -stop option, or you can use the Windows ODBC Data Source Administrator as follows:

1. Open the ODBC Data Source Administrator:

   On the Windows NT Desktop, choose **Start > Settings > Control Pane**l

   On Windows 2000 and XP, choose **Start > Settings >Control Panel> Administrator Tools**

2. Double-click **Services**. All currently available services are displayed.
3. Select **Oracle TimesTen Data Manager 6.0**, then click the appropriate button to stop or start the service.

---

**Note:** You must have administrative privileges to start and stop the TimesTen service.

---

## Starting and stopping the daemon on UNIX


You must be root or the instance administrator to start and stop the TimesTen daemon.

For root installations, the TimesTen main daemon starts automatically when TimesTen is installed and each time the operating system is booted. The daemon operates continually in the background.

For nonroot installations, the instance administrator must manually start and stop the daemon unless the `setuproot` script has been run.

To manually start and stop the TimesTen main daemon, you can use the **ttDaemonAdmin** utility with the `-start` or `-stop` option, or you can use the TimesTen main daemon startup script for your platform.

The following table shows the location of the TimesTen main daemon startup script by platform. These scripts exist only if the installation was root or if the `setuproot` script has been run in a nonroot installation.

| | |
|---|---|
| **Solaris** | `# /etc/init.d/tt_`*`TTinstance`* `start` |
| **HP-UX** | `# /sbin/init.d/tt_`*`TTinstance`* `start` |
| **Linux** | `# /etc/rc.d/init.d/tt_`*`TTinstance`* `start` |
| **AIX** | `# startsrc -s tt_`*`TTinstance`* |
| **Tru64** | `# /sbin/init.d/tt_`*`TTinstance`* `start` |

To use the main daemon startup script to stop the daemon, enter:

| | |
|---|---|
| **Solaris** | `# /etc/init.d/tt_`*`TTinstance`* `stop` |
| **HP-UX** | `# /sbin/init.d/tt_`*`TTinstance`* `stop` |
| **Linux** | `# /etc/rc.d/init.d/tt_`*`TTinstance`* `stop` |
| **AIX** | `# stopsrc -s tt_`*`TTinstance`* |
| **Tru64** | `# /sbin/init.d/tt_`*`TTinstance`* `stop` |

On an AIX system with a root installation, you can also determine the status of the daemon at any time by entering the command:

```
# lssrc -s TTinstance
```

# Managing TimesTen daemon options

The `ttendaemon.options` file allows you to set and modify TimesTen daemon options. During installation, the installer sets some of these options to correspond to your responses to the installation prompts. The features that the `ttendaemon.options` file controls are:

- The addresses on which the daemon listens
- The verbosity of informational messages
- The minimum and maximum number of TimesTen subdaemons that can exist for the TimesTen instance
- Whether or not the TimesTen Server daemon runs on your system
- Whether you use shared memory segments for client/server inter-process communication
- The number of Server processes that are prespawned on your system

On Windows, the `ttendaemon.options` file is located in the directory:

*`install_dir`*`\srv\info`

On UNIX, the `ttendaemon.options` file is located in the directory:

`/var/TimesTen/`*`TTinstance`*`/` (For root installations)

*`install_dir`*`/info/` (For nonroot installations)

---

**Note:** To make most changes to the `ttendaemon.options` file, you must first stop the TimesTen daemon and restart it after you have completed your changes. For TimesTen Server options, it is only necessary to stop the server. It is not necessary to stop the TimesTen daemon.

---

The rest of this section includes the following topics:

- Determining the daemon listening address
- Informational messages
- Changing the allowable number of subdaemons
- Diskless operations

## Determining the daemon listening address

By default, the TimesTen main daemon, all subdaemons and agents listen on a socket for requests, using any available address. All TimesTen utilities and agents use the loopback address to talk to the main daemon, and the main daemon uses the loopback address to talk to agents.

The `-listenaddr` entry in a separate line in the `ttendaemon.options` file tells the TimesTen daemons to listen on the specific address indicated in the value supplied. The address specified with this option can be either a host name or a numerical IP address.

To explicitly specify the address on which the daemons should listen on a separate line in the ttendaemon.options file, enter:

```
-listenaddr address
```

For example, if you want to restrict the daemon to listen to just the loopback address, you say either:

```
-listenaddr 127.0.0.1
```

or

```
-listenaddr localhost
```

This means that only processes on the local machine can communicate with the daemon. Processes from other machines would be excluded, so you would not be able to replicate to or from other machines, or grant client access from other machines.

If you have multiple ethernet cards on different subnets, you can specify `-listenaddr` entries to control which machines can connect to the daemon.

You can enter up to four addresses on which to listen by specifying the option and a value on up to four separate lines in the `ttendaemon.options` file. In addition to the addresses you specify, the loopback address is always listened on.

## Informational messages

As the daemon operates, it generates error, warning and informational messages. These messages may be useful for TimesTen system administration and for debugging applications.

### Modifying the informational message options on Windows

On Windows, the TimesTen Service (daemon) messages are logged in the Event Viewer.

WINDOWS

To turn off detailed log messages, add a '#' before the `-verbose` to comment it out in the `ttendaemon.options` file.

To log daemon messages to a user-specified file instead of sending them to syslog, add the following on a separate line of the `ttendaemon.options` file:

```
-f filename
```

To view log messages, follow these steps:

1. Open the **Event Viewer** window on your Windows Desktop.
2. From the Log menu, choose **Application**.

   The window changes to display only log messages generated by applications. Any messages with the word "TimesTen" in the "Source" column were generated by the Oracle TimesTen Data Manager service.
3. To view any TimesTen message, double-click the message summary.

The message window is displayed. You can view additional messages by clicking **Next** / **Previous** or the up / down arrows, depending on your version of Windows.

---

**Note:** You can also view messages using the **ttDaemonLog** utility.

---

### Modifying the informational message options on UNIX

On UNIX systems, daemon messages are routed in a variety of ways, including recording them to a file. These files can grow to be quite large. You should prune them periodically to conserve disk space.

To specify the `syslog` facility used to log TimesTen Daemon and subdaemon messages, on a separate line of the `ttendaemon.options` file add:

`-facility` *`name`*

Possible *`name`* values are: `auth`, `cron`, `daemon`, `local0-local7`, `lpr`, `mail`, `news`, `user`, or `uucp`.

To turn off detailed log messages, add a "#" before the `-verbose` in the `ttendaemon.options` file to comment it out.

To log daemon messages to a user-specified file instead of sending them to syslog, on a separate line of the `ttendaemon.options` file add:

`-f` *`filename`*

When logging messages to a file, to additionally prepend the date to the message, add the following on a separate line of the `ttendaemon.options` file:

`-showdate`

## Changing the allowable number of subdaemons

TimesTen uses subdaemons to manage data stores. The main TimesTen daemon spawns subdaemons dynamically as they are needed. You can manually specify a range of subdaemons that the daemon may spawn, by specifying a minimum and maximum.

At any point in time, one subdaemon is potentially needed for TimesTen process recovery for each failed application process that is being recovered at that time.

By default, the maximum number of subdaemons is 50.

By default, TimesTen spawns a minimum of 4 subdaemons. However, you can change these settings by assigning new values to the `-minsubs` and `-maxsubs` options in the `ttendaemon.options` file.

## Diskless operations

TimesTen allows you to work in diskless mode on UNIX systems. In diskless mode the daemon does not use any disk files for its own purposes. In diskless

mode you can only work with temporary data stores. To set diskless mode, you must add the `-diskless` option to the `ttendaemon.options` file.

1. If it does not already exist, create the `ttendaemon.options` file.
2. In this file, put a line consisting of the string `-diskless`.
3. Stop and restart the daemon.

# Managing TimesTen Client/Server daemon options

This section includes the following topics:

- Modifying the TimesTen Server daemon options
- Controlling the TimesTen Server daemon
- Prespawning TimesTen Server processes
- Using shared memory for Client/Server IPC
- Controlling the TimesTen Server log messages

## Modifying the TimesTen Server daemon options

The TimesTen Server is a subdaemon of the TimesTen daemon that operates continually in the background. To modify the TimesTen Server daemon options, you must:

1. Stop the TimesTen Server
2. Modify the options in the `ttendaemon.options` file as described in the following sections
3. Restart the TimesTen Server.

## Controlling the TimesTen Server daemon

The `-server` *`portno`* entry in a separate line in the `ttendaemon.options` file tells the TimesTen daemon to start the TimesTen Server daemon and what port to use. The *`portno`* is the port number on which the server will listen.

If the TimesTen Server is installed, you can enable or disable the TimesTen Server daemon by:

- To enable the Server daemon, remove the comment symbol '#' in front of the `-server` *`portno`* entry.
- To disable the Server daemon, add a comment symbol '#' in front of the `-server` *`portno`* entry.

## Prespawning TimesTen Server processes

Each TimesTen Client connection requires one server process. By default, a server process is spawned when a client requests a connection.

You can prespawn a pool of server processes, making them immediately available for a client connection, thus improving client/server connection performance.

The `-serverpool` *`number`* entry in a separate line in the `ttendaemon.options` file on the Server machine tells the TimesTen Server daemon create *`number`* processes. If this option is not specified, 0 processes are pre-spawned.

If you request more process than allowed by your operating system, a warning is returned. Regardless of the number of processes requested, an error does not occur unless a client requests a connection when no more are available on the system. If there are no items in the server pool, a new process is spawned when a connection is requested, as long as you have not met the operating system limit.

---

**Note:** This option is available for all systems, except Linux RedHat 8 systems. These changes to the TimesTen Server daemon do not occur until the TimesTen daemon is restarted.

---

## Using shared memory for Client/Server IPC

By default, TimesTen uses TCP/IP communication between applications linked with the TimesTen Client driver and the TimesTen Server.

Where the client application resides on the same machine as the TimesTen Server, you can alternatively use shared memory for the inter-process communication (IPC).

This can be useful for performance purposes or to allow 32-bit client applications to communicate with a 64-bit data store on the server. Before using shared memory as IPC verify that you have configured your system correctly. See "Installation prerequisites" in Chapter 2 of the *Oracle TimesTen In-Memory Database Installation Guide*.

The `-serverShmIpc` entry in a separate line in the `ttendaemon.options` file tells the TimesTen Server daemon to accept a client connection that intends to use a shared memory segment for IPC.

If this entry is missing, add this line to the `ttendaemon.options` file to start the TimesTen Server daemon with shared memory IPC capability when the TimesTen daemon is restarted.

If the entry exists, add the `#` symbol before the line in the `ttendaemon.options` file to comment it out. The TimesTen Server is no longer started with shared memory IPC capability when the TimesTen daemon starts.

---

**Note:** TimesTen supports a maximum of 16 different instances of the shared memory IPC-enabled server. If an application tries to connect to more than 16 different shared memory segments it receives an ODBC error.

---

### Managing the size of the shared memory segment

The `-serverShmSize` *`size`* entry in a separate line in the `ttendaemon.options` file tells the TimesTen Server daemon to create a shared memory segment of the specified size in MB.

If this entry is missing, the TimesTen Server daemon creates a shared memory segment of 64MB.

An appropriate value for the shared memory segment depends on:

- The expected number of concurrent client/server connections to all data stores that belong to an instance of the TimesTen Server.
- The number of concurrent allocated statements within each such connection.
- The amount of data being transmitted for a query.

Some guidelines for determining the size of the shared memory segment include:

- TimesTen needs 1 MB of memory for internal use.
- Each connection needs a fixed block of 16 KB.
- Each statement starts with a block of 16 KB for the IPC. But, this size is increased or decreased depending upon the size of the data being transmitted for a query. TimesTen increments the statement buffer size by doubling it and decreases it by halving it.

For example, if the user application anticipates a max of 100 simultaneous shared-memory-enabled client/server connections, and if each connection is anticipated to have a maximum of 50 statements, and the largest query returns 128 KB of data, use this formula to configure the `serverShmSize`:

```
serverShmSize = 1 MB + (100 * 16) KB + (100 * 50 * 128) KB
              = 1 MB + 2 MB + 625 MB =  628 MB
```

This is the most memory required for this example. The entire memory segment would be used only if all 100 connections have 50 statements each and each statement has a query that returns 128 KB of data in a row of the result.

In this example, if you configured the `serverShmSize` to 128 MB, either a new shared-memory-enabled client/server connection is refused by the TimesTen Server or a query may fail due to lack of resources within the shared memory segment.

### Changing the size of the shared memory segment

Once configured, to change the value of the shared memory segment you must stop the TimesTen Server. Stopping the server detaches all existing client/server connections to any data store that is associated with that instance of the TimesTen Server. The steps for modifying the value of the `-serverShmSize` option are:

1. Use the **ttDaemonAdmin** utility to stop the TimesTen Server.
2. Modify the value of `-serverShmSize` in the `ttendaemon.options` file.
3. Use the **ttDaemonAdmin** utility to restart the TimesTen Server.

## Controlling the TimesTen Server log messages

The `-noserverlog` entry in a separate line in the `ttendaemon.options` file tells the TimesTen daemon to turn off logging of connects and disconnects from the client applications.

If the TimesTen Server is installed, you can enable or disable logging of connect and disconnect messages by:

- To enable logging, add a comment symbol '#' before the `-noserverlog` entry.
- To disable logging, remove the comment symbol '#' before the `-noserverlog` entry.

# Modifying the TimesTen web server options

The TimesTen web server is a subdaemon of the TimesTen daemon that operates continually in the background. The TimesTen web server allows users to use the Cache Administrator, a web-based interface that allows you to work with cache groups. For more information on cache groups, see the *TimesTen Cache Connect to Oracle Guide*. If the TimesTen web server is specified in the `ttendaemon.options` file, then the TimesTen web server starts and stops with the TimesTen daemon.

To change whether the TimesTen web server is started by the daemon, perform the following tasks:

1. Stop the TimesTen daemon.
2. Modify the `ttendaemon.options` file as described in "Controlling the TimesTen web server".
3. Restart the TimesTen daemon.

### Controlling the TimesTen web server

The existence of a `-webserver` entry in a separate line in the `ttendaemon.options` file tells the TimesTen daemon to start the TimesTen web server.

If the TimesTen web server is installed, you can enable or disable the TimesTen web server by:

- To start the web server, remove the comment symbol '#' in front of the `-webserver` entry.
- To stop the web server, add a comment symbol '#' in front of the `-webserver` entry.

---

**Note:** These changes do not take effect until the TimesTen daemon is restarted.

---

You can also start and stop the web server by using the **ttDaemonAdmin** utility.

# 5

# *Using the ttIsql Utility*

The TimesTen **ttIsql** utility is a general tool for working with a TimesTen data source. The **ttIsql** command line interface is used to execute SQL statements and built-in **ttIsql** commands to perform various operations. Some common tasks that are typically accomplished using **ttIsql** include:

- Data store setup and maintenance. Creating tables and indexes, altering existing tables and updating table statistics can be performed quickly and easily using **ttIsql**.
- Retrieval of information on data store structures. The definitions for tables, indexes and cache groups can be retrieved using built-in **ttIsql** commands. In addition, the current size and state of the data store can be displayed.
- Optimizing data store operations. The **ttIsql** utility can be used to alter and display query optimizer plans for the purpose of tuning SQL operations. The time required to execute various ODBC function calls can also be displayed.

This chapter describes how the **ttIsql** utility is used to perform these types of tasks. The topics are:



For more information on TimesTen SQL and for a detailed description of all **ttIsql** commands see the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

## Batch mode vs. interactive mode

The **ttIsql** utility can be used in two distinctly different ways: batch mode or interactive mode. When **ttIsql** is used in interactive mode, users type commands directly into **ttIsql** from the console. When **ttIsql** is used in batch mode, a prepared script of **ttIsql** commands is executed by specifying the name of the file containing the commands.

Batch mode is commonly used for the following types of tasks:

- Performing periodic maintenance operations including the updating of table statistics, compacting the data store and purging log files.
- Initializing a data store by creating tables, indexes and cache groups and then populating the tables with data.
- Generating simple reports by executing common queries.

Interactive mode is suited for the following types of tasks:

- Experimenting with TimesTen features, testing design alternatives and improving query performance.
- Solving data store problems by examining data store statistics.
- Any other data store tasks that are not performed routinely.

By default, when starting **ttIsql** from the shell, **ttIsql** is in interactive mode. The **ttIsql** utility prompts you to type in a valid **ttIsql** built-in command or SQL statement by printing the **Command>** prompt.

**Example 5.1**

```
C:\>ttIsql

ttIsql (c) 1996-2005, Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.
All commands must end with a semicolon character.


Command>
```

All built-in **ttIsql** commands and SQL statements should be terminated with a semicolon (;) character. The semicolon character tells **ttIsql** that the preceding command is ready to be processed.

Batch mode can be accessed in two different ways. The most common way is to specify the `-f` option on the **ttIsql** command line followed by the name of file to run.

**Example 5.2** For example, executing a file containing a CREATE TABLE statement will look like this:

```
C:\>ttIsql -f create.sql MY_DSN

ttIsql (c) 1996-2005, Oracle.  All rights reserved.
```

```
Type ? or "help" for help, type "exit" to quit ttIsql.
All commands must end with a semicolon character.

Command> connect "DSN=MY_DSN;Overwrite=1"
Connection successful: DSN=MY_DSN;DataStore=E:\ds\MY_DSN;
DRIVER=E:\WINNT\System32\TTdv60.dll;
(Default setting AutoCommit=1)

Command> run "create.sql"

CREATE TABLE LOOKUP (KEY INTEGER NOT NULL PRIMARY KEY, VALUE CHAR
(64))

Command> exit
Disconnecting...
Done.

C:\>
```

The other way to use batch mode is to enter the `run` command directly from the interactive command prompt. The `run` command is followed by the name of the file containing **ttIsql** built-in commands and SQL statements to execute.

**Example 5.3**

```
Command> run "create.sql";

CREATE TABLE LOOKUP (KEY INTEGER NOT NULL PRIMARY KEY, VALUE CHAR
(64))
Command>
```

## Using ttIsql's online help

The **ttIsql** utility has an online version of command syntax definitions and descriptions for all built-in **ttIsql** commands. To access this online help from within **ttIsql** use the `help` command. To view a detailed description of any built-in **ttIsql** commands type the `help` command followed by one or more **ttIsql** commands to display help for. The example below displays the online description for the `connect` and `disconnect` commands.

**Example 5.4**

```
Command> help connect disconnect;

All commands must end with a semicolon character.
Arguments in <> are required.
Arguments in [] are optional.

Command Usage: connect [DSN|connection_string]
Command Aliases: (none)
Description: Connects to the data source specified by the
optional DSN or
connection string argument. If an argument is not given, then the
DSN or
connection string from the last successful connection is used.
Requires an active connection: NO
Requires autocommit turned off: NO
Reports elapsed execution time: YES
Works only with a TimesTen data source: NO
Example: connect; -or- connect RunData; -or- connect
"DSN=RunData;Overwrite=1";

Command Usage: disconnect
Command Aliases: (none)
Description: Disconnects from the currently connected data
source. If a
transaction is active when disconnecting then the transaction
will be rolled
back automatically. If a connection exists when executing the
"bye", "quit" or
"exit" commands then the "disconnect" command will be executed
automatically.
Requires an active connection: YES
Requires autocommit turned off: NO
Reports elapsed execution time: YES
Works only with a TimesTen data source: NO
Example: disconnect;
Command>
```

To view a short description of all **ttIsql** built-in commands type the `help` command without an argument. To view a detailed description of all built-in **ttIsql** commands type the `help` command followed by the `all` argument.

# Using ttIsql's 'editline' feature (UNIX only)

On UNIX systems, you can use the 'editline' library to set up emacs (default) or vi bindings that enable you to scroll through previous **ttIsql** commands, as well as edit and resubmit them. This feature is not available or needed on Windows.

The set up and keystroke information is described for each type of editor:

- Emacs binding
- vi binding

## Emacs binding

To get the current settings, create a file `~/.editrc` and put "bind" on the last line of the file, run **ttIsql**. The editline lib will print the current bindings.

The keystrokes when using **ttIsql** with the emacs binding are:

| Keystroke | Action |
|---|---|
| <Left-Arrow> | Move the insertion point left (back up) |
| <Right-Arrow> | Move the insertion point right (forward) |
| <Up-Arrow> | Scroll to the command prior to the one being displayed. Places the cursor at the end of the line. |
| <Down-Arrow> | Scroll to a more recent command history item and put the cursor at the end of the line. |
| <Ctrl-A> | Move the insertion point to the beginning of the line. |
| <Ctrl-E> | Move the insertion point to the end of the line. |
| <Ctrl-K> | "Kill" (Save and erase) the characters on the command line from the current position to the end of the line. |
| <Ctrl-Y> | "Yank" (Restore) the characters previously saved and insert them at the current insertion point. |
| <Ctrl-F> | Forward char - move forward 1 (see Right Arrow) |
| <Ctrl-B> | Backward char - move back 1 (see Left Arrow) |
| <Ctrl-P> | Previous History (see Up Arrow) |
| <Ctrl-N> | Next History (see up Down Arrow) |

## vi binding

To use the vi bindings, create a file `${HOME}/.editrc` and put "`bind-v`" in the file, run **ttIsql**. To get the current settings, create a file `${HOME}/.editrc` and put "bind" on the last line of the file. When you execute **ttIsql**, the editline lib will print the current bindings.

The keystrokes when using **ttIsql** with the vi binding are:

| Keystroke | Action |
|---|---|
| <Left-Arrow>, h | Move the insertion point left (back up) |
| <Right-Arrow>, l | Move the insertion point right (forward) |
| <Up-Arrow>, k | Scroll to the prior command in the history and put the cursor at the end of the line. |
| <Down-Arrow>, j | Scroll to the next command in the history and put the cursor at the end of the line. |
| ESC | Vi Command mode |
| 0, $ | Move the insertion point to the beginning of the line, Move to end of the line. |
| i, I | Insert mode, Insert mode at beginning of the line |
| a, A | Add ("Insert after") mode, Append at end of line |
| R | Replace mode |
| C | Change to end of line |
| B | Move to previous word |
| e | Move to end of word |
| <Ctrl-P> | Previous History (see Up Arrow) |
| <Ctrl-N> | Next History (see up Down Arrow) |

## Using ttIsql's command history

The **ttIsql** utility stores a list of the last 100 commands executed within the current **ttIsql** session. The commands in this list can be viewed or executed again without having to type the entire command over. Both SQL statements and built-in **ttIsql** commands are stored in the history list. Use the `history` command (or `h` for short) to view the list of previously executed commands.

**Example 5.5**

```
Command> h;
8     INSERT INTO T3 VALUES (3)
9     INSERT INTO T1 VALUES (4)
10    INSERT INTO T2 VALUES (5)
11    INSERT INTO T3 VALUES (6)
12    autocommit 0
13    showplan
14    SELECT * FROM T1, t2, t3 WHERE A=B AND B=C AND A=B
15    trytbllocks 0
16    tryserial 0
17    SELECT * FROM T1, t2, t3 WHERE A=B AND B=C AND A=B
Command>
```

The `history` command displays the last 10 SQL statements or **ttIsql** built-in commands executed. To display more than that last 10 commands specify the maximum number to display as an argument to the `history` command.

Each entry in the history list is identified by a unique number. The `!` character followed by the number of the command can be used to execute the command again.

For example:

**Example 5.6**

```
Command>
Command> ! 12;

autocommit 0
Command>

To execute the last command again simply type a sequence of two !
characters.

Command> !!;

autocommit 0
Command>
```

To execute the last command that begins with a given string type the **!** character followed by the first few letters of the command. For example:

**Example 5.7**

```
Command> ! auto;

autocommit 0
Command>
```

## Working with transactions

The **ttIsql** utility has several built-in commands for managing transactions. These commands are summarized below:

- **`autocommit` –** Turns on or off the autocommit feature.
- **`commit` –** Commits the current transaction.
- **`commitdurable` –** Commits the current transaction and ensures that the committed work will be recovered in case of data store failure.
- **`rollback` –** Rolls back the current transaction.
- **`isolation` –** Changes the transaction isolation level.
- **`sqlquerytimeout` –** Specifies the number of seconds to wait for a SQL statement to execute before returning to the application.

When starting **ttIsql** the autocommit feature is turned on by default. In this mode every SQL operation against the data store is committed automatically. To turn the autocommit feature off execute **ttIsql**'s built-in `autocommit` command with an argument of 0.

When autocommit is turned off transactions must be committed or rolled back manually by executing **ttIsql**'s `commit`, `commitdurable` or `rollback` commands. The `commitdurable` command will ensure that the transaction's effect is preserved in case of data store failure.

The `isolation` command can be used to change the current connection's transaction isolation properties. The isolation can be changed only at the beginning of a transaction. The `isolation` command accepts one of the following constants: READ_COMMITTED and SERIALIZABLE. If the `isolation` command is executed without an argument then the current isolation level is reported.

The `sqlquerytimeout` command sets the timeout period for SQL statements. If the execution time of a SQL statement exceeds the number of seconds set by `sqlquerytimeout`, the SQL statement is not executed and an 6111 error is generated. For details, see "Setting a timeout value for executing SQL statements" in the *Oracle TimesTen In-Memory Database Java Developer's and Reference Guide* and "Setting a timeout value for executing SQL statements" in the *Oracle TimesTen In-Memory Database C Developer's and Reference Guide*.

The example below demonstrates the common use of **ttIsql**'s built-in transaction management commands.

**Example 5.8**

```
E:\>ttIsql
ttIsql (c) 1996-2005, Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.
All commands must end with a semicolon character.
```

```
Command> connect "DSN=MY_DSN";
Connection successful:
DSN=MY_DSN;DataStore=E:\ds\MY_DSN;DRIVER=E:\WINNT\System32\
      TTdv60.dll;
(Default setting AutoCommit=1)
Command> autocommit 0;
Command> CREATE TABLE LOOKUP (KEY INTEGER NOT NULL PRIMARY KEY,
VALUE CHAR (64));
Command> commit;
Command> INSERT INTO LOOKUP VALUES (1, 'ABC');
1 row inserted.
Command> SELECT * FROM LOOKUP;
< 1,
ABC
>
1 row found.
Command> rollback;
Command> SELECT * FROM LOOKUP;
0 rows found.
Command> isolation;
isolation = READ_COMMITTED
Command> commitdurable;
Command> sqlquerytimeout 10;
Command> sqlquerytimeout;
Query timeout = 10 seconds
Command> disconnect;
Disconnecting...
Command> exit;
Done.
E:\>
```

## Displaying data store information

There are several built-in **ttIsql** commands that display information on data store structures. The most useful commands are summarized below:

- **describe** – Displays information on tables, prepared statements and procedures.
- **cachegroups** – Displays the attributes of cache groups.
- **dssize** – Reports the current sizes of the permanent and temporary data store partitions.
- **monitor** – Displays a summary of the current state of the data store.

The `describe` command is used to display information on table and result set columns as well as parameters for prepared SQL statements and built-in procedures. The argument to the `describe` command can be the name of a table, a built-in procedure, a SQL statement or a command id for a previously prepared SQL statement.

**Example 5.9**

```
Command> CREATE TABLE T1 (KEY INTEGER NOT NULL PRIMARY KEY, VALUE
CHAR (64));
Command> describe T1;

Table USER.T1:
  Columns:
   *KEY                              INTEGER NOT NULL
    VALUE                            CHAR (64)

1 table found.
(primary key columns are indicated with *)
Command> describe SELECT * FROM T1 WHERE KEY=?;

Prepared Statement:
  Parameters:
    Parameter 1                      INTEGER
  Columns:
    KEY                              INTEGER NOT NULL
    VALUE                            CHAR (64)
Command> describe ttOptUseIndex;

Procedure TTOPTUSEINDEX:
  Parameters:
    Parameter INDOPTION                      VARCHAR (1024)
  Columns:
    (none)

1 procedure found.
Command>
```

The `cachegroups` command is used to provide detailed information on cache groups defined in the current data store. The attributes of the root and child tables defined in the cache group are displayed in addition to the WHERE clauses associated with the cache group and the DURATION value for cache groups that use cache aging. The argument to the `cachegroups` command is the name of the cache group that you want to display information for.

**Example 5.10**

```
Command> cachegroups MY_CACHE_GROUP;

Cache Group USER.MY_CACHE_GROUP:

  Duration: 40 Minutes

  Root Table: USER.T1
  Where Clause: (T1.KEY < 100)
  Type: Not Propagate

  Child Table: USER.T2
  Where Clause: (none)
  Type: Propagate

1 cache group found.
Command>
```

The `dssize` command is used to report the current memory status of the permanent and temporary partitions as well as the maximum, allocated and in-use sizes for the data store. The `monitor` command displays all of the information provided by the `dssize` command plus additional statistics on the number of connections, checkpoints, lock timeouts, commits, rollbacks and other information collected since the last time the data store was loaded into memory.

**Example 5.11**

```
Command> monitor;

  TIME_OF_1ST_CONNECT:        Thu Sep 23 11:51:38 2005
  DS_CONNECTS:                4
  DS_DISCONNECTS:             0
  DS_CHECKPOINTS:             2
  DS_CHECKPOINTS_FUZZY:       0
  DS_COMPACTS:                0
  PERM_ALLOCATED_SIZE:        204800
  PERM_IN_USE_SIZE:           780
  PERM_IN_USE_HIGH_WATER:     780
  TEMP_ALLOCATED_SIZE:        36864
  TEMP_IN_USE_SIZE:           2048
  TEMP_IN_USE_HIGH_WATER:     2048
  SYS18:                      0
  XACT_BEGINS:                16
  XACT_COMMITS:               15
```

```
  XACT_D_COMMITS:              0
  XACT_ROLLBACKS:              0
  LOG_FORCES:                  2
  DEADLOCKS:                   0
  LOCK_TIMEOUTS:               0
  LOCK_GRANTS_IMMED:           290
  LOCK_GRANTS_WAIT:            0
  CMD_PREPARES:                15
  CMD_REPREPARES:              0
  CMD_TEMP_INDEXES:            0
  LAST_LOG_FILE:               0
  REPHOLD_LOG_FILE:            -1
  REPHOLD_LOG_OFF:             -1
  REP_XACT_COUNT:              0
  REP_CONFLICT_COUNT:          0
  REP_PEER_CONNECTIONS:        0
  REP_PEER_RETRIES:            0
  FIRST_LOG_FILE:              0
  LOG_BYTES_TO_LOG_BUFFER:     10960
  LOG_FS_READS:                0
  LOG_FS_WRITES:               2
  LOG_BUFFER_WAITS:            0
  CHECKPOINT_BYTES_WRITTEN:    1037176
  SYS1:                        0
  SYS2:                        0
  SYS3:                        0
  SYS4:                        0
  SYS5:                        0
  SYS6:                        0
  SYS7:                        23
  SYS8:                        0
  SYS10:                       0
  SYS11:                       0
  SYS12:                       0
  SYS13:                       0
  SYS14:                       0
  SYS15:                       0
  SYS16:                       0
  SYS17:                       0
  SYS19:                       0
  SYS9:

Command>
```

## Viewing and changing query optimizer plans

The built-in `showplan` command is used to display the query optimizer plans used by the TimesTen Data Manager for executing queries. In addition, **ttIsql** contains built-in query optimizer hint commands for altering the query optimizer plan. By using the `showplan` command in conjunction with the built-in commands summarized below, the optimum execution plan can be designed. For detailed information on the TimesTen query optimizer see Chapter 9 the *Oracle TimesTen In-Memory Database C Developer's and Reference Guide*.

- **optprofile** – Displays the current optimizer hint settings and join order.
- **setjoinorder** – Sets the join order.
- **setuseindex** – Sets the index hint.
- **tryhash** – Enables or disables the use of hash indexes.
- **trymergejoin** – Enables or disables merge joins.
- **trynestedloopjoin** – Enables or disables nested loop joins.
- **tryserial** – Enables or disables serial scans.
- **trytmphash** – Enables or disables the use of temporary hash indexes.
- **trytmptable** - Enables or disables the use of an intermediate results table.
- **trytmpttree** – Enables or disables the use of temporary ttree indexes.
- **tryttree** – Enables or disables the use of ttree indexes.
- **tryrowid** – Enables or disables the use of rowid scans.
- **trytbllocks** – Enables or disables the use of table locks.
- **unsetjoinorder** – Clears the join order.
- **unsetuseindex** – Clears the index hint.

When using the `showplan` command and the query optimizer hint commands the auto-commit feature must be turned off. Use **ttIsql**'s `autocommit` built-in command to turn auto-commit off.

The example below shows how these commands can be used to change the query optimizer execution plan.

**Example 5.12**

```
Command> CREATE TABLE T1 (A INTEGER);
Command> CREATE TABLE T2 (B INTEGER);
Command> CREATE TABLE T3 (C INTEGER);
Command>
Command> INSERT INTO T1 VALUES (3);
1 row inserted.
Command> INSERT INTO T2 VALUES (3);
1 row inserted.
Command> INSERT INTO T3 VALUES (3);
1 row inserted.
Command> INSERT INTO T1 VALUES (4);
1 row inserted.
```

```
Command> INSERT INTO T2 VALUES (5);
1 row inserted.
Command> INSERT INTO T3 VALUES (6);
1 row inserted.
Command>
Command> autocommit 0;
Command> showplan;
Command> SELECT * FROM T1, T2, T3 WHERE A=B AND B=C AND A=B;

Query Optimizer Plan:

  STEP:         1
  LEVEL:        3
  OPERATION:    TblLkSerialScan
  TBLNAME:      T1
  IXNAME:       <NULL>
  PRED:         <NULL>
  OTHERPRED:    <NULL>

  STEP:         2
  LEVEL:        3
  OPERATION:    TblLkSerialScan
  TBLNAME:      T2
  IXNAME:       <NULL>
  PRED:         <NULL>
  OTHERPRED:    T1.A = T2.B AND T1.A = T2.B

  STEP:         3
  LEVEL:        2
  OPERATION:    NestedLoop
  TBLNAME:      <NULL>
  IXNAME:       <NULL>
  PRED:         <NULL>
  OTHERPRED:    <NULL>

  STEP:         4
  LEVEL:        2
  OPERATION:    TblLkSerialScan
  TBLNAME:      T3
  IXNAME:       <NULL>
  PRED:         <NULL>
  OTHERPRED:    T2.B = T3.C

  STEP:         5
  LEVEL:        1
  OPERATION:    NestedLoop
  TBLNAME:      <NULL>
  IXNAME:       <NULL>
  PRED:         <NULL>
```

```
  OTHERPRED:   <NULL>

< 3, 3, 3 >
1 row found.
Command> trytbllocks 0;
Command> tryserial 0;
Command> SELECT * FROM T1, t2, t3 WHERE A=B AND B=C AND A=B;

Query Optimizer Plan:

  STEP:        1
  LEVEL:       3
  OPERATION:   TmpTtreeScan
  TBLNAME:     T1
  IXNAME:      <NULL>
  PRED:        <NULL>
  OTHERPRED:   <NULL>

  STEP:        2
  LEVEL:       3
  OPERATION:   TmpTtreeScan
  TBLNAME:     T2
  IXNAME:      <NULL>
  PRED:        T2.B >= T1.A
  OTHERPRED:   <NULL>

  STEP:        3
  LEVEL:       2
  OPERATION:   MergeJoin
  TBLNAME:     <NULL>
  IXNAME:      <NULL>
  PRED:        T1.A = T2.B AND T1.A = T2.B
  OTHERPRED:   <NULL>

  STEP:        4
  LEVEL:       2
  OPERATION:   TmpTtreeScan
  TBLNAME:     T3
  IXNAME:      <NULL>
  PRED:        <NULL>
  OTHERPRED:   T2.B = T3.C


  STEP:        5
  LEVEL:       1
  OPERATION:   NestedLoop
  TBLNAME:     <NULL>
  IXNAME:      <NULL>
  PRED:        <NULL>
```

```
  OTHERPRED:   <NULL>

< 3, 3, 3 >
1 row found.
Command>
```

In this example a query against three tables is executed and the query optimizer plan is displayed. The first version of the query simply uses the query optimizer's default execution plan. However, in the second version the `trytbllocks` and `tryserial` built-in hint commands have been used to alter the query optimizer's plan. Instead of using serial scans and nested loop joins the second version of the query uses temporary index scans and merge joins.

In this way the `showplan` command in conjunction with **ttIsql**'s built-in query optimizer hint commands can be used to quickly determine which execution plan should be used to meet application requirements.

## Timing ODBC function calls

Information on the time required to execute common ODBC function calls can be displayed by using **ttIsql**'s `timing` command. When the timing feature is enabled many built-in **ttIsql** commands will report the elapsed execution time associated with the primary ODBC function call corresponding to the **ttIsql** command that is executed.

For example, when executing **ttIsql**'s `connect` command several ODBC function calls are executed, however, the primary ODBC function call associated with `connect` is **SQLDriverConnect** and this is the function call that is timed and reported as shown below.

**Example 5.13**

```
Command> timing 1;
Command> connect "DSN=MY_DSN";
Connection successful:
DSN=MY_DSN;DataStore=E:\ds\MY_DSN;DRIVER=E:\WINNT\System32\
      TTdv60.dll;
(Default setting AutoCommit=1)
Execution time (SQLDriverConnect) = 1.2626 seconds.
Command>
```

In the example above, the **SQLDriverConnect** call took about 1.25 seconds to execute.

When using the timing command to measure queries, the time required to execute the query plus the time required to fetch the query results is measured. To avoid measuring the time to format and print query results to the display, set the verbosity level to 0 before executing the query.

**Example 5.14**

```
Command> timing 1;
Command> verbosity 0;
Command> SELECT * FROM T1;
Execution time (SQLExecute + FetchLoop) = 0.064210 seconds.
Command>
```

## Working with prepared and parameterized SQL statements

Preparing a SQL statement just once and then executing it multiple times is much more efficient for TimesTen applications than re-preparing the statement each time it is to be executed. **ttIsql** has a set of built-in commands to work with prepared SQL statements. These commands are summarized below:

- **prepare** – Prepares a SQL statement. Corresponds to a **SQLPrepare** ODBC call.
- **exec** – Executes a previously prepared statement. Corresponds to a **SQLExecute** ODBC call.
- **execandfetch** – Executes a previously prepared statement and fetches all result rows. Corresponds to a **SQLExecute** call followed by one or more calls to **SQLFetch**.
- **fetchall** – Fetches all result rows for a previously executed statement. Corresponds to one or more **SQLFetch** calls.
- **fetchone** – Fetches only one row for a previously executed statement. Corresponds to exactly one **SQLFetch** call.
- **close** – Closes the result set cursor on a previously executed statement that generated a result set. Corresponds to a **SQLFreeStmt** call with the SQL_CLOSE option.
- **free** – Closes a previously prepared statement. Corresponds to a **SQLFreeStmt** call with the SQL_DROP option.
- **describe** – Describes the prepared statement including the input parameters and the result columns.

The **ttIsql** utility prepared statement commands also handle SQL statement parameter markers. When parameter markers are included in a prepared SQL statement, **ttIsql** will automatically prompt for the value of each parameter in the statement at execution time.

The example below uses **ttIsql**'s prepared statement commands to prepare an INSERT statement into a table containing an INTEGER and a CHAR column. The statement is prepared and then executed twice with different values for each of the statement's two parameters. **ttIsql**'s `timing` command is used to display the elapsed time required to executed the primary ODBC function call associated with each command.

**Example 5.15**

```
Command> connect "DSN=MY_DSN;Overwrite=1";
Connection successful:
DSN=MY_DSN;DataStore=E:\ds\MY_DSN;Overwrite=1;DRIVER=E:\WINNT\Sys
tem32\TTdv60.dll;
(Default setting AutoCommit=1)
Command> timing 1;
Command> CREATE TABLE T1 (KEY INTEGER NOT NULL PRIMARY KEY, VALUE
CHAR (64));
```

```
Execution time (SQLExecDirect) = 0.1337 seconds.
Command> prepare INSERT INTO T1 VALUES (?,?);
Execution time (SQLPrepare) = 0.0668 seconds.
Command> exec;

All parameter values must be terminated with a semicolon
character.
Type '?;' for help on entering parameter values.
Type '*;' to abort the parameter entry process.

Enter Parameter 1 (INTEGER) >1;
Enter Parameter 2 (CHAR) >'abc';
1 row inserted.
Execution time (SQLExecute) = 0.0757 seconds.
Command> exec;

All parameter values must be terminated with a semicolon
character.
Type '?;' for help on entering parameter values.
Type '*;' to abort the parameter entry process.

Enter Parameter 1 (INTEGER) >2;
Enter Parameter 2 (CHAR) >'def';
1 row inserted.
Execution time (SQLExecute) = 0.0306 seconds.
Command> free;
Command> SELECT * FROM T1;
< 1,
abc
>
< 2,
def
>
2 rows found.
Execution time (SQLExecDirect) = 0.0316 seconds.
Command> disconnect;
Disconnecting...
Execution time (SQLDisconnect) = 1.7091 seconds.
Command>
Command>
```

In the example above, the `prepare` command is immediately followed by the SQL statement to prepare. Whenever a SQL statement is prepared in **ttIsql** a unique command ID is assigned to the prepared statement. **ttIsql** uses this ID to keep track of multiple prepared statements. A maximum of 256 prepared statements can exist in a **ttIsql** session simultaneously. When the `free` command is executed, the command ID is automatically disassociated from the prepared SQL statement.

To see the command IDs generated by **ttIsql** when using the prepared statement commands, set the verbosity level to 4 using the `verbosity` command before preparing the statement, or use the `describe *` command to list all prepared statements with their IDs.

Command IDs can be referenced explicitly when using **ttIsql**'s prepared statement commands. For a complete description of the syntax of **ttIsql**'s prepared statement commands see the *Oracle TimesTen In-Memory Database API and SQL Reference Guide* or type `help` at the **ttIsql** command prompt.

The example below prepares and executes a SELECT statement with a predicate containing one INTEGER parameter. The `fetchone` command is used to fetch the result row generated by the statement. The `showplan` command is used to display the execution plan used by the TimesTen query optimizer when the statement is executed. In addition, the verbosity level is set to 4 so that the command ID used by **ttIsql** to keep track of the prepared statement is displayed.

**Example 5.16**

```
Command> connect "DSN=MY_DSN;Overwrite=1";
Connection successful:
DSN=MY_DSN;DataStore=E:\ds\MY_DSN;Overwrite=1;DRIVER=E:\WINNT\Sys
tem32\TTdv60.dll;
(Default setting AutoCommit=1)
The command succeeded.
Command> CREATE TABLE T1 (KEY INTEGER NOT NULL PRIMARY KEY, VALUE
CHAR (64));
The command succeeded.
Command> INSERT INTO T1 VALUES (1, 'abc');
1 row inserted.
The command succeeded.
Command> autocommit 0;
The command succeeded.
Command> showplan 1;
The command succeeded.
Command> verbosity 4;
The command succeeded.
Command> prepare SELECT * FROM T1 WHERE KEY=?;
Assigning new prepared command id = 0.

Query Optimizer Plan:

  STEP:         1
  LEVEL:        1
  OPERATION:    RowLkHashScan
  TBLNAME:      T1
  IXNAME:       T1
  PRED:         T1.KEY = qmark_1
  OTHERPRED:    <NULL>

The command succeeded.
```

```
Command> exec;
Executing prepared command id = 0.

All parameter values must be terminated with a semicolon
character.
Type '?;' for help on entering parameter values.
Type '*;' to abort the parameter entry process.

Enter Parameter 1 (INTEGER) >1;
The command succeeded.
Command> fetchone;
Fetching prepared command id = 0.
< 1,
abc
>
1 row found.
The command succeeded.
Command> close;
Closing prepared command id = 0.
The command succeeded.
Command> free;
Freeing prepared command id = 0.
The command succeeded.
Command> commit;
The command succeeded.
Command> disconnect;
Disconnecting...
The command succeeded.
Command>
```

## Defining default settings with the TTISQL environment variable

The **ttIsql** utility can be customized to automatically execute a set of command line options every time a **ttIsql** session is started from the command prompt. This is accomplished by setting an environment variable called TTISQL to the value of the **ttIsql** command line that you prefer. A summary of **ttIsql** command line options is shown below. For a complete description of **ttIsql**'s command line options see the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

**Example 5.17**

```
Usage: ttIsql  [-h | -help | -helpcmds | -helpfull | -V]
               [-connStr <connection_string>]
               [-f <filename>]
               [-v <verbosity>]
               [-e <initialization_commands>]
               [-interactive]
               [-N <ncharencoding>]
               [-wait]
```

The TTISQL environment variable has the same syntax requirements as the **ttIsql** command line. When **ttIsql** starts up it reads the value of the TTISQL environment variable and applies all options specified by the variable to the current **ttIsql** session. If a particular command line option is specified in both the TTISQL environment variable and the command line then the command line version will always take precedence.

**Example 5.18**

The procedure for setting the value of an environment variable differs based on the platform and shell that **ttIsql** is started from. As an example, setting the TTISQL environment variable on Windows could look like this:

```
C:\>set TTISQL=-connStr "DSN=MY_DSN" -e "autocommit 0;tables;"
```

**Example 5.19**

In this example, **ttIsql** will automatically connect to a DSN called MY_DSN, turn off autocommit and display all tables in the data store as shown below:

```
C:\>ttIsql

ttIsql (c) 1996-2005, Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.
All commands must end with a semicolon character.

Command> connect "DSN=MY_DSN";
Connection successful:
DSN=MY_DSN;DataStore=E:\ds\MY_DSN;DRIVER=E:\WINNT\System32\TTdv60
.dll;
```

```
(Default setting AutoCommit=1)

Command> autocommit 0;

Command> tables;
  SYS.CACHE_GROUP
  SYS.COLUMNS
  SYS.COL_STATS
  SYS.INDEXES
  SYS.MONITOR
  SYS.PLAN
  SYS.SEQUENCES
  SYS.TABLES
  SYS.TBL_STATS
  SYS.TRANSACTION_LOG_API
  SYS.VIEWS
  TTREP.REPELEMENTS
  TTREP.REPLICATIONS
  TTREP.REPPEERS
  TTREP.REPSTORES
  TTREP.REPSUBSCRIPTIONS
  TTREP.REPTABLES
  TTREP.TTSTORES
16 tables found.
Command>
```

## Managing XLA bookmarks

You can use the **xladeletebookmark** command to both check the status of the current XLA bookmarks and delete them. This command requires ADMIN privilege or object ownership.

For example, when running the XLA application, ‘xlaSimple,’ you can check the bookmark status by entering:

```
Command> xladeletebookmark;

XLA Bookmark: xlaSimple
  Read Log File:  0
  Read Offset:    630000
  Purge Log File: 0
  Purge Offset:   629960
  PID:            2808
  In Use:         No
1 bookmark found.
```

To delete the bookmark, enter:

```
Command> xladeletebookmark xlaSimple;
Command>
```

## Handling Unicode characters

TimesTen supports these national character set configurations:

- Japanese
- Korean
- Western European

By default, the **ttIsql** utility automatically detects the native OS locale settings and processes data accordingly. In addition to the locale-based output, TimesTen supports ASCII and UTF-8 output.

To override the locale-based output format, use the `ncharencoding` option or the `-N` option. Setting `ncharencoding` to ASCII results in Unicode characters being printed in escaped format. To set the output back to the locale-based setting, use the `ncharencoding LOCALE` command. You do not need to have an active connection to change the output method.

---

**Note:** This feature is not supported on Linux platforms.

---

# 6

# *Working with Data in a TimesTen Data Store*

This chapter provides detailed information on the basic components in a TimesTen data store and simple examples of how you can use SQL to manage these components. For more information about SQL, see the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

For information on how to call SQL from within a C or Java application, see "Managing TimesTen Data" in the *Oracle TimesTen In-Memory Database Java Developer's and Reference Guide* or "Managing TimesTen data" in the *Oracle TimesTen In-Memory Database C Developer's and Reference Guide*.

This chapter includes the following topics:

# Data store overview

This section describes the main TimesTen data store elements and features. It includes the following topics:

- Data store components
- Data store users and owners
- Data store persistence

## Data store components

A TimesTen data store has the following permanent components:

- ***Tables***. The primary components of a TimesTen data store are the tables that contain the application data. See "Understanding tables" on page 120.
- ***Materialized Views.*** Read-only tables that hold a summary of data selected from one or more "regular" TimesTen tables. See "Understanding materialized views" on page 124.
- ***Views***. Logical tables that are based on one or more tables called *detail tables*. A view itself contains no data. See "Understanding views" on page 130.
- ***Indexes***. Indexes on one or more columns of a table may be created for faster access to tables. See "Understanding indexes" on page 133.
- ***Rows***. Every table consists of 0 or more rows. A row is a formatted list of values. See "Understanding rows" on page 136.
- ***System tables***. System tables contain TimesTen metadata, such as a table of all tables. See "System tables" on page 122 and the chapter "System and Replication Tables" in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

There are also many temporary components, including prepared commands, cursors and locks.

## Data store users and owners

Unless Access Control is enabled, the TimesTen Data Manager does not authenticate user names. It accepts user names and ignores passwords entirely. TimesTen Client/Server does authenticate users with passwords. Applications should choose one UID for the application itself because by default the login name that is being used to run the application becomes the owner of the data store. If two different logins are used, TimesTen may have difficulty finding the correct tables. If you omit the UID connection attribute in the connection string, TimesTen uses the current user's login name. TimesTen converts all user names to upper case characters.

Users cannot access TimesTen data stores as user SYS. TimesTen determines the user name by the value of the UID connection attribute, or if not present, then by

the login name of the connected user. If a user's login is SYS, set the UID connection to override the login name.

## Data store persistence

When a data store is created, it has either the permanent or temporary attribute set:

- ***Permanent data stores*** are stored to disk automatically through a procedure called checkpointing. TimesTen automatically performs background checkpoints based on the settings of the data store attributes **CkptFrequency** and **CkptLogVolume**. TimesTen also checkpoints the data store when the last application disconnects. Applications can also checkpoint a data store directly to disk by invoking the **ttCkpt** or **ttCkptBlocking** built-in procedures described in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.
- ***Temporary data stores*** are not stored to disk. A temporary data store is automatically destroyed when no applications are connected to it. TimesTen recommends either diskless logging or logging to disk to support transactions and enable row-level locking. TimesTen removes any temporary disk-based files, for example log files, when the last application disconnects. To delete log files that are no longer needed, checkpoint operations should be used with temporary data stores that use logging to disk.

---

**Note:** You cannot change the permanent or temporary attribute on a data store after it is created.

---

# Understanding tables

A TimesTen table consists of rows that have a common format or structure. This format is described by the table's columns. Each column has:

- A data type
- Optional nullability, primary key and foreign key properties
- An optional default value

Unless you explicitly declare a column NOT NULL, columns are nullable. If a column in a table is nullable, it can contain a NULL value. Otherwise, each row in the table must have a non-NULL value in that column.

The format of TimesTen columns cannot be altered. It is possible to add or remove columns but not to change column definitions. To add or remove columns, use the ALTER TABLE statement. To change column definitions, an application must first destroy the table and then recreate it with the new definitions.

The rest of this section includes the following topics:

- In-line and out-of-line columns
- Default column values
- Table names
- Table access
- Primary keys, foreign keys and unique indexes
- System tables

## In-line and out-of-line columns

The in-memory layout of the rows of a table is designed to provide fast access to rows while minimizing wasted space. TimesTen designates each column of a table as either *in-line* or *not inline*.

- An in-line column has a fixed length. All values of fixed-length columns of a table are stored row wise.
- A not inline column has a varying length. Some VARCHAR, NVARCHAR or VARBINARY data type columns are stored not inline. Not inline columns are not stored contiguously with the row but are allocated. Accessing out-of-line columns is slightly slower than accessing in-line columns. By default, columns whose declared column length is > 128 bytes are stored out of line. Columns whose declared column length is <= 128 bytes are stored inlined.

The maximum sizes of in-line and out-of-line portions of a row are listed in "Estimating table size" on page 123.

## Default column values

When you create a table, you can specify default values for the columns. The default value you specify must be compatible with the data type of the column. You can specify one of the following default values for a column:

- NULL
- A constant value
- SYSDATE
- USER
- CURRENT_USER
- SYSTEM_USER

If you use the DEFAULT clause of the CREATE TABLE statement but do not specify the default value, the default value is NULL. See "Column Definition" in *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

## Table names

A TimesTen table is identified uniquely by its owner name and table name. Every table has an owner. By default, the owner is the user who created the table. Tables created by TimesTen, such as system tables, have the owner name `SYS`, or `TTREP` if created during replication.

To uniquely refer to a table, specify both its owner and name separated by a period "." (for example, `MARY.PAYROLL`). If an application does not specify an owner, TimesTen looks for the table under the user name of the caller, then under the user name `SYS`.

A name is an alphanumeric value that begins with a letter (not a digit). A name can include underscores. The maximum length of a table name is 30 characters. The maximum length of an owner name is also 30 characters. TimesTen displays all table, column and owner names to upper case characters. See Chapter 10, "Names" in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide* for additional information.

## Table access

Applications access tables through SQL statements. The TimesTen query optimizer automatically chooses a fast way to access tables. It uses existing indexes or, if necessary, creates temporary indexes to speed up access. For improved performance, applications should explicitly create indexes for frequently searched columns because the automatic creation and destruction of temporary indexes incurs a performance overhead. For more details, see "Tune statements and use indexes" on page 168.

## Primary keys, foreign keys and unique indexes

The creator of a TimesTen table can designate one or more columns as a *primary key* to indicate that duplicate values for that set of columns should be rejected. Primary key columns cannot be nullable. A table can have at most one primary key. TimesTen automatically creates a hash index on the primary key to enforce uniqueness on the primary key and to guarantee fast access through the primary key. Indexes are discussed in "Understanding indexes" on page 133. Once a row is inserted, its primary key columns cannot be modified.

Primary key indexes use hashes. All other indexes use T-trees. Although a table may have only one primary key, additional uniqueness properties may be added to the table using *unique indexes* (see "CREATE INDEX" in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*).

A primary key is optional and an application can achieve a similar effect by creating a table with no primary key columns and then creating a unique index, but it will create a t-tree.

---

**Note:** Columns of a primary key cannot be nullable; a unique index can be built on nullable columns.

---

A table may also have one or more foreign keys through which rows correspond to rows in another table. Foreign keys relate to a primary key in the other table. Foreign keys use a T-tree index on the referencing columns (see "CREATE TABLE" in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*).

## System tables

In addition to tables created by applications, a TimesTen data store contains system tables. System tables contain TimesTen metadata such as descriptions of all tables and indexes in the data store, as well as other information such as optimizer plans. Applications may query system tables just as they query user tables. Applications may not update system tables. TimesTen system tables are described in the chapter "System and Replication Tables" in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

---

**Note:** TimesTen system table formats may change between releases and are different between the 32- and 64-bit versions of TimesTen.

---

# Working with tables

This section includes the following topics:

- Creating a table
- Destroying a table
- Estimating table size

## Creating a table

To create a table, use the SQL statement CREATE TABLE. The syntax for all SQL statements is provided in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*. TimesTen converts table names to upper case characters.

**Example 6.1** The following SQL statement creates a table, called *NameID*, with two columns: *CustId* and *CustName*. The table maps character names to integer identifiers.

```
CREATE TABLE NameID (CustId INTEGER, CustName VARCHAR(50));
```

This example creates a table, called *Customer*, with the columns: *CustId*, *CustName, Addr, Zip,* and *Region.* The *CustId* column is designated as the primary key, so that the *CustId* value in a row uniquely identifies that row in the table, as described in "Primary keys, foreign keys and unique indexes" on page 122. The UNIQUE HASH ON (custId) PAGES value indicates that 30 buckets are to be allocated for the table's hash index.

```
CREATE TABLE Customer
    (custId INT NOT NULL PRIMARY KEY,
     custName CHAR(100) NOT NULL,
     Addr CHAR(100),
     Zip INT,
     Region CHAR(10))
     UNIQUE HASH ON (custId) PAGES = 30;
```

## Destroying a table

To destroy a TimesTen table, call the SQL statement **DROP TABLE**.

**Example 6.2** The following example drops the table *NameID*.

```
DROP TABLE NameID;
```

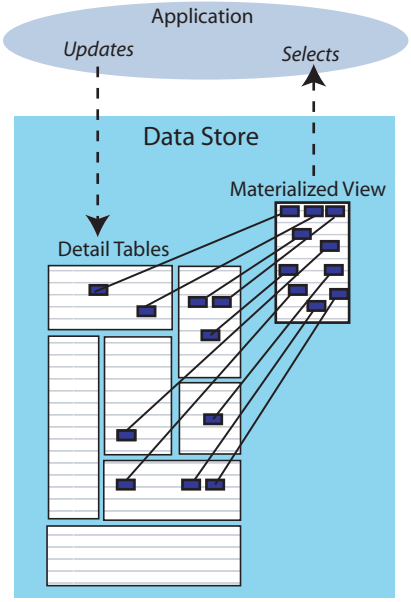
## Estimating table size

Increasing the size of a TimesTen data store can be done on first connect or with an exclusive connection. To avoid having to increase the size of a data store, it is important not to underestimate the eventual data store size. Use the utility **ttSize** to estimate data store size.

# Understanding materialized views

As described in Chapter 8, "Event Notification" of the *Oracle TimesTen In-Memory Database Architectural Overview*, a materialized view is a read-only table that maintains a summary of data selected from one or more "regular" TimesTen tables. The summary data in a materialized view is called a *result set* and the "regular" TimesTen tables queried to make up the result set are called *detail tables*.

**Figure 6.1 Materialized View**



# Working with materialized views

This section includes the following topics:

- Creating a materialized view
- Restrictions on materialized views and detail tables
- Performance implications of materialized views
- Destroying a materialized view

## Creating a materialized view

To create a materialized view, use the SQL statement CREATE MATERIALIZED VIEW. The syntax for all SQL statements is provided in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

Assume the following two tables have been created:

```
CREATE TABLE customer(custId int not null,
       custName char(100) not null,
       Addr char(100),
       Zip int,
       Region char(10),
       PRIMARY KEY (custId));

CREATE TABLE bookOrder(orderId int not null,
       custId int not null,
       ordNo int not null,
       book char(100),
       PRIMARY KEY (orderId),
       FOREIGN KEY (custId) REFERENCES Customer(custId));
```

The following creates a materialized view, named *SampleMV*, that generates a result set from selected columns in the *customer* and *bookOrder* detail tables described above.

```
CREATE MATERIALIZED VIEW SampleMV AS
    SELECT customer.custId, custName, ordNo, book
    FROM customer, bookOrder
    WHERE customer.custId=bookOrder.custId;
```

When creating a materialized view, you can establish primary keys and the size of the hash table in the same manner as described for tables in "Primary keys, foreign keys and unique indexes" on page 122.

### The SELECT query in the CREATE MATERIALIZED VIEW statement

The SELECT query used to define the contents of a materialized view is similar to the top-level SQL SELECT statement described in Chapter 13, "SQL Statements" in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*, with the following restrictions:

- All columns in the GROUP BY *GroupColumnList* must be included in the *SelectList*.
- Aggregate views must include a COUNT(*) in the *SelectList* so that TimesTen can do incremental updates of a group. For example, a group should be removed if its count becomes zero.
- SUM and COUNT are allowed, but not expressions involving them, including AVG.
- The following clauses cannot be used in a materialized view SELECT query:
  - DISTINCT
  - FIRST
  - HAVING
  - ORDER BY
  - FOR UPDATE
- Each expression in the *SelectList* must have a unique name. A name of a simple column expression would be that column's name unless a column alias is defined. ROWID is considered an expression and needs an alias.
- OUTER JOINs are allowed, but the *SelectList* must project at least one non-nullable column from each of the inner tables specified in the OUTER JOIN. Outer join syntax for a SELECT in a materialized view definition is identical to that in a top-level SELECT. The restrictions noted in the description of SELECT statements apply.
- Self joins are allowed. A self join is a join of a table to itself. This table appears twice in the FROM clause and is followed by table aliases that qualify column names in the join condition. Materialized views created with self-join in this release of TimesTen will not work with any prior releases.

## Restrictions on materialized views and detail tables

After a materialized view is created, changes made to the data in the detail tables are immediately reflected in the materialized view. The only way to update a materialized view is by changing the viewed data in the detail tables. A materialized view is a read-only table that cannot be updated directly. This means a materialized view cannot be updated by an INSERT, DELETE, or UPDATE statement by replication, XLA, or the cache agent (it cannot be part of a cache group).

For example, an attempt to update a row in a materialized view generates the error:

```
805: Update view table directly has not been implemented
```

Readers familiar with other implementations of materialized views should note the following characteristics of TimesTen views:

- Detail tables can be replicated, but materialized views cannot.

- Neither a materialized view nor its detail tables can be part of a cache group.
- TimesTen does not automatically create indexes on materialized views or detail tables. No referential indexes can be defined on the materialized view.
- A materialized view cannot be dropped with a DROP TABLE statement. You must use the DROP VIEW statement.
- A materialized view cannot be altered with an ALTER TABLE statement. You must use the DROP VIEW statement and then create a new materialized view with a CREATE MATERIALIZED VIEW statement.
- Materialized views must be explicitly created by the application. The TimesTen query optimizer has no facility to automatically create materialized views.
- The TimesTen query optimizer does not rewrite queries on the detail tables to reference materialized views. Application queries must directly reference views, if they are to be used.
- There is no deferred maintenance option for materialized views. A materialized view is refreshed automatically when changes are committed to its detail tables and there is no facility for manually refreshing a view.
- There are some restrictions to the SQL used to create materialized views. See CREATE MATERIALIZED VIEW in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide* for details.

## Performance implications of materialized views

The performance of UPDATE, INSERT and CREATE VIEW operations may be impacted if the updated table is referenced in a materialized view. The performance impact depends on many factors, such as the nature of the view (how many detail tables, whether outer join or aggregation is used), which indexes are present on the detail table and on the view, and how many view rows will be affected by the change.

A view is a persistent, up-to-date copy of a query result. To keep the view up to date, TimesTen must do "view maintenance" when you change a view's detail table. For example, if you have a view named *V* that selects from tables *T1*, *T2*, and *T3*, then any time you insert into *T1*, or update *T2*, or delete from *T3*, TimesTen does "view maintenance."

View maintenance needs appropriate indexes just like regular database operations. If they're not there, view maintenance will take a very long time.

All of the updates/inserts/deletes on detail tables have execution plans, as described in Chapter 9, "The TimesTen Query Optimizer." For example, an update of a row in *T1* will have a first stage of the plan where it updates the view *V*, followed by a second stage where it updates *T1*.

For view maintenance to be fast, you should look at the plans for all the operations that update (write to) the detail tables and make sure they look good. This may be time-consuming, so here's an easy method to start off with:

1. Look at all the WHERE clauses for the updates/deletes that frequently happen on the detail tables. Those clauses will probably use an index key. Make a note of each of those keys. For example, if the operations that an application does 95% of the time are:

   * update T1 set A=A+1 WHERE K1=? AND K2=?

   * delete from T2 WHERE K3=?

   Then the keys you're remembering are (K1,K2) and K3.

2. Make sure that the view selects all of those key columns. In this example, the view should select K1, K2, and K3.

3. Create an index on the view on each of those keys. In this example, the view should have two indexes, one on (V.K1,V.K2) and one on V.K3. The indexes don't have to be unique, though they can be. (And the names of the view columns can be different from the names of the table columns, though they're the same in this example.)

   With this method, when you update a detail table, your WHERE clause is used to do the corresponding update of the view. This allows maintenance to be done in a batch, and is very fast.

   The above method may not always work, however. For example, an application may have many different ways of updating the detail tables, and so would have to select far too many things in the view, or create too many indexes on the view, taking up more space or more performance than you might like. So, if the above is too cumbersome, an alternative method is:

1. For each table in the view's FROM clause (each detail table), check which ones are frequently changed by UPDATE, INSERT, or CREATE VIEW statements. For example, a view's FROM clause may have tables T1, T2, T3, T4, T5, but of those, only T2 and T3 are frequently changed.

2. For each of those tables, make sure the view selects their rowids. In this example, the view should select T2.rowid and T3.rowid.

3. Create an index on the view on each of those rowid columns. In this example, the columns might be called T2rowid and T3rowid, and indexes would be created on V.T2rowid and V.T3rowid.

   With this method, view maintenance is done on a row-by-row basis, rather than on a batch basis. But the rows can be matched very efficiently between a view and its detail tables. So that speeds up the maintenance. It's generally not as fast as the first method, but it's still pretty good.

## Destroying a materialized view

To destroy a materialized view, call the DROP VIEW SQL statement.

**Example 6.3** The following statement drops the *sampleMV* materialized view, *sampleMV*.

```
DROP VIEW sampleMV;
```

## Understanding views

A *view* is a logical table that is based on one or more tables called *detail tables*. The view itself contains no data. It is sometimes called a *nonmaterialized view* to distinguish it from a materialized view, which does contain data that has already been calculated from the detail tables. Views cannot be updated directly, but changes to the data in the detail tables are immediately reflected in the view.

To choose whether to create a view or a materialized view, consider where the cost of calculation lies. For a materialized view, the cost falls on the users who update the detail tables because calculations must be made to update the data in the materialized views. For a nonmaterialized view, the cost falls on a connection that queries the view, because the calculations must be made at the time of the query.

# Working with views

This section includes the following topics:

- Creating a view
- The SELECT query in the CREATE VIEW statement
- Restrictions on views and their detail tables
- Destroying a view

## Creating a view

To create a view, use the CREATE VIEW SQL statement. The syntax for all SQL statements is provided in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

```
CREATE VIEW ViewName AS SelectQuery;
```

This selects columns from the detail tables to be used in the view. You can also create indexes on the view.

For example, create a view from the table `t1`:

```
CREATE VIEW v1 AS SELECT * FROM t1;
```

Now create a view from an aggregate query on the table `t1`:

```
CREATE VIEW v1 (max1) AS SELECT max(x1) FROM t1;
```

## The SELECT query in the CREATE VIEW statement

The SELECT query used to define the contents of a materialized view is similar to the top-level SQL SELECT statement described in Chapter 13, "SQL Statements" in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*, with the following restrictions:

- A SELECT * query in a view definition is expanded at view creation time. Any columns added after a view is created do not affect the view.
- The following cannot be used in a SELECT statement that is creating a view:
  - DISTINCT
  - FIRST
  - ORDER BY
  - UNION
  - UNION ALL
  - User functions: **USER**, **CURRENT_USER** and **SYSTEM_USER**
- Each expression in the select list must have a unique name. A name of a simple column expression would be that column's name unless a column alias is defined. *RowId* is considered an expression and needs an alias.
- No SELECT FOR UPDATE or SELECT FOR INSERT statements can be used on a view.

- Certain TimesTen query restrictions are not checked when a non-materialized view is created. Views that violate those restrictions may be allowed to be created, but an error is returned when the view is referenced later in an executed statement.

## Restrictions on views and their detail tables

Views have the following restrictions:

- When a view is referenced in the FROM clause of a SELECT statement, its name is replaced by its definition as a derived table at parsing time. If it is not possible to merge all clauses of a view to the same clause in the original select to form a legal query without the derived table, the content of this derived table is **materialized**. For example, if both the view and the referencing select specify aggregates, the view is **materialized** before its result can be joined with other tables of the select.
- A view cannot be dropped with a DROP TABLE statement. You must use the DROP VIEW statement.
- A view cannot be altered with an ALTER TABLE statement.
- Referencing a view can fail due to dropped or altered detail tables.

## Destroying a view

The DROP VIEW statement deletes the specified view, including any hash indexes and any T-tree indexes associated with it.

The following statement drops the `CustOrder` view:

```
DROP VIEW CustOrder;
```

## Understanding indexes

Indexes are auxiliary data structures that speed up table searches. They are used automatically by the query optimizer to speed up the execution of a query. For information about the query optimizer, see Chapter 9, "The TimesTen Query Optimizer."

TimesTen provides two types of indexes to enable faster access to tables: *hash* indexes and *T-tree* (or *range*) indexes.

- ***Hash Indexes***. Hash indexes are useful for finding rows with an exact match on one or more columns. TimesTen currently supports a maximum of one hash index per table. The hash index must be over the primary key of a table. A hash index is created automatically when a table is created with a primary key specified on one or more columns or with the UNIQUE HASH option specified on one or more columns of the table.

  The section "CREATE TABLE" of the *Oracle TimesTen In-Memory Database API and SQL Reference Guide* discusses in detail the automatic creation of hash indexes.

- ***T-tree Indexes***. T-tree indexes are useful for finding rows with column values within a certain range. They are also useful for finding rows with an exact match on one or more columns. T-trees were designed specifically for in-memory applications. T-tree indexes may be created over one or more columns of a table, and up to 32 T-tree indexes may be created on one table.

Hash indexes are useful only for doing equality searches, while T-trees and equijoins can be used for equality and range (greater than/equal to, less than/equal to, etc.) searches. So if you have a primary key on a field and want to see if `FIELD > 10`, the primary key index will not expedite finding the answer, but a separate index will.

---

**Note:** Hash indexes are faster than T-tree indexes for exact match lookups, but they require more space than T-tree indexes. Hash indexes cannot be used for lookups involving ranges.

---

You can designate an index as *unique*, which means that each row in the table has a unique value for the indexed column or columns. Unique indexes can be created over nullable columns. In conformance with the SQL standard, multiple NULL values are permitted in a unique index.

When sorting data values, TimesTen considers NULL values to be larger than all non-NULL values. See the *Oracle TimesTen In-Memory Database API and SQL Reference Guide* for more information on NULL values.

In addition to hash and T-tree indexes, lookups by RowID can be used for fast access to data. RowID lookups are faster than either type of index lookup. See

the *Oracle TimesTen In-Memory Database API and SQL Reference Guide* for more information on RowIDs.

# Working with indexes

This section includes the following topics:

- Creating an index
- Destroying an index
- Estimating index size

## Creating an index

To create an index, call the SQL statement CREATE INDEX. TimesTen converts index names to upper case characters.

Every index has an owner. The owner is the user who created the underlying table. Indexes created by TimesTen itself, such as indexes on system tables, are created with the user name `SYS` or with the user name `TTREP` if created during replication.

**Example 6.4** Create an index *IxID* over column *CustID* of table *NameID*.

```
CREATE INDEX IxID ON NameID (CustID);
```

Currently, it is not possible to create a hash index except by using the primary key or hash clause in the table creation statement. However, TimesTen may create temporary hash and T-tree indexes automatically during query processing to speed up query execution.

## Destroying an index

To uniquely refer to an index, an application must specify both its owner and name. If the application does not specify an owner, TimesTen looks for the index first under the user name of the caller, then under the user name `SYS`.

To destroy a TimesTen index, call the **DROP INDEX** SQL statement. All indexes in a table are destroyed automatically when the table is destroyed.

**Example 6.5** The following drops the index `IxID`.

```
DROP INDEX IxID;
```

## Estimating index size

Increasing the size of a TimesTen data store can be done on first connect or with an exclusive connection. To avoid having to increase the size of a data store, it is important not to underestimate the eventual data store size. Use the utility **ttSize** to estimate data store size.

## Understanding rows

Rows are used to store TimesTen data. TimesTen supports several data types for fields in a row, including:

- One-byte, two-byte, four-byte and eight-byte integers.
- Four-byte and eight-byte floating-point numbers.
- Fixed-length and variable-length character strings, both ASCII and Unicode.
- Fixed-length and variable-length binary data.
- Fixed-length fixed-point numbers.
- Time represented as `hh:mm:ss [AM|am|PM|pm]`.
- Date represented as `yyyy-mm-dd`.
- Timestamp represented as `yyyy-mm-dd hh:mm:ss`.

Chapter 9, "Data Types" in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide* contains a detailed description of these data types.

# Working with rows

This section includes the following topics:

- Inserting rows
- Deleting rows

## Inserting rows

To insert a row, call INSERT or INSERT SELECT. You can also use the **ttBulkCp** utility.

**Example 6.6** To insert a row in the table *NameID*, enter

```
INSERT INTO NameID VALUES(23125, 'John Smith';
```

---

**Note:** When inserting multiple rows into a table, it is more efficient to use prepared commands and parameters in your code. Bulk loads are fastest if done with an exclusive connection (See "ExclAccess" in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*) to the data store. Create indexes after the bulk load is completed.

---

## Deleting rows

To delete a row, call the DELETE statement.

**Example 6.7** The following deletes all the rows from the table `NameID` for names that start with the letter "S."

```
DELETE FROM NameID WHERE CustName LIKE 'S%';
```

# 7

# *Transaction Management and Recovery*

TimesTen supports transactions that provide atomic, consistent, isolated and durable (ACID) access to data.

TimesTen transactions support ANSI Serializable and ANSI Read_Committed levels of isolation. ANSI Serializable isolation is the most stringent transaction isolation level. ANSI Read_Committed allows greater concurrency. Read_Committed is the default and is an appropriate isolation level for most applications. These isolation levels can be combined with each other and with a range of durability options.

TimesTen allows applications to choose the transaction features they need so they do not incur the performance overhead of features they do not need. See Chapter 1, "Data Store Attributes" in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide* for details on how to set isolation levels and durability options.

The main topics in this chapter are:

- Transaction semantics
- Transaction atomicity and durability
- Controlling durability and logging
- Concurrency control
- Checkpoints

## Transaction semantics

TimesTen maintains user-specified levels of isolation, atomicity and durability. As a transaction modifies data in a data store, locking, versioning and logging are used to ensure ACID properties:

- ***Locking.*** TimesTen acquires locks on data items that the transaction reads and/or writes, depending on the transaction isolation level. See "Concurrency control" on page 150.
- ***Logging.*** Modifications to the data store are recorded at user-specified levels in a log. See "Transaction atomicity and durability" on page 143.
- ***Versioning.*** TimesTen makes multiple copies of data items to allow non-serializable read and write operations on those data items to proceed in parallel.

The following table shows how TimesTen uses locks and logs:

| If | Then |
|---|---|
| Transaction is terminated successfully (committed) | • Log is posted to disk (if the DurableCommits attribute is turned on).<br>• Newly modified values of data are made available for other transactions to read and to modify.<br>• Locks that were acquired on behalf of the transaction are released and the corresponding data becomes available to other transactions. |
| Transaction is rolled back | • Log is used to undo the effects of the transaction and to restore any modified data items to the state they were before the transaction began.<br>• Locks that were acquired on behalf of the transaction are released and the corresponding data becomes available to other transactions. |

| If | Then |
|---|---|
| System fails (data not committed) | • On first connect, TimesTen automatically performs data store recovery by reading the latest checkpoint image and applying the log to restore the data store to its latest transactionally consistent state. (See "Checkpoints" on page 153). |
| Application fails | • Transaction is rolled back, if possible.<br>• If rollback is not possible, other connections to data store may be invalidated.<br>• Recovery is done on next connect (See "Recovery" on page 25.) |

TimesTen supports temporary data stores, which have essentially no checkpoints. Recovery is never done for such data stores. They will be destroyed after a data store or application shuts down or fails.

TimesTen supports data stores with no logging. Rollback is not possible for such data stores, and recovery uses only the latest checkpoint. This mode is suitable only for special-purpose operations, such as bulk-loading a data store.

Transactions are started automatically on behalf of an application as needed. Virtually all operations on the data store, even those that do not modify or access application data, require transactional access. For example, compaction and checkpoint operations begin a transaction if one has not already been started. An application can commit a transaction by calling the ODBC **SQLTransact** (*henv*, *hdbc*, SQL_COMMIT) function or JDBC **Connection.commit** method, or abort it by calling the ODBC **SQLTransact** (*henv*, *hdbc*, SQL_ROLLBACK) function or **Connection.rollback** method. Any subsequent data store operation will automatically cause a new transaction to be started.

In compliance with ODBC standards, the default AUTOCOMMIT setting is ON. Commits are costly for performance and can be intrusive if they are implicit.TimesTen recommends that you turn AUTOCOMMIT off so that commits are intentional. Use the ODBC **SQLSetConnectOption** function or JDBC **Connection.setAutoCommit**(false) method in your TimesTen application to set SQL_AUTOCOMMIT_OFF.

When using ODBC or JDBC batch operations to **INSERT**, **UPDATE** or **CREATE VIEW** several rows in one call, when **AUTOCOMMIT** is on, a commit occurs after the entire batch operation has completed. If there is an error during the batch operation, those rows that have been successfully modified will be committed. If an error occurs due to a problem on a particular row, the preceding rows are committed. The **pirow** parameter to the ODBC **SQLParamOptions** function contains the number of the row in the batch that had the problem.

Even with durable commits and autocommit enabled, you could lose work if there is a failure or the application exits without closing cursors. An open cursor under **AUTOCOMMIT** means that you are in effect running with **AUTOCOMMIT** off but without the ability to rollback. Write locks (from DDL or DML) are held until all cursors are closed.

---

**Note:** Autocommit is the default mode for ODBC applications. Applications must explicitly turn autocommit off to avoid it.

---

# Transaction atomicity and durability

The TimesTen Data Manager provides durability with a combination of checkpointing and logging. (See "Checkpoints" on page 153 and "Log files" on page 149.)

Because transaction support adds overhead to execution time, TimesTen allows applications to choose from the following options:

- *Guaranteed atomicity and durability*. (**Logging**=**1**, **DurableCommits**=**1**)
- *Guaranteed atomicity, delayed durability*. (**Logging**=**1**, **DurableCommits**=**0**)
- *Guaranteed atomicity, no guaranteed durability*. (**Logging**=**2**, **DurableCommits**=**0**)
- *No guaranteed atomicity, no guaranteed durability*. (**Logging**=**0**, **DurableCommits**=**0**)

## Overview

The following table summarizes the guarantees and limitations of the atomicity and durability options:

| Logging Attribute Setting | Non-blocking checkpoints possible? | Row-level locking possible ? | Transaction rollback possible? | Recovery procedure | Transactions vulnerable to loss |
|---|---|---|---|---|---|
| Logging = 1<br>Durable Commits = 1 | Yes | Yes | Yes | Read most recent checkpoint image.<br>Apply log. | Uncommitted transactions at the time of failure |
| Logging = 1<br>Durable Commits = 0 | Yes | Yes | Yes | Read most recent checkpoint image.<br>Apply log. | Transactions that committed after the last checkpoint or durable commit |

| Logging Attribute Setting | Non-blocking checkpoints possible? | Row-level locking possible ? | Transaction rollback possible? | Recovery procedure | Transactions vulnerable to loss |
|---|---|---|---|---|---|
| Logging = 2<br>Durable Commits = 0 | No | Yes | Yes | Read most recent checkpoint image. There is no log to apply. | Transactions that committed after the last checkpoint |
| Logging = 0<br>Durable Commits = 0 | No | No | No | Read most recent checkpoint image. There is no log to apply. | Transactions that committed after the last checkpoint |

The sections that follow description these options in greater detail.

## Guaranteed atomicity and durability

**(Logging=1, DurableCommits=1)**

By default, all TimesTen transactions are durable. The effects of the transaction are persistent and will not be lost in the event of system failure.

Durability is implemented with a combination of checkpointing and logging. (See "Checkpoints" on page 153 and "Log files" on page 149.) A checkpoint operation writes the current data store image to a checkpoint file on disk, which has the effect of making all transactions that committed before the checkpoint durable. For transactions that committed after the last checkpoint, TimesTen uses conventional logging techniques to make them durable. As each transaction progresses, it records its data store modifications in an in-memory log. At commit time, the relevant portion of the log is flushed to disk. This log flush operation makes that transaction, and all previously-committed transactions, durable.

In the case of a system failure, recovery uses the last checkpoint image together with the log to reconstruct the latest transaction-consistent state of the data store.

In addition to being durable, by default all TimesTen transactions are also atomic. Either all or none of the effects of the transaction is applied to the data store.

Atomicity is implemented by using the log to undo the effects of a transaction if it is rolled back. Rollback can be caused explicitly by the application, using the

ODBC **SQLTransact** function or JDBC **Connection.rollback** method, or during data store recovery because the transaction was not committed at the time of failure.

In order to have guaranteed atomicity and durability, applications must set the **Logging** and **DurableCommits** attributes to **1**, which is the default for each attribute for permanent data stores.

## Guaranteed atomicity, delayed durability

(**Logging**=**1**, **DurableCommits**=**0**)

It is possible to connect to a data store with logging enabled but with guaranteed durability disabled. In this case, the atomicity of a transaction is guaranteed, but not its durability. This mode is known as delayed durability mode.

In delayed durability mode, as in guaranteed durability mode, each transaction enters records into the in-memory log as it makes modifications to the data store. However, when a transaction commits in delayed durability mode, it does not wait for the log to be posted to disk before returning control to the application. Since the content of the in-memory log would be lost in a system failure, transactions committed in this mode are *not* durable.

Non-durably committed transactions typically have better response time than durably-committed transactions, because no I/O is required to commit the transaction. A non-durably committed transaction can be made durable either by checkpointing (See "Checkpoints" on page 153.) or by committing a subsequent transaction durably. As with guaranteed durability mode, a checkpoint makes all transactions that committed before the checkpoint durable. Committing a transaction durably commits that transaction and all previously committed transactions.

Applications that wish to take advantage of the performance benefits of delayed durability mode but which can only tolerate the loss of a small number of transactions can perform periodic durable commits in a background process — only those transactions that committed non-durably after the last durable commit are vulnerable to loss in the event of a system failure.

Applications request delayed durability mode by setting the **Logging** attribute to **1** (the default) and the **DurableCommits** attribute to **0**. When in this mode, applications can call the **ttDurableCommit** built-in procedure to force the current transaction to commit durably when it commits.

## Guaranteed atomicity, no guaranteed durability

(**Logging**=**2**, **DurableCommits**=**0**)

TimesTen's diskless logging mode can be used for applications needing even greater performance than is offered by delayed durability mode (or which do not

have access to a disk), and which can tolerate even greater potential transaction loss.

In diskless logging (no guaranteed durability) mode, the in-memory log is never written to disk at all. It therefore cannot be used to ensure transaction durability — its use in this mode is limited to enabling transaction atomicity through transaction rollback. In no guaranteed durability mode, the only way to make a transaction durable is by checkpointing the data store. (See "Checkpoints" on page 153.)

In the event of a system failure, recovery reloads the data store from the last checkpoint image, as in the other modes. However, since there are no on-disk logs, the effects of any transactions that committed after the last checkpoint are lost.

Diskless logging must be used with care as it has many limitations

- The in-memory log may grow to exhaust the size of its buffer if there are long-running transactions
- Replication with diskless logging is restricted to temporary data stores
- Nonblocking checkpoints are not available, because recovery from nonblocking checkpoints requires an on-disk log.

For these reasons, we recommend the use of disk logging over diskless logging when possible.

To enable diskless logging mode, applications must set **Logging=2.** In addition, they must set **DurableCommits=0**.

---

**Note:** Logging to disk is highly recommended when row-level locking is enabled. (See "LockLevel." and "Logging." in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.) Diskless logging is provided for systems without disks, or for systems with relatively slow disks where even an occasional disk write has significant impact on performance. When diskless logging is enabled, log records are stored only in the internal log buffer. TimesTen does not write the log buffer to disk. If TimesTen runs out of space in the internal log buffer while transactions are active, TimesTen forces transactions that encounter a full log buffer to abort. An application that chooses to run with diskless logging may be forced periodically to abort some of its transactions.

---

## No guaranteed atomicity, no guaranteed durability

**(Logging=0, DurableCommits=0)**

For the best response time, applications can turn off logging altogether. However, by doing so, they risk both a loss of durability and atomicity. Therefore, the no-logging mode should be used with care. (See "Logging." in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.)

As with diskless logging mode, the only way to make a transaction durable when logging is disabled is to checkpoint the data store. Therefore, a system failure will result in the loss of all transactions that committed after the last checkpoint. In addition, some operations that would fail atomically with logging enabled will fail non-atomically with logging disabled. For example, a SQL UPDATE statement that encounters an error after already having updated one or more rows will not restore the original contents of the updated rows. When this situation arises, a warning is returned indicating that the operation has failed non-atomically.

With logging disabled, transaction rollback is not possible as there is no log of the data store modifications made by the transaction. Therefore all transactions must commit. Attempts to roll a transaction back in no-logging mode will result in an error. Row-level locking is also disabled when logging is disabled. Applications must use data store-level locking.

Because of the many restrictions that exist in non-logging mode, its use should be limited to applications that can be restarted in the event of failure (loading data into a new data store, for example). Otherwise uncommitted data may persist, resulting in inconsistencies.

To use the no-logging mode, applications set **Logging**=**0**, which requires the following additional attribute settings: **LockLevel**=**1**, **DurableCommits**=**0**, and **LogPurge**=**0**.

# Controlling durability and logging

Applications can control whether a transaction is durable and whether log files are created:

- ***Durability***. By default, transactions are durable (DurableCommits=1). To turn off guaranteed durability, turn off the **DurableCommits** connection attribute.

  A running application can override the delayed transaction durability it set for its connection by invoking the TimesTen built-in procedure **ttDurableCommit** to ensure the durability of a specific transaction.

  For a shared data store, durable connections can coexist with connections that are not guaranteed durable.

- ***Logging*.** By default, transaction logs go to disk (Logging=1). To disable logging completely, set the **Logging** connection attribute to 0. To enable diskless logging, set the **Logging** connection attribute to 2. An application can switch between logging to disk, diskless logging, and no logging on the same data store after existing connections have been terminated.

  All concurrent connections to a data store must have the same **Logging** attribute setting. A request for a connection whose logging attributes are incompatible with existing connections is rejected unless **MatchLogOpts** is set.

  When using row-level locking or caching Oracle tables, you must use either logging to disk or diskless logging.

## Using durable commits

The performance cost of durable commits can be mitigated if many threads are running at the same time, due to an effect called "group commit." Under group commit, a single disk write commits a group of concurrent transactions durably. Group commit does not improve the response time of any given commit operation, as each durable commit must wait for a disk write to complete, but it can significantly improve the throughput of a series of concurrent transactions.

When durable commits are used frequently, TimesTen can support more connections than there are CPUs, as long as transactions are short. This is true because each connection spends more time waiting to commit than it spends using the CPU. This is in contrast to applications that do infrequent durable commits, in which case each connection tends to be very CPU-intensive for the TimesTen portion of its workload. In the latter case, using more connections than there are processors will tend to give worse performance, due to CPU contention.

Applications that require lower response time and can tolerate some transaction loss may elect to perform periodic durable commits. By committing only every *n*th transaction durably, or performing a durable commit every *n* seconds (typically in a background process), an application can achieve low response time while maintaining a small window of vulnerability to transaction loss.

Because a durable commit commits not only itself but all previously-committed transactions durably — even those performed by other threads or other processes, an application that commits every *n* transactions durably only risks the loss of the last *n* transactions.

Similarly, an application that performs a durable commit every *n* seconds, risks only the transactions that committed during the last *n* seconds.

To enable periodic durable commits, an application connects with **Logging=1** and **DurableCommits=0**, so transactions commit non-durably by default. When a durable commit is needed, the application calls the **ttDurableCommit** built-in procedure before committing. As with all SQL statements, it is best to pre-prepare the call to **ttDurableCommit** if it will be used frequently. The **ttDurableCommit** built-in procedure does not actually commit the transaction; it merely causes the commit to be durable when it occurs.

Another option for avoiding data loss is to use TimesTen replication instead of durable commits to maintain a copy of the data in two memories. Although two memories are not as durable as disk, replication can provide higher data availability by allowing for failover without data store recovery. This type of trade-off is common in high-performance systems. For more details on TimesTen replication, see the *TimesTen to TimesTen Replication Guide*.

## Log files

If logging to disk is enabled, log files are created in the same directory as the data store files unless the **LogDir** attribute is specified. Multiple log files can exist for a single data store. The log file names have the form *ds_name*`.log`*n*, where *ds_name* is the data store path name given in the data store's DSN and *n* is the log file number, starting at zero.

By default, TimesTen automatically removes archived log files at checkpoint time. To retain archived log files, set the **LogPurge** attribute to 0. When the **LogPurge** attribute is not set for the data store connection, TimesTen renames log files no longer needed to perform recovery to `ds_name.logn.arch`. In this case, the application is responsible for removing these unneeded log files. See "LogPurge" on page 38 for details on purging log files.

When a log file exceeds the value of **LogFileSize** attribute, TimesTen closes that file and begins a new log file.

# Concurrency control

Transaction isolation allows each active transaction to operate as if there were no other transactions active in the system. TimesTen supports two transaction isolation levels: ANSI Serializable and ANSI Read_Committed isolation.

## Transaction isolation levels

- In **ANSI Serializable** isolation, each transaction acquires locks on all data items that it reads or writes. It holds these locks until it commits or rolls back. As a result, a row that has been read by one transaction cannot be updated or deleted by another transaction until the original transaction terminates. Similarly, a row that has been inserted, updated or deleted by one transaction cannot be accessed in any way by another transaction until the original transaction terminates.

  Repeatable reads are assured: a transaction that executes the same query multiple times is guaranteed to see the same result set each time. Other transactions cannot **UPDATE** or **CREATE VIEW** any of the returned rows, nor can they **INSERT** a new row that satisfies the query predicate.

  In this isolation level, readers can block writers and writers can block readers and other writers.

- In **ANSI Read_Committed** isolation, each transaction acquires locks only on the items that it writes. Items read by **SELECT** statements, and the **SELECT** portion of **INSERT SELECT** statements, are not locked. This is the default isolation level for TimesTen.

  In this isolation level, readers do not block writers, nor do writers block readers, even when they read and write the same data items. To allow readers and writers to access the same items without blocking, writers create private copies of the items that they update. These private copies become public when the transaction commits, or are discarded if the transaction rolls back. Therefore, when a transaction reads an item that has been updated by another in-progress transaction, it sees the state of that item before it was updated. It cannot see an uncommitted state.

  Non-repeatable reads are possible in this isolation level. If a Read_Committed transaction executes the same query multiple times, the commit of an updater transaction may cause it to see different results.

All data access in TimesTen uses locking or copying to provide isolation, with the exception of exclusive-mode connections (see "ExclAccess." in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*), which are always isolated since they prevent other connections to the data store. Applications set the transaction isolation level either using the SQLSetConnectOption ODBC function with the SQL_TXN_ISOLATION flag, or by setting the **Isolation** connection attribute.

To ensure that materialized views are always in a consistent state, all view maintenance operations are effectively performed under Serializable isolation, even when the transaction is otherwise in Read_Committed isolation. This means that the transaction will obtain read locks for any data items read during view maintenance. However, the transaction will release these read locks at the end of the **INSERT**, **UPDATE** or **CREATE VIEW** statement that triggered the view maintenance instead of holding them until it terminates.

## Locking granularities

TimesTen supports row-level locking and data store-level locking:

- With row-level locking, transactions usually obtain locks on the individual rows that they access, although a transaction may obtain a lock on an entire table if TimesTen determines that doing so would result in better performance. Row-level locking is the default and is the best choice for most applications, as it provides the finest granularity of concurrency control. Row-level locking requires disk logging or diskless logging. It cannot be used if logging is disabled. Applications can control the circumstances under which TimesTen will elect to use a table lock by setting optimizer flags. (See "ttOptSetFlag."in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.) To use row-level locking, applications must set the **LockLevel** connection attribute to **1** (the default value) or call the **ttLockLevel** built-in procedure with the *lockLevel* parameter set to `Row`.
- With data store-level locking, every transaction obtains an exclusive lock on the entire data store, thus ensuring that there is no more than one active transaction in the data store at any given time. Data store-level locking often provides better performance than row-level locking, due to reduced locking overhead. However, its applicability is limited to those applications that never need to execute multiple concurrent transactions. Data store-level locking is similar to an exclusive connection (See "ExclAccess."in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.) in that both modes of operation prevent concurrent transactions from accessing the data store. However, a shared connection with data store-level locking permits multiple connections to a data store, but allows only one connection to begin a transaction at a time. Exclusive-access, by contrast, prevents multiple connections from being made to the data store. With data store-level locking, every transaction effectively runs in ANSI Serializable isolation, since concurrent transactions are disallowed. Data store level is required when logging is disabled. To use data store-level locking, applications set the **LockLevel** connection attribute to **0** or call the **ttLockLevel** built-in procedure with the *lockLevel* parameter set to set to `DS`.

### Coexistence of different locking levels

Different connections can coexist with different levels of locking, but the presence of even one connection using data store-level locking leads to reduced concurrency. For performance information, see "Choose the best method of locking." in the *Oracle TimesTen In-Memory Database Java Developer's and Reference Guide*. For information on tuning TimesTen C applications, see "Choose the best method of locking." in the *Oracle TimesTen In-Memory Database C Developer's and Reference Guide*.

## Checkpoints

A checkpoint is an operation that saves the state of a data store to disk files, known as checkpoint files. By default, TimesTen performs "background" checkpoints at regular intervals. Alternatively, applications can programmatically initiate checkpoint operations. See "Setting and managing checkpoints" on page 155, for more details.

Each TimesTen data store has two checkpoint files, named *dsname*.ds0 and *dsname*.ds1, where *dsname* is the data store path name specified in the data store's DSN. A checkpoint operation identifies the checkpoint file to which the last checkpoint was written and writes its checkpoint to the other file. Therefore, the two files always contain the two most recent data store images.

Data store recovery uses these files to recover the most recent transaction-consistent state of the data store after a data store shutdown or system failure. It identifies the file that contains the more recent of the two checkpoint images and applies the log to that file's data store image, as appropriate, to recover the up-to-date data store state. If any errors occur during this process, or if the more recent checkpoint image is incomplete (for example, if a system failure occurred while that checkpoint was begin written), then recovery restarts, using the other checkpoint file.

TimesTen also creates a *dsName*.res0 and *dsName*.res1 file for each data store. These files are used internally by TimesTen for the creation of logs.

A checkpoint operation has two primary purposes. First, it decreases the amount of time required for data store recovery, because it provides a more up-to-date data store image for recovery to begin with. Second, it makes a portion of the log unneeded for any future data store recovery operation, typically allowing one or more log files to be deleted. Both of these functions are very important to TimesTen applications. The reduction in recovery time is important, as the amount of log needed to recover a data store has a direct impact on the amount of downtime seen by an application after a system failure. The removal of unneeded log files is important because it frees disk space that can then be used for new log files. If these files were never removed, they would eventually consume all available space in the log directory's file system, causing data store operations to fail due to log space exhaustion.

For these reasons, either TimesTen applications should checkpoint their data stores periodically or you should set the data store first connection attributes **CkptFrequency** and/or **CkptLogVolume**, which determine how often TimesTen performs a "background" checkpoint.

Checkpointing may generate a large amount of I/O activity and have a long execution time depending on the size of the data store and the number of data store changes since the most recent checkpoint.

## Types of checkpoints

TimesTen supports two kinds of data store checkpoints:

### Transaction-consistent checkpoints

Transaction-consistent checkpoints, also known as blocking checkpoints, obtain an exclusive lock on the data store for a portion of the checkpoint, blocking all access to the data store during that time. The resulting checkpoint image contains the effects of all transactions that committed before the checkpointer obtained its lock. Because no transactions can be active while the data store lock is held, no modifications made by in-progress transactions are included in the checkpoint image.

Transaction-consistent checkpoints can be used with any of the three logging modes (disk logging, diskless logging and no logging). When disk logging is in effect, the log will be used during recovery to reapply the effects of transactions that committed durably after the checkpoint completed. To request a transaction-consistent checkpoint, an application uses the **ttCkptBlocking** built-in procedure. The actual checkpoint will be delayed until the requesting transaction commits or rolls back. If a transaction-consistent checkpoint is requested for a data store for which both checkpoint files are already up to date then the checkpoint request will be ignored.

### Fuzzy or non-blocking checkpoints

Fuzzy checkpoints, or non-blocking checkpoints, allow transactions to execute against the data store while the checkpoint is in progress. Fuzzy checkpoints do not obtain locks of any kind, and therefore have a minimal impact on other data store activity. Because these other transactions may modify the data store while it is being written to the checkpoint file, the resulting checkpoint image may contain some effects of transactions that were active while the checkpoint was in progress. Furthermore, different portions of the checkpoint image may reflect different points in time. For example, one portion may have been written before a given transaction committed, while another portion was written afterward. The term “fuzzy checkpoint” derives its name from this fuzzy state of the data store image. TimesTen background checkpoints are always non-blocking.

To recover from a fuzzy checkpoint, TimesTen uses the log both to bring the various portions of the checkpoint into a consistent state with one another and to reapply the effects of transactions that committed durably after the checkpoint completed. For this reason, fuzzy checkpoints can only be used when disk logging is in effect. To request a fuzzy checkpoint, an application uses the **ttCkpt** built-in procedure. If disk logging is in effect, this procedure will issue a fuzzy checkpoint. If diskless logging is in effect, or if logging is disabled, then a blocking (transaction-consistent) checkpoint will be requested. As with all blocking checkpoints, the actual checkpoint will be delayed until the requesting transaction commits or rolls back.

## Setting and managing checkpoints

By default, TimesTen performs automatic non-blocking checkpoints in the background if **Logging**=1. In this case, background checkpoints are non-blocking. (See "Fuzzy or non-blocking checkpoints" on page 154.)

Several data store attributes and built-in procedures are available to set, manage and monitor checkpoints. These include:

- **CkptFrequency** attribute
- **CkptLogVolume** attribute
- **CkptRate** attribute
- **ttCkpt** built-in procedure
- **ttCkptBlocking** built-in procedure
- **ttCkptConfig** built-in procedure
- **ttCkptHistory** built-in procedure

TimesTen also automatically performs a transaction-consistent checkpoint when the last application disconnects from the data store, unless the RAM policy is `always`. For temporary data stores with **Logging**=1, checkpoints are still taken to purge the log files. For non-disk logging Temporary data stores, background checkpointing is off. See "Transaction-consistent checkpoints."

In addition, applications can programatically perform a checkpoint using the **ttCkpt** or **ttCkptBlocking** built in procedure. For details on how to call the **ttCkpt** and other TimesTen built-in procedures from a C or Java program, see "Calling TimesTen built-in procedures within C applications." in the *Oracle TimesTen In-Memory Database C Developer's and Reference Guide* or "Calling TimesTen built-in procedures." in the *Oracle TimesTen In-Memory Database Java Developer's and Reference Guide*.

By default, TimesTen performs background checkpoints at regular intervals. If an application attempts to perform a checkpoint while a background checkpoint is in progress, TimesTen returns an error to the application. To turn off background checkpointing, set **CkptFrequency**=0 and **CkptLogVolume**=0. You can also use the built-in procedure **ttCkptConfig** to configure background checkpointing or turn it off. The values set by **ttCkptConfig** take precedence over those set with the data store attributes.

Using these attributes and the built-in procedure, you can configure TimesTen to checkpoint either when the log files contain a certain amount of data or at a specific frequency. For information on default values and usage, see the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

You need to be aware when background checkpointing is occurring. If the application attempts to back up a data store while a background checkpoint is in process, TimesTen waits until the checkpoint finishes and before beginning the backup. If a background checkpoint starts while a backup is in progress, the

background checkpoint will not take place until the backup has completed. If a background checkpoint starts while an application-initiated checkpoint is in progress, then an error results.

You can use, the **ttCkptHistory** built-in procedure to display the history of last eight checkpoints, the settings for checkpoint frequency and log volume and the status of in-progress checkpoint disk writes.

### Setting the checkpoint rate for background checkpoints

By default, there is no limit to the rate at which checkpoints are written to disk. You can use the **CkptRate** attribute or the **ttCkptConfig** built-in procedure to set the maximum rate at which background checkpoints are written to disk, if you would like to have control over the rate. The rate is expressed in MB per second. Checkpoints taken during recovery and final checkpoints do not honor this rate; their rate is unlimited.

See the *Oracle TimesTen In-Memory Database API and SQL Reference Guide* for details on using these features.

Setting a rate too low can cause checkpoints to take an excessive amount of time and cause the following problems;

- Delay the purging of unneeded log files
- Delay the start of backup operations
- Increase recovery time.

When choosing a rate, you should take into consideration the amount of data written by a typical checkpoint and the amount of time checkpoints usually take. Both of these pieces of information are available through the **ttCkptHistory** built-in procedure.

In addition, you can monitor the progress of a running checkpoint by looking at the *Percent_Complete* column of the **ttCkptHistory** result set. If a running checkpoint appears to be progressing too slowly, the rate can be increased by calling the **ttCkptConfig** built-in procedure. If a call to **ttCkptConfig** changes the rate, the new rate takes effect immediately, affecting even the running checkpoint.

A simple method of calculating the checkpoint rate is:

1. Call the **ttCkptHistory** built-in procedure.
2. For any given checkpoint, subtract the *starttime* from the *endtime*.
3. Divide the number of bytes written by this elapsed time in seconds to get the number of bytes per second.
4. Divide this number by 1024*1024 to get the number of megabytes per second.

When setting the checkpoint rate, some other things to consider are:

- The specified checkpoint rate is only approximate. The actual rate of the checkpoint may be below the specified rate, depending on the hardware, system load and other factors.
- Calculating the actual checkpoint rate using the above method may produce a result that is below the requested rate. This is because the *starttime*/*endtime* interval includes other checkpoint activities in addition to the writing of dirty blocks to the checkpoint file.
- The *Percent_Complete* field of the **ttCkptHistory** call may show 100 percent before the checkpoint is actually complete. The *Percent_Complete* field shows only the progress of the writing of dirty blocks and does not include additional bookkeeping at the end of the checkpoint.
- When adjusting the checkpoint rate, you may also need to adjust the checkpoint frequency, as a slower rate makes checkpoints take longer, which effectively increases the minimum time between checkpoint beginnings.

# 8

# *Data Store Performance Tuning*

An application using the TimesTen Data Manager should obtain an order of magnitude performance improvement in its data access over an application using a traditional DBMS. However, poor application design and tuning can erode the TimesTen advantage. This chapter discusses factors that can affect the performance of a TimesTen application. These factors range from subtle, such as data conversions, to more overt, such as preparing a command at each execution.

This chapter explains the full range of these factors, with a section on each factor indicating:

- How the problem can be detected
- Whether the potential performance impact is large, medium, small or variable.
- Where the performance gain may be
- What the tradeoffs are

As discussed throughout this chapter, many performance problems can be identified by examining the **SYS.MONITOR** table.

Topics are:

- System and data store tuning
- Client/server tuning
- SQL tuning
- Improving performance of materialized views
- Scaling to Multiple CPUs
- XLA acknowledgement modes

For information on tuning TimesTen Java applications, see Chapter 5, "Application Tuning in the *Oracle TimesTen In-Memory Database Java Developer's and Reference Guide*. For information on tuning TimesTen C applications, see Chapter 5, "Application Tuning in the *Oracle TimesTen In-Memory Database C Developer's and Reference Guide*.

# System and data store tuning

**Performance impact: Large**

## Provide enough memory

Configure your system so that the entire data store fits in main memory. The use of virtual memory substantially decreases performance. You will know that the data store (or working set) does not fit if a performance monitoring tool shows excessive paging or virtual memory activity.

You may have to add physical memory or configure the system software to allow a large amount of shared memory to be allocated to your process(es). TimesTen includes the **ttSize** utility to help you estimate the size of your data store.

**Performance impact: Variable**

## Size your data store correctly

When you create a data store, you are required to specify a data store size. Specifically, you specify a size for the permanent partition of the data store and a size for the temporary partition of the data store.

Once you have created a data store, you can increase the size of the permanent partition of the data store and you can either increase or decrease the temporary partition of the data store. See "Specifying the size of a data store" on page 46.

To make size estimates, use the **ttSize** utility or run the application until you can make a reasonable estimate.

Another method for estimating size requirements is to use the **SYS.MONITOR** table. The columns PERM_ALLOCATED_SIZE, TEMP_ALLOCATED_SIZE, PERM_IN_USE_SIZE and TEMP_IN_USE_SIZE show (in KB units) the currently allocated size of the data store, and the in-use size of the data store. The system updates this information each time a connection is made or released and each time a transaction is committed or rolled back.

You can monitor block-level fragmentation in the data store by using **ttBlockInfo**.

**Performance impact: Large**

## Use multi-processor optimizations if appropriate

Applications running on single processor machines should set **SMPOptLevel** to 0. If an application running on a single processor machine sets **SMPOptLevel** to greater than 0, it will experience a drop in performance due to the increased use of synchronization primitives.

**Performance impact: Large**

## Increase LogBuffSize if needed

Increasing the value of **LogBuffSize** can have a substantial positive performance impact. If LOG_BUFFER_WAITS is increasing, then increase the value of **LogBuffSize**.

The trade-off is that more transactions are buffered in memory and may be lost if the process crashes. If **DurableCommits** are enabled, increasing the default **LogBuffSize** value does not improve performance.

**Performance impact: Variable**

## Use temporary data stores if appropriate

A TimesTen data store may be permanent or **Temporary**. A temporary data store disappears when the last connection goes away or when there is a system or application failure.

If you do not need to save the data store to disk, you can save checkpoint overhead by creating a temporary data store.

Temporary data stores are never fully checkpointed to disk, although the log is periodically written to disk when logging to disk is enabled. Checkpoint operations can have significant overhead for permanent data stores, depending on data store size and activity, but have very little impact for temporary data stores. Checkpoints are still necessary to remove log files.

**Performance impact: Large**

## Avoid connection overhead

By default, TimesTen loads an idle data store (a data store with no connections) into memory when a first connection is made to it. When the final application disconnects from a data store, a delay occurs when the data store is written to disk. If applications are continually connecting and disconnecting from a data store, the data store may be loaded and unloaded to/from memory continuously, resulting in excessive disk I/O and poor performance. Similarly, if you are using a very large data store you may want to pre-load the data store into memory before any applications attempt to use it.

To avoid the latency of loading a data store into memory, you can change the RAM policy of the data store to allow data stores to always remain in memory. The trade-off is that since the data store is never unloaded from memory, a final disconnect checkpoint never occurs. So, applications should checkpoint the data store explicitly in order to reduce the disk space taken up by log files.

Alternatively, you can specify that the data store remain in memory for a specified interval of time and accept new connections. If no new connections occur in this interval, TimesTen unloads the data store from memory and checkpoints it. You can also specify a setting to allow a system administrator to load and unload the data store from memory manually.

To change the RAM policy of a data store, use the **ttAdmin** utility.

**Perfect impact: large**

## Load the data store into RAM when duplicating

When you duplicate a data store, use the `-ramLoad` option of the **ttAdmin** utility. This places the data store in memory, available for connections, instead of unloading it with a blocking checkpoint. See "Avoid connection overhead".

**Performance impact: Medium**

## Avoid OS paging at load time

All of the TimesTen platform operating systems implement a dynamic file system buffer pool in main memory. If this buffer pool is allowed to be large, TimesTen and the operating system both retain a copy of the file in memory, causing some of the TimesTen shared segment to be paged out.

This behavior may not occur for data stores that are less than half of the installed memory size. On some systems, it is possible to limit the amount of main memory used by the file system. On other systems, this effect is less pronounced. On HP-UX, Solaris and Linux systems, consider using the **MemoryLock** attribute to specify whether the data store should be locked in memory. If used, the data store cannot be paged out.

On HP-UX, consider the settings for the kernel parameters `dbc_min_pct` and `dbc_max_pct`. These parameters control the minimum and maximum percent of real memory devoted to the file system. The default maximum is 50 percent. TimesTen recommends reducing the maximum to 10 percent.

**Performance impact: Medium**

## Consider special options for maintenance

During special operations such as initial loading, you can choose different options than you would use during normal operations. In particular, consider turning **Logging** off and using data store-level locking for bulk loading; an example would be using **ttBulkCp** or **ttMigrate**. These choices can improve loading performance by a factor of two.

**Performance impact: large**

## Check your driver

There are two versions of the TimesTen Data Manager driver for each platform, a debugging version and a production version. Unless you are actually debugging, use the production version. The debugging library can be significantly slower. See "Specify the DSN" on page 40 and "Specify the ODBC driver" on page 37 for a description of the TimesTen Data Manager drivers for the different platforms.

WINDOWS

On Windows, make sure that applications that use the ODBC driver manager use a DSN that accesses the correct TimesTen driver. Make sure that direct-linked applications are linked with the correct TimesTen driver. An application can call the ODBC **SQLGetInfo** function with the **SQL_DRIVER_NAME** argument to determine which driver it is using.

**Performance impact: Large**

## Enable tracing only as needed

Both ODBC and JDBC provide a trace facility to help debug applications. For performance runs, make sure that tracing is disabled except when debugging applications.

To turn the JDBC tracing on, use:

```
DriverManager.setLogStream method:
DriverManager.setLogStream(new PrintStream(System.out, true));
```

By default tracing is off. You must call this method before you load a JDBC driver. Once you turn the tracing on, you cannot turn it off for the entire execution of the application.

**Performance impact: Variable**

## Investigate alternative JVMs

JRockit, IBM and HP provide JVMs that may perform better than the Sun JVM.

**Performance impact: Large**

## Adjust log buffer size and CPU for a large number of subscribers

If you are planning a replication scheme that includes a large number of subscribers, then ensure the following:

- The setting for **LogBuffSize** should result in the value of LOG_FS_READS in the SYS.MONITOR table being 0 or close to 0. This ensures that the replication agent does not have to read any log records from disk. If the value of LOG_FS_READS is increasing, then increase the log buffer size.
- CPU resources are adequate. The replication agent on the master data store will spawn a thread for every subscriber data store. Each thread reads and processes the log independently and needs adequate CPU resources to make progress.

# Client/server tuning

**Performance impact: Large**

## Work locally when possible

Using TimesTen Client to access data stores on a remote server machine can add network overhead to your connections. Whenever possible, write your applications to access the TimesTen Data Manager locally, and link the application directly with the TimesTen Data Manager.

**Performance impact: Variable**

## Use shared memory segment as IPC when client and server are on the same machine

The TimesTen Client normally communicates with TimesTen Server using TCP/IP sockets. If both the TimesTen Client and TimesTen Server are on the same machine, client applications show improved performance by using a shared memory segment or a UNIX domain socket for inter-process communication (IPC).

To use a shared memory segment as IPC, you must set the server options in the ttendaemon.options file. For a description of the server options, see "Modifying the TimesTen web server options" in Chapter 4 of the *Oracle TimesTen In-Memory Database Operations Guide*.

In addition, applications that use shared memory for IPC must use a logical server name (for the Client DSN) with `ttShmHost` as the Network Address. For more information, see "Creating and configuring Client DSNs on UNIX" on page 67.

This feature may require a significant amount of shared memory. The TimesTen Server pre-allocates the shared memory segment irrespective of the number of existing connections or the number of statements within all connections.


If your application is running on a UNIX machine and you are concerned about memory usage, the applications using TimesTen Client ODBC driver may improve the performance by using UNIX domain sockets for communication. The performance improvement when using UNIX domain sockets is not as large as when using ShmIPC.

Applications that take advantage of UNIX domain sockets for local connections must use a logical server name (for the Client DSN) with `ttLocalHost` as the Network Address. For more information, see "Creating and configuring Client DSNs on UNIX" on page 67. In addition, make sure that your system kernel parameters are configured to allow the number of connections you require. See "Installation prerequisites" in the *Oracle TimesTen In-Memory Database Installation Guide*.

## Enable/Disable TT_PREFETCH_CLOSE for SELECT queries

**Performance impact: Variable**

With the TimesTen connection option TT_PREFETCH_CLOSE, an application using TimesTen Client Driver can have better performance for certain SELECT queries. This is an ODBC statement option that can be set with the **SQLSetStatementOption**. In order to use the option, the application must explicitly turn ON or OFF. The option is designed to remove network I/O for closing a cursor and committing.

TimesTen Server already pre-fetches a full or partial result set from the data store during the execution of SQLExecute and SQLExecDirect. TimesTen buffers this data at the TimesTen Client. This is done to reduce network I/O between the Client and the Server. With AUTOCOMMIT off, the Server also closes the cursor if the end of the result set is reached.

For read-only transactions when TT_PREFETCH_CLOSE is on, the Server commits the transaction if AUTOCOMMIT is off. If AUTOCOMMIT is on, the Server closes the cursor and commits the transaction.

---

**Note:** The TT_PREFETCH_CLOSE option is only useful for Client applications. If used in a TimesTen Data Manager application, TimesTen ignores it.

---

Normally, an application should turn on the TT_PREFETCH_CLOSE option when all of the following criteria are satisfied:

- No other cursor is open.
- The transaction is a read only transaction.
- The query is not a SELECT…….FOR UPDATE.

Once the application has committed such a transaction, the application should turn off the option unless the next transaction of the application also desires to use this feature.

---

**Note:** This option should be used cautiously. If a transaction has an INSERT, UPDATE or DELETE statement while this option is set, the SERVER may commit those changes although the application never issued a commit.

---

The following examples show how to use the TT_PREFETCH_CLOSE option with JDBC and ODBC.

---

**Example 8.1** To use the TT_PREFETCH_CLOSE option with JDBC:

```
SQLSetConnectOption (hdbc, TT_PREFETCH_CLOSE,
     TT_PREFETCH_CLOSE_ON);
SQLExecDirect (hstmt, "SELECT * FROM T", SQL_NTS);
```

```
while (SQLFetch (hstmt) != SQL_NO_DATA_FOUND)
{
// do the processing
}
SQLFreeStmt (hstmt, SQL_CLOSE);
SQLTransact (henv, hdbc, SQL_COMMIT);
SQLSetConnectOption (hdbc, TT_PREFETCH_CLOSE,
     TT_PREFETCH_CLOSE_OFF);
```

**Example 8.2** To use the TT_PREFETCH_CLOSE option with ODBC:

```
con = DriverManager.getConnection ("jdbc:timesten:client:" + DSN);
stmt = con.createStatement();

import com.timesten.sql
……..
……..
con.setTtPrefetchClose(boolean);
rs = stmt.executeQuery("select * from t");
while(rs.next())
{
 // do the processing
}

import com.timesten.sql
....
try {
       ((TimesTenConnection)con).setTtPrefetchClose(true);
}
catch (SQLException) {
...
}

rs.close();
con.commit();
con.setTtPreFetchClose (false);
```

**Performance impact: Large**

## Use a connection handle when calling SQLTransact

An application can make a call to SQLTransact with either SQL_NULL_HDBC and a valid environment handle:

```
SQLTransact (ValidHENV, SQL_NULL_HDBC, fType)
```

or a valid connection handle:

`SQLTransact (SQL_NULL_HENV, ValidHDBC, fType)`.

When using a valid environment handle, the TimesTen driver tries to commit/rollback transactions on all connection handles that are in a connected state. In Client/Server mode, this causes at least *n* network I/O operations, where *n* is the number of connection handles that are in a connected state. Normally, network I/O operations are much more expensive than TimesTen database operations.

Hence, if the intention of the application is simply to commit/rollback on a single connection, it should use a valid connection handle when calling SQLTransact.

# SQL tuning

After you have determined the overall logging, locking and I/O strategies, make sure that the individual SQL statements are executed as efficiently as possible.

**Performance impact: Large**

## Tune statements and use indexes

Verify that all statements are executed efficiently. For example, use queries that reference only the rows necessary to produce the required result set. If only col1 from table t1 is needed, use the statement:

```
SELECT col1 FROM t1...
```

instead of using:

```
SELECT * FROM t1...
```

Chapter 9, "The TimesTen Query Optimizer," describes how to view the plan that TimesTen uses to execute a statement. Alternatively, you can use the `ttIsql showplan` command to view the plan. View the plan for each frequently executed statement in the application. If indexes are not used to evaluate predicates, consider creating new indexes or rewriting the statement or query so that indexes can be used. For example, indexes can only be used to evaluate WHERE clauses when single columns appear on one side of a comparison predicate (equalities and inequalities), or in a BETWEEN predicate.

If this comparison predicate is evaluated often, it would therefore make sense to rewrite

```
WHERE c1+10 < c2+20
```

to

```
WHERE c1 < c2+10
```

and create an index on `c1`.

The presence of indexes does slow down write operations such as UPDATE, INSERT, DELETE and CREATE VIEW. If an application does few reads but many writes to a table, an index on that table may hurt overall performance rather than help it.

The FIRST keyword can be used to operate on a specific number of rows in the SQL statements, SELECT, UPDATE and DELETE. This attribute can improve throughput and response time. Alternatively, if an application plans to fetch at most one row for a query, and a unique index isn't being used to fetch the row, the application should set SQL_MAX_ROW_COUNT to 1. See the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

Occasionally, the system may create a temporary index to speed up query evaluation. If this happens frequently, it is better for the application itself to create the index. The CMD_TEMP_INDEXES column in the MONITOR table indicates how often a temporary index was created during query evaluation.

**Performance impact: Variable**

## Select hash or T-tree indexes appropriately

The TimesTen Data Manager supports both hash and T-tree indexes. Hash indexes are created for tables that declare a PRIMARY KEY. T-tree indexes are created with the CREATE INDEX statement.

The index structures have different strengths. Hash indexes are faster than T-tree indexes for exact key lookups on all columns of the primary key. However, hash indexes have the following restrictions:

- Hash indexes do not speed up queries on LESS THAN or GREATER THAN comparisons on indexed columns.
- All columns must appear in the WHERE clause.
- Hash indexes can only be created on the columns of the PRIMARY KEY; these columns cannot be set to NULL, cannot be changed once a record is inserted and only allow unique values of the indexed columns to be inserted.

T-tree indexes can also speed up exact key lookups but are more flexible and can speed up other queries as well. Select a T-tree index if your queries include LESS THAN or GREATER THAN comparisons. T-tree indexes can also be used to speed up "prefix" queries. A prefix query has equality conditions on all but the last key column that is specified. The last column of a prefix query can have either an equality condition or an inequality condition.

Consider the following table and index definitions:

```
CREATE TABLE T(i1 integer, i2 integer, i3 integer, ...);
CREATE INDEX IXT on T(i1, i2, i3);
```

The index IXT can be used to speed up the following queries:

```
SELECT * FROM T WHERE i1>12;
SELECT * FROM T WHERE i1=12 and i2=75;
SELECT * FROM T WHERE i1=12 and i2 BETWEEN 10 and 20;
SELECT * FROM T WHERE i1=12 and i2=75 and i3>30;
```

The index IXT will not be used for queries like:

```
SELECT * FROM T WHERE i2=12;
```

because the prefix property is not satisfied. There is no equality condition for `i1`.

The index IXT will be used, but matching will only occur on the first two columns for queries like:

```
SELECT * FROM T WHERE i1=12 and i2<50 and i3=630;
```

T-tree indexes have a dynamic structure that adjusts itself automatically to accommodate changes in table size. A T-tree index can be either unique or non-unique and can be declared over nullable columns. It also allows the indexed column values to be changed once a record is inserted. A T-tree index is likely to be more compact than an equivalent hash index.

## Size hash indexes appropriately

**Performance impact: Variable**

TimesTen uses hash indexes to enforce primary key constraints. The number of buckets used for the hash index is determined by the PAGES parameter specified in the UNIQUE HASH ON clause of the CREATE TABLE statement. The value for PAGES should be the expected number of rows in the table divided by 256. A smaller value may result in a greater number of collisions, decreasing performance, while a larger value may provide somewhat increased performance at the cost of extra space used by the index.

If the number of values to be indexed varies dramatically, it is best to err on the side of a large index. If the size of a table cannot be accurately predicted, consider using a T-tree index with CREATE INDEX. Also, consider the use of unique indexes when the indexed columns are large CHAR or binary values or when many columns are indexed. Unique indexes may be faster than hash indexes in these cases.

If the performance of record inserts degrades as the size of the table gets larger, it is very likely that you have underestimated the expected size of the table. You can resize the hash index by using the ALTER TABLE statement to reset the PAGES value in the UNIQUE HASH ON clause.

## Use foreign key constraint appropriately

**Performance impact: Variable**

The declaration of a foreign key has no performance impact on SELECT queries, but it slows down the INSERT and UPDATE operations on the table that the foreign key is defined on and the UPDATE and DELETE operations on the table referenced by the foreign key. The slow down is proportional to the number of foreign keys that either reference or are defined on the table.

## Computing exact or estimated statistics

**Performance impact: Variable**

If statistics are available on the data in the data store, the TimesTen optimizer uses them when preparing a command to determine the optimal path to the data. If there are no statistics, the optimizer uses generic guesses about the data distribution. If you have examined the plans generated for your statements (see Chapter 9, "The TimesTen Query Optimizer") and you think they may not be optimal, consider computing statistics before preparing your statements and re-examining the plans.

If you haven't examined the plans, we generally recommend computing statistics since the information is likely to result in more efficient plans.

There are two built-in procedures for computing statistics: **ttOptUpdateStats** and **ttOptEstimateStats**. **ttOptUpdateStats** looks at every row of the table(s) in question and computes exact statistics. **ttOptEstimateStats** looks at only a sampling of the rows of the table(s) in question and produces estimated statistics. Estimating statistics can be considerably faster but can also result in less accurate

statistics. In general, if time is not an issue, it's best to call **ttOptUpdateStats**. But estimation is preferable if overall application performance may be affected. As a rule of thumb, computing statistics with a sample of 10 percent is about ten times faster than computing exact statistics and generally results in the same execution plans. Since computing statistics is a time-consuming operation, it's best to compute statistics once after loading your data store (and before preparing commands), and then periodically only if the composition of your data changes substantially.

**Performance impact: Variable**

## Avoid ALTER TABLE

The ALTER TABLE statement allows applications to add columns to a table and to drop columns from a table. Although the ALTER TABLE statement itself runs very quickly in most cases, the modifications it makes to the table can cause subsequent operations on the table to run more slowly. The actual performance degradation the application experiences varies with the number of times the table has been altered and with the particular operation being performed on the table.

Dropping VARCHAR and VARBINARY columns is slower than dropping columns of other data types since a table scan is required to free the space allocated to the existing VARCHAR and VARBINARY values in the column to be dropped.

**Performance impact: Variable**

## Avoid nested queries

TimesTen supports nested queries with some limitations. However, performance varies and is generally not optimal. See the *Oracle TimesTen In-Memory Database API and SQL Reference Guide* for details on subqueries.

# Improving performance of materialized views

**Performance impact: Variable**

## Limit number of join rows

Larger numbers of join rows decrease performance. You can limit the number of join rows by controlling the join condition. For example, use only equality conditions that map one row from one table to one or at most a few rows from the other table.

**Performance impact: Variable**

## Use indexes on join columns

Create indexes on the columns of the detail table that are specified in the SELECT statement that creates the join. Also consider creating an index on the materialized view itself. This can improve the performance of keeping the materialized view updated.

If an UPDATE or DELETE operation on a detail table is often based on a condition on a column, try to create an index on the materialized view on this column if possible.

For example, `CustOrder` is a materialized view of customer orders, based on two tables. The tables are `Customer` and `bookOrder`. The former has two columns (`custNo` and `custName`) and the latter has three columns (`ordNo`, `book`, and `custNo`). If you often update the `bookOrder` table to change a particular order by using the condition `bookOrder.ordNo=const`, then create an index on `CustOrder.ordNo`. On the other hand, if you often update based on the condition `bookOrder.custNo=const`, then create an index on `CustOrder.custNo`.

If you often UPDATE using both conditions and cannot afford to create both indexes, you may want to add `bookOrder.rowId` in the view and create an index on it instead. In this case, TimesTen updates the view for each detail row update instead of updating all of the rows in the view directly and at the same time. The scan to find the row to be updated is an index scan instead of a row scan, and no join rows need to be generated.

If ViewUniqueMatchScan is used in the execution plan, it is a sign that the execution may be slower or require more space than necessary. A ViewUniqueMatchScan is used to handle an update or delete that cannot be translated to a direct update or delete of a materialized view, and there is no unique mapping between a join row and the associated row in the materialized view. This can be fixed by selecting a unique key for each detail table that is updated or deleted.

**Performance impact: Variable**

## Avoid unnecessary updates

Try not to update a join column or a “group by” column because this involves deleting the old value and inserting the new value.

Try not to update an expression that references more than one table. This may disallow direct update of the view because TimesTen may perform another join operation to get the new value when one value in this expression is updated.

View maintenance based on an update or delete is more expensive when:

- The view cannot be updated directly. For example, not all columns specified in the detail table UPDATE or DELETE statement are selected in the view, or
- There is not an indication of a one-to-one mapping from the view rows to the join rows.

For example:

```
CREATE MATERIALIZED VIEW v1 AS SELECT x1 FROM t1, t2 WHERE x1=x2;
DELETE FROM t1 WHERE y1=1;
```

The extra cost comes from the fact that extra processing is needed to ensure that one and only one view row is affected due to a join row.

The problem is resolved if either `x1` is UNIQUE or a unique key from `t1` is included in the select list of the view. ROWID can always be used as the unique key.

**Performance impact: Variable**

## Avoid changes to the inner table of an outer join

Since outer join maintenance is more expensive when changes happen to an inner table, try to avoid changes to the inner table of an outer join. When possible, perform INSERT operations on an inner table before inserting into the associated join rows into an outer table. Likewise, when possible perform DELETE operations on the outer table before deleting from the inner table. This avoids having to convert non-matching rows into matching rows or vice versa.

**Performance impact: Variable**

## Limit number of columns in a view table

The number of columns projected in the view SelectList can impact performance. As the number of columns in the select list grows, the time to prepare operations on detail tables increases. In addition, the time to execute operations on the view detail tables also increases. Do not select values or expressions that are not needed.

The optimizer considers the use of temporary indexes when preparing operations on detail tables of views. This can significantly slow down prepare time, depending upon the operation and the view. If prepare time seems slow, consider using ttOptSetFlag to turn off temporary ttree indexes and temporary hash scans.

# Scaling to Multiple CPUs

**Performance impact: Variable**

## Run the demo applications as a prototype

One way to see what sort of scaling you can expect from TimesTen is to run one of the scalable demo applications, such as `tptbm`, on your system.

The tptbm application has a parameter `-proc` that lets you vary the number of processes that execute TimesTen operations. In addition, it has other parameters that let you specify the transaction mix of READs, WRITEs and INSERTs.

Run `tptbm -help` to see the full list of options. For example, you can specify whether it uses a hash index or a t-tree index.

By default the demo does one operation per transaction. You can use the `-ops` options to add more operations in a transaction to better model your application. Larger transactions may scale better or worse, depending on the application profile.

Run multi-processor versions of the demo to evaluate how your application can be expected to perform on systems that have multiple CPUs. If the demo scales well, but your application scales poorly, you might try simplifying your application to see where the issue is. Some users "stub out" the TimesTen calls and find they still have bad scaling, and they discover an issue in their application.

You may find that some simulated application data is not being generated properly, so all the operations are accessing the same few rows. That type of localized access will greatly inhibit scalability if the accesses involve changes to the data.

**Performance impact: Variable**

## Limit database-intensive connections per CPU

Check the LOCK_TIMEOUTS or LOCK_GRANTS_WAIT fields in the SYS.MONITOR table. If they have high values, this may indicate undue contention, which can lead to poor scaling.

Because TimesTen is quite CPU-intensive, optimal scaling is achieved by having at most one database-intensive connection per CPU. If you have a 4-CPU system or a 2-CPU system with hyperthreading, then a 4-processor application will run well, but an 8-processor application will not perform well. The contention between the active threads will be too high. The only exception to this rule is when many transactions are committed durably. In this case, the connections are not very CPU-intensive because of the increase in I/O operations to disk, and so the machine can support many more concurrent connections. We have seen up to 60 connections on a 4 CPU system.

**Performance impact: Variable**

## Use read operations when available

Read operations scale better than write operations. Make sure that the read/write balance reflects the real-life workload of your application.

**Performance impact: Variable**

## Limit prepares, re-prepares and connects

Prepares do not scale. Make sure that you pre-prepare commands that are executed more than once. The CMD_PREPARES and CMD_REPREPARES columns of the SYS.MONITOR table indicate how often commands were prepared or automatically re-prepared (due to creation or deletion of indexes). If either has a high value, modify your application to do connection pooling, so that connects and disconnects are rare events.

Connects do not scale. Make sure that you pre-prepare commands that are executed more than once. Look at the DS_CONNECTS field in the SYS.MONITOR table. If the field has a high value, modify your application to do connection pooling, so that connects and disconnects are rare events.

If your application uses more than 64 active connections, you may wan to set the **CkptRate** attribute to the number of active connections.

**Performance impact: Variable**

## Limit replication transmitters and receivers and XLA readers

Replication and XLA operations have significant logging overhead. Replication scales best when there are a limited number of transmitters and/or receivers. Check your replication topology and see if you can simplify it. XLA scales best when there are a limited number of readers. If your application has numerous readers, see if you can reduce the number.

Monitor XLA and replication to ensure they are reading from the log buffer rather than from the disk. With a lot of concurrent updates, replication may not keep up. Updates are single-threaded at the subscriber. You can achieve better XLA throughput if the frequency of acknowledgements is reduced.

Estimate the number of readers and transmitters required by checking the values in the LOG_FS_READS and LOG_BUFFER_WAITS columns in the **SYS.MONITOR** table. The system updates this information each time a connection is made or released and each time a transaction is committed or rolled back.

Setting **LogFlushMethod=2** can improve performance of RETURN TWOSAFE replication operations and RETURN RECEIPT with DURABLE TRANSMIT operations.

**Performance impact: variable**

## Allow indexes to be rebuilt in parallel during recovery

On multi-processor systems, set **RecoveryThreads** to minimum(number of CPUs available, number of indexes) to allow indexes to be rebuilt in parallel if

recovery is necessary. If a rebuild is necessary, progress can be viewed in the daemon log. Setting **RecoveryThreads** to a number larger than the number of CPUs available can cause recovery to take longer than if it were single-threaded.

**Performance impact: variable**

## Use private commands

On multi-processor systems, if many threads are executing the same commands, then try setting **PrivateCommands**=1 to improve throughput or response time. The use of private commands increases the amount of temporary space used.

# XLA acknowledgement modes

**Performance impact: medium**

## Prefetch multiple update records

Prefetching multiple update records at a time is more efficient than obtaining each update record from XLA individually. Because updates are not prefetched when you use AUTO_ACKNOWLEDGE mode, it can be slower than the other modes. If possible, you should design your application to tolerate duplicate updates so you can use DUPS_OK_ACKNOWLEDGE, or explicitly acknowledge updates. Explicitly acknowledging updates usually yields the best performance if the application can tolerate not acknowledging each message individually.

**Performance impact: medium**

## Acknowledge XLA updates

To explicitly acknowledge an XLA update, you call `acknowledge` on the update message. Acknowledging a message implicitly acknowledges all previous messages. Typically, you receive and process multiple update messages between acknowledgements. 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.

# 9

# *The TimesTen Query Optimizer*

The TimesTen cost-based query optimizer uses information about an application's tables and their available indexes to choose a fast path to the data. Application developers can examine the plan chosen by the optimizer to check that indexes are used appropriately. If necessary, application developers can also modify the optimizer's behavior so that it chooses a different plan.

This chapter includes the following topics:

## When optimization occurs

It is useful to understand when TimesTen performs query optimization, since a single command may be optimized several times.

TimesTen invokes the optimizer whenever a SELECT, UPDATE, DELETE, INSERT SELECT or CREATE VIEW statement is prepared through an ODBC **SQLPrepare** or **SQLExecDirect** function or any of the JDBC execute methods. If **SQLPrepare** is used, the resulting plan persists until an *invalidating* event occurs, or the command is dropped by the application. A command is *invalidated* under the following circumstances:

- A table it uses is dropped
- A table it uses is altered
- An index on a table it references is dropped
- An index is created on a table it references
- Statistics are recomputed

An invalid command is usually reprepared automatically just before it is re-executed. This means that the optimizer is invoked again at this time, possibly resulting in a new plan. Thus, a single command may be prepared several times.

---

**Note:** When using JDBC, you must manually reprepare commands when a table has been altered.

---

A command may have to be prepared manually if, for example, the table that the command referenced was dropped and a new table with the same name was created. When you prepare a statement manually, you should commit the prepare statement so it can be shared. If the command is recompiled because it was invalid, and if recompilation involves DDL on one of the referenced tables, then the prepared statement must be committed to release the command lock.

For example, in ODBC a command joining tables T1 and T2 may undergo the following changes:

| | |
|---|---|
| **SQLPrepare** | Command is prepared. |
| **SQLExecute** | Command is executed. |
| **SQLExecute** | Command is re-executed. |
| | . . . |
| **Create Index on T1** | Command is invalidated. |
| **SQLExecute** | Command is reprepared, then executed. |

| | |
|---|---|
| **SQLPrepare** | Command is prepared. |
| **SQLExecute** | Command is re-executed. |
| | . . . |
| **ttOptUpdateStats** on T1 | Command is invalidated (if the invalidate flag is passed to the **ttOptUpdateStats** procedure). |
| | . . . |
| SQLExecute | Command is reprepared, then executed. |
| SQLExecute | Command is re-executed. |
| | . . . |
| SQLFreeStmt | Command is dropped. |

In JDBC, a command joining tables T1 and T2 may undergo the following changes:

| | |
|---|---|
| **Connection.prepareStatement** | Command is prepared. |
| **PreparedStatement.execute** | Command is executed. |
| **PreparedStatement.execute** | Command is re-executed. |
| | . . . |
| **Create Index on T1** | Command is invalidated. |
| **PreparedStatement.execute** | Command is reprepared, then executed. |
| **PreparedStatement.execute** | Command is re-executed. |

| | |
|---|---|
| **Connection.prepareStatement** | Command is prepared. |
| | . . . |
| ttOptUpdateStats on T1 | Command is invalidated (if the invalidate flag is passed to the **ttOptUpdateStats** procedure). |
| | . . . |
| **PreparedStatement.execute** | Command is reprepared, then executed. |
| **PreparedStatement.execute** | Command is re-executed. |
| | . . . |
| **PreparedStatement.close** | Command is dropped. |

As illustrated, optimization is generally performed at prepare time, but it may also be performed later when indexes are dropped or created, or when statistics are modified. Optimization does not occur if a prepare can use a command in the cache.

If a command was prepared with the `genPlan` flag set, it will be recompiled with the same flag set. Thus, the plan is generated even though the plan for another query was found in the SYS.PLAN table.

If an application specifies hints to modify the optimizer's behavior (see "Modifying plan generation" on page 187), these hints persist until the command is deleted. (For example, when the ODBC **SQLPrepare** function or JDBC **Connection.prepareStatement** method is called again on the same handle or when the **SQLFreeStmt** function or **PreparedStatement.close** method is called.) This means that any intermediate reprepare operations that occur because of invalidations will use those same hints.

# Viewing a plan

Several steps are needed to view a plan for a prepared SELECT, UPDATE, DELETE, INSERT SELECT or CREATE VIEW statement:

- Generating the plan involves instructing TimesTen to store the command's plan in the system PLAN table.
- Preparing the statement means calling the ODBC **SQLPrepare** function or JDBC **Connection.prepareStatement** method on the statement.
- Reading the SYS.PLAN table is the final step.

After the first two steps have been completed, TimesTen places the command's plan in the PLAN table. The stored plan is then updated automatically whenever the command is reprepared. This re-preparation is not likely to occur often. However, if a table in the statement is altered, or if indexes are created or dropped, or the application chooses to invalidate commands when statistics are updated, it may help to read the PLAN table again to see if the command's plan has changed.

## Generating the plan

Before you can view the plan, you must call the built-in procedure **ttOptSetFlag** with the `GenPlan` flag. This call informs TimesTen that all subsequent calls to the ODBC **SQLPrepare** function or JDBC **Connection.prepareStatement** method in the transaction should store the resulting plan in the current SYS.PLAN table.

---

**Note:** Make sure AUTOCOMMIT is not set. If it is, the current transaction will complete after the processing of the command and prepares in the next transaction will not be affected.

---

The SYS.PLAN table only stores one plan, so each call to the ODBC **SQLPrepare** function or JDBC **Connection.prepareStatement** method overwrites any plan currently stored in the table.

If a command was prepared with the `genPlan` flag set, it will be recompiled with the same flag set. Thus, the plan is generated even though the plan for another query was found in the SYS.PLAN table.

For the purposes of experimentation, you can try query and optimizer hints using the **ttIsql** utility. To display optimizer plans, issue the commands:

```
autocommit 0;
showplan 1;
```

## Reading the PLAN table

Once plan generation has been turned on and a command has been prepared, one or more rows in the SYS.PLAN table store the plan for the command. The number of rows in the table depends on the complexity of the command. Each

row has seven columns, as described in the chapter "System and Replication Tables" in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

**Example 9.1 A Query and its plan**

Assume you prepare the following query:

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

The optimizer may generate the five SYS.PLAN rows shown in the following table. Each row is one step in the plan and reflects an operation that is performed during query execution.

| Step | Level | Operations | Tbl Names | IXName | Pred | Other Pred |
|---|---|---|---|---|---|---|
| 1 | 3 | TblLkTtreeScan | T1 | IX1 | | |
| 2 | 3 | TblLkTtreeScan | T2 | IX2(D) | | T2.B <> 0 |
| 3 | 2 | MergeJoin | | | T1.A = –T2.A | |
| 4 | 2 | TblLkTtreeScan | T3 | IX3(D) | | |
| 5 | 1 | MergeJoin | | | T2.A = T3.A | T3.B / T1.B > 1 |

The remainder of this section provides detailed information about each column in the SYS.PLAN table, using Example 9.1 throughout.

## PLAN table columns

The SYS.PLAN table has seven columns. They are described in this section.

**Column 1 (Step)**

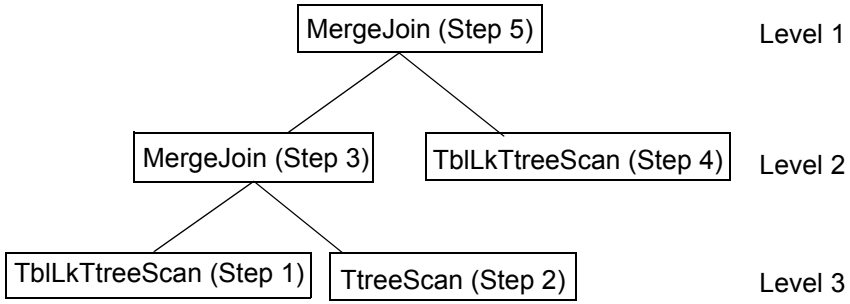
Indicates the order of operation. This example uses a table lock T-tree scan. The order is:

1. Table locking T-tree scan of IX1 on table T1.
2. Table locking T-tree scan of IX2 on T2.
3. Merge join of T1 and T2 and so forth.

The steps always start with 1.

**Column 2 (Level)**

Indicates the position of the operation in the join-tree diagram that describes the execution. For the example above, the join tree looks like:




**Column 3 (Operation)**

Indicates the type of operation being executed. For a description of the values in this field and the type of table scan each represents, see SYS.PLAN in the chapter "System and Replication Tables" in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

Not all operations the optimizer performs are visible to the user. Only operations significant to performance analysis are shown in the SYS.PLAN table. TblLk is an optimizer hint that is honored at execution time in serializable or read-committed isolation. Table locks are used during a scan only if row locks are disabled during preparation.

**Column 4 (TblNames)**

Indicates the table that is being scanned. This column is used only when the operation is a scan (one of the first five operations listed above). In all other cases, this column is NULL.

**Column 5 (IXName)**

Indicates the index that is being used. This column is used only when the operation is an index scan using an existing index (using a hash or T-tree scan). In all other cases, this column is NULL. Names of T-tree indexes are followed with

"(D)" if the scan is descending (from large to small rather than from small to large).

**Column 6 (Pred)**

Indicates the predicate that participates in the operation, if there is one. Predicates are used only with index scan and `MergeJoin` operations.

This column may be NULL (no predicate) for a T-tree scan. The optimizer may choose a T-tree scan over a table scan because it has two useful properties in addition to filtering:

- Rows are returned in sorted order (on index key).
- Rows may be returned faster (especially if the table is sparse).

In Example 9.1, the T-tree scans are used for their sorting capability; none of them evaluates a predicate.

The predicate character string is limited to 1,024 characters.

**Column 7 (Other Pred)**

Indicates any other predicate that is applied while the operation is being executed. These predicates do not participate directly in the scan or join but are evaluated on each row returned by the scan or join.

For example, at step 2 of the plan generated for the example above, a T-tree scan is performed on table T2. When that scan is performed, the predicate `T2.B <> 0` is also evaluated. Similarly, once the final merge-join has been performed, it is then possible to evaluate the predicate `T3.B / T1.B > 1`.

# Modifying plan generation

This section explains why you may want to modify execution plans and then describes how to modify them.

## Why modify an execution plan?

Applications may want to modify an execution plan for two reasons:

- ***The plan is optimally fast but is ill-suited for the application.*** The optimizer may select the fastest execution path, but this path may not be desirable from the application's point of view. For example, if the optimizer chooses to use certain indexes, these choices may prevent other operations—such as certain update or delete operations—from occurring simultaneously on the indexed tables. In this case, an application can prevent the use of those indexes.

  The plan chosen by the optimizer may also consume more memory than is available or than the application wants to allocate. For example, this may happen if the plan stores intermediate results or requires the creation of temporary indexes.
- ***The plan is not optimally fast.*** The query optimizer chooses the plan that it estimates will execute fastest based on its knowledge of the tables' contents, available indexes, statistics and the relative costs of various internal operations. The optimizer often has to make estimates or generalizations when evaluating this information, so there can be instances where it does not choose the fastest plan. In this case, an application can adjust the optimizer's behavior to try to produce a better plan.

## When to modify an execution plan

Applications can modify an execution plan by giving hints to the optimizer. Hints are specified by calls to one of the TimesTen optimizer built-in procedures (see "How to modify execution plan generation" on page 192) and are in effect for all calls to the ODBC **SQLPrepare** function or JDBC **PreparedStatement** objects in the transaction.

---

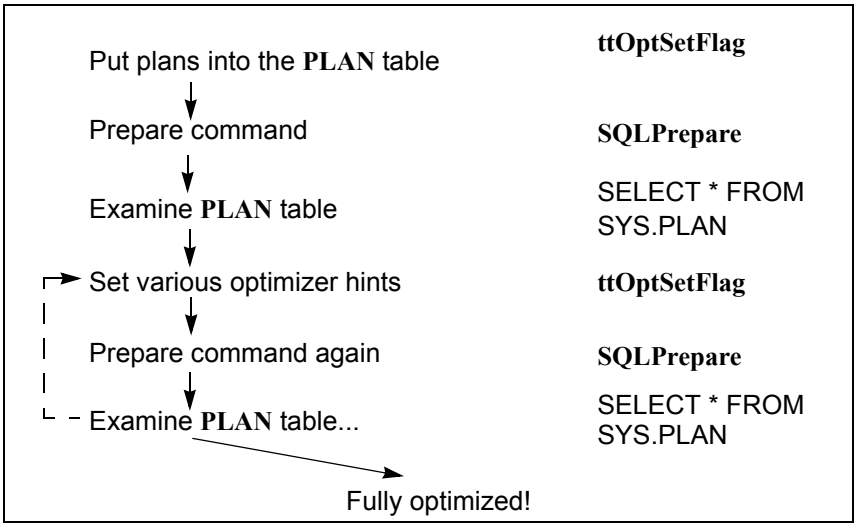
**Note:** Make sure AUTOCOMMIT is not set. If it is, the current transaction will complete after processing the **ttOptSetFlag** procedure and prepares in the next transaction will not be affected.

---

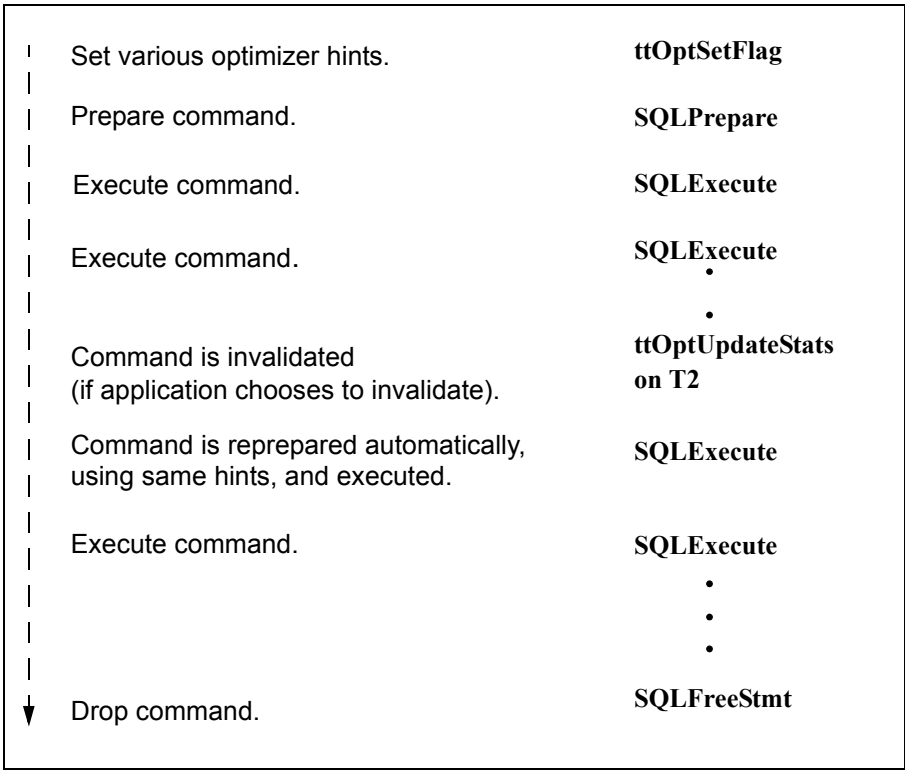
If a command is prepared with certain hints in effect, those hints continue to apply if the command is reprepared automatically, even when this happens outside the initial prepare's transaction. This can happen when a table is altered, or an index is dropped or created, or when statistics are modified, as described in "When optimization occurs" on page 180.

If a command is prepared without hints, subsequent hints will not affect the command if it is reprepared automatically. An application must call the ODBC **SQLPrepare** function or JDBC **Connection.prepareStatement** method a second time so that hints have an effect.

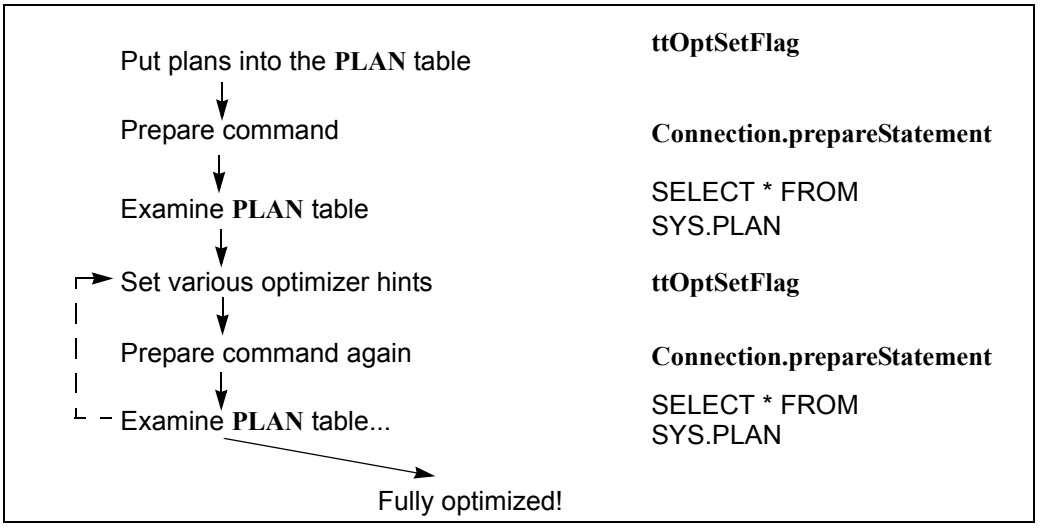
**Example 9.2** When using ODBC, a developer tuning a join on T1 and T2 might go through the following steps:



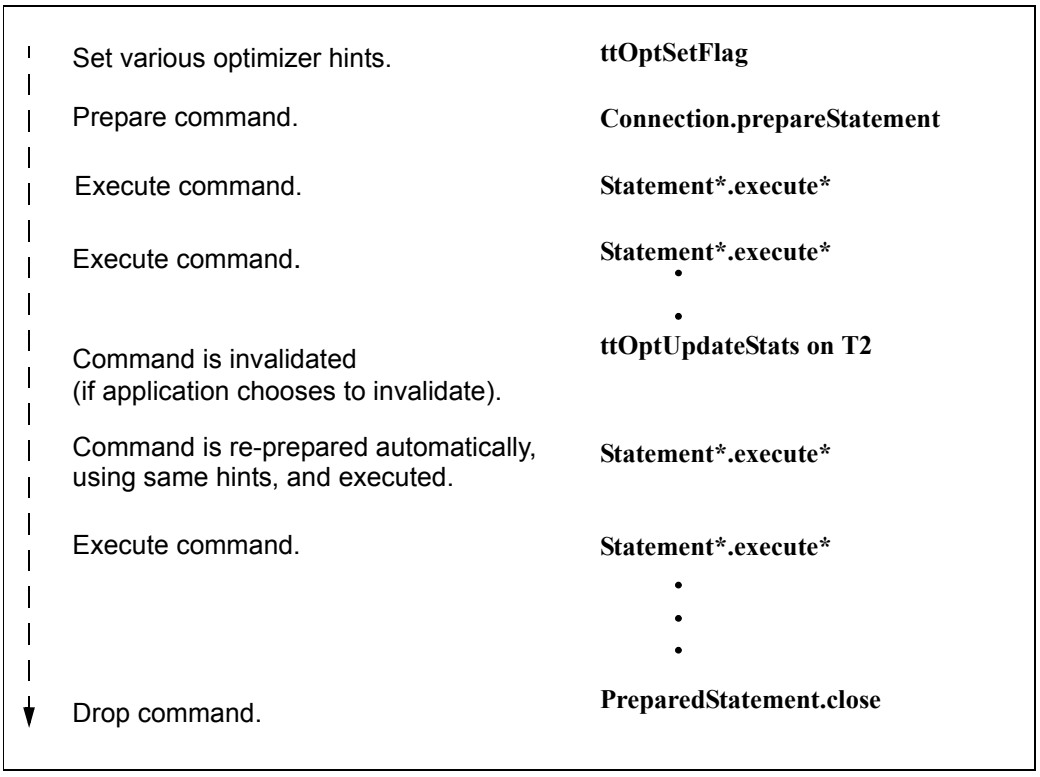
During execution, the application may then go through the following steps (with no user intervention):



**Example 9.3** When using JDBC, a developer tuning a join on T1 and T2 might go through the following steps:



During execution, the application may then go through the following steps (with no user intervention):



## How to modify execution plan generation

To change the query optimizer behavior, an application calls one of the following built-in procedures using the ODBC procedure call interface:

- ttOptClearStats
- ttOptEstimateStats
- ttOptSetColIntvlStats
- ttOptSetFlag
- ttOptSetOrder
- ttOptSetTblStats
- ttOptUpdateStats
- ttOptUseIndex

The procedure **ttOptSetFlag** sets certain optimizer parameters. **ttOptSetOrder** allows an application to specify the table join order. The procedure **ttOptUseIndex** allows an application to disable the use of certain indexes. The remaining procedures manipulate statistics the TimesTen Data Manager maintains on the application's data that are used by the query optimizer to estimate costs of various operations. See the Chapter 3, "Built-In Procedures" in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

Example 9.4 shows an ODBC example of how to use **ttOptSetFlag**. Example 9.5 shows a JDBC example.

**Example 9.4** The ODBC example below illustrates the use of **ttOptSetFlag** to prevent the optimizer from choosing a merge join.

```
import java.sql.*;
class Example
{
 public void myMethod() {
    CallableStatement cStmt;
    PreparedStatement pStmt;
    . . . . .
    try {
        . . . . . .
        // Prevent the optimizer from choosing Merge Join
        cStmt = con.prepareCall("{
            CALL ttOptSetFlag('MergeJoin', 0)}");
        cStmt.execute();
```

```
            // Next prepared query
            pStmt=con.prepareStatement(
            "SELECT * FROM Tbl1, Tbl2 WHERE Tbl1.ssn=Tbl2.ssn");
            . . . . . .
            catch (SQLException ex) {
                ex.printStackTrace();
            }
        }
        . . . . . .
}
```

**Example 9.5** The JDBC example below illustrates the use of **ttOptSetFlag** to prevent the optimizer from choosing a merge join.

```
#include <sql.h>
SQLRETURN rc;
SQLHSTMT hstmt; fetchStmt;
....
rc = SQLExecDirect (hstmt, (SQLCHAR *)
     "{CALL ttOptSetFlag (MergeJoin, 0)}",
     SQL_NTS)
/* check return value */
...
rc = SQLPrepare (fetchStmt, ...)
/* check return value */
...
```

You can also experiment with optimizer settings using the **ttIsql** utility. The commands that start with `try` control the optimizer hints. To view the current optimizer hint settings, use the `optprofile` command.

# 10

# *UNIX Configuration Files*

This chapter provides supplemental information about the UNIX configuration files:

# Working with the ODBC.INI file

This section includes the following topics:



## The user ODBC.INI file

On UNIX, user DSNs are defined in the file `$HOME/.odbc.ini` or in a file named by the ODBCINI environment variable. This file is referred to as the "user ODBC.INI file." Although a user DSN is private to the user who created it, it is only the DSN (the character-string name and its attributes) that is private. The underlying data store can be referenced by other user DSNs or by system DSNs. TimesTen supports data sources for the TimesTen Data Manager and data sources for the TimesTen Client in the `.odbc.ini` file.

For information on how to create a copy of the `.odbc.ini` file in your home directory and how to override the name and location of the `.odbc.ini` file, see "Data source names" on page 33.

## The system ODBC.INI file

On UNIX, system DSNs are defined in the `/var/TimesTen/sys.odbc.ini` file. This file is referred to as the "system ODBC.INI file." A system DSN can be used by any user on the machine on which the system DSN is defined.

## Searching for a DSN

See "Searching for a DSN" on page 67 for the rules of precedence that TimesTen follows when searching for a DSN.

## ODBC Data Sources

Each entry in the optional ODBC Data Sources section lists a data source and a description of the driver it uses. The data source section has the format:

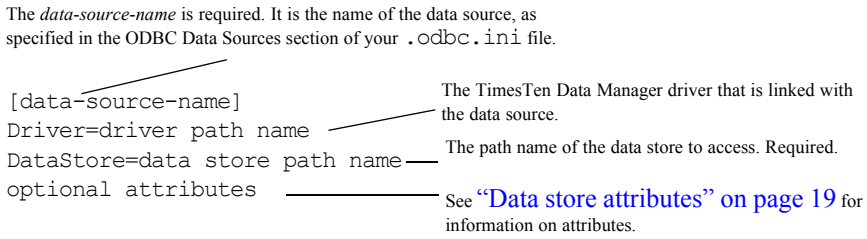
```
[ODBC Data Sources]
data-source-name=driver-description
```

The *data-source-name* is required. It identifies the data source to which the driver connects. You choose this name.

The *driver-description* is required. It describes the driver which connects to the data source.

## Data Source Specification

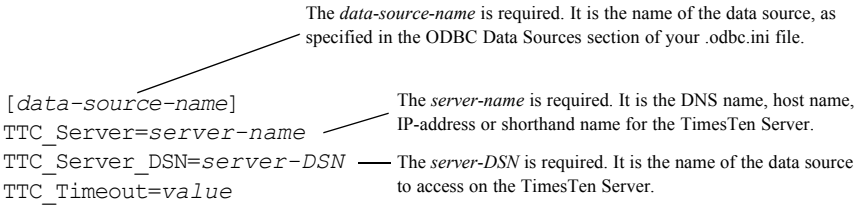
Each data source listed in the ODBC Data Sources section has its own data source specification section. The data store specification for TimesTen Data Manager data stores has the format:




For example, a data source called *RunData* may have the following data source specification entry:

```
[RunData]
Driver=install_dir/lib/libtten.sl
DataStore=/users/robin/SalesDs
#create data store if it's not found
AutoCreate=1
#do not wait if cannot connect to data store
WaitForConnect=0
#remove old log files at connect and checkpoint
LogPurge=1
```

The data store specification for TimesTen Client configurations has the format:




---

**Note:** Most TimesTen Data Manager attributes are ignored for TimesTen Client data stores.

---

For example, the data source called *RunDataCS* that connects to the data source, named *RunData*, on the *ttserver* TimesTen Server may have the data source specification entry:

```
[RunDataCS]
TTC_Server=ttserver
TTC_Server_DSN=RunData
TTC_Timeout=30
```

For example, the data source called *ShmRunDataCS* that uses a shared memory segment to connect to the data source named *RunData* on the *ShmHost60* TimesTen Server may have the data source specification entry:

```
[ShmRunDataCS]
TTC_Server=ShmHost60
TTC_Server_DSN=RunData
TTC_Timeout=30
```

## odbc.ini file example

The following example shows a UNIX `.odbc.ini` file:

```
[ODBC Data Sources]
RunData_giraffe=TimesTen 6.0 Driver
RunDataCS_giraffe=TimesTen Client 6.0

[RunDataCS_giraffe]
TTC_Server=tt_server_logical
TTC_Server_DSN=RunData
TTC_Timeout=30

[RunData_giraffe]
Driver=install_dir/lib/libtten.sl
DataStore=/users/robin/RunData
PermSize=8
Exclusive=0
Logging=1
ThreadSafe=1
```

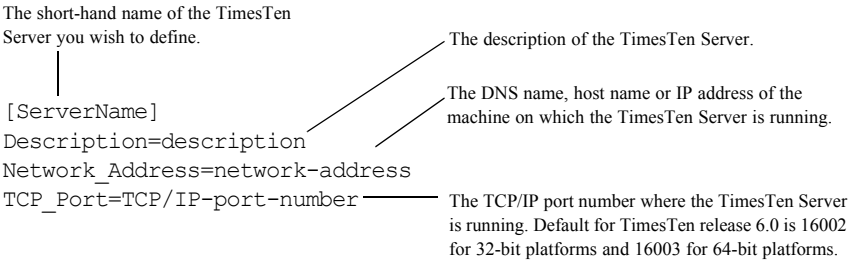
# Working with the TTCONNECT.INI file

TimesTen uses theTTCONNECT.INI file to define the names and attributes for servers and the mappings between logical server names and their network addresses. This information is stored on machine where the TimesTen Client is installed. By default, the TTCONNECT.INI file is `/var/TimesTen/sys.ttconnect.ini`.

To override the name and location of this file at runtime, set the `SYSTTCONNECTINI` environment variable to the name and location of the TTCONNECT.INI file before launching the TimesTen application.

## Defining a server name on UNIX

You can define short-hand names for TimesTen Servers on UNIX in the TTCONNECT.INI file. The format of a TimesTen Server specification in the TTCONNECT.INI file is:




For example, the server specification for a remote TimesTen Server might appear as:

```
[ttserver]
Description=TimesTen Client/Server
Network_Address=server.company.com
TCP_Port=16002
```

For a local TimesTen Client/Server application that is using UNIX domain sockets, the network address must be defined as `ttLocalHost`. The server specification might appear as:

```
[LocalHost60]
Description=Shm TimesTen Client/Server
Network_Address=localhost
TCP_Port=16002
```

For a TimesTen Client/Server application that is using a shared memory segment for inter-process communication, the network address must be defined as `ttShmHost`. The server specification might appear as:

```
[ShmHost60]
Description=Shm TimesTen Client/Server
Network_Address=shmhost.company.com
```

```
TCP_Port=16002
```

# Index

## Symbols



## A



## B



## C

## D

## E



## F



## G



## H



## I

## J



## L



## M

## N



## O



## P

## Q



## R



## S

## T

## U



## V



## W

## X