Oracle® TimesTen In-Memory Database

Operations Guide

Release 11.2.1

E13065-03

August 2009

Oracle TimesTen In-Memory Database Operations Guide, Release 11.2.1

E13065-03

Contents

Preface

Oracle TimesTen In-Memory Database is a memory-optimized relational database. Deployed in the application tier, Oracle TimesTen In-Memory Database operates on databases that fit entirely in physical memory using standard SQL interfaces. High availability for the in-memory database is provided through real-time transactional replication.

This guide provides:

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

Audience

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

Related documents

TimesTen documentation is available on the product distribution media and on the Oracle Technology Network:

`http://www.oracle.com/technology/documentation/timesten_doc.html`

Conventions

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 and AIX.

This document uses the following text conventions:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Convention	Meaning
`italic monospace`	Italic monospace type indicates a variable in a code example that you must replace. For example: `Driver=`*`install_dir`*`/lib/libtten.sl` Replace *`install_dir`* with the path of your TimesTen installation directory.
[ ]	Square brackets indicate that an item in a command line is optional.
{ }	Curly braces indicated that you must choose one of the items separated by a vertical bar (|) in a command line.
|	A vertical bar (or pipe) separates alternative arguments.
. . .	An ellipsis (. . .) after an argument indicates that you may use more than one argument on a single command line.
%	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:

Convention	Meaning
`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.
`bits` or *`bb`*	Two digits, either 32 or 64, that represent either the 32-bit or 64-bit operating system.
`release` or *`rr`*	Three numbers that represent the first three numbers of the TimesTen release number, with or without a dot. For example, 1121 or 11.2.1 represents TimesTen Release 11.2.1.
`jdk_version`	Two digits that represent the version number of the major JDK release. Specifically, 14 represent JDK 1.4; 5 represents JDK 5.
`timesten`	A sample name for the TimesTen instance administrator. You can use any legal user name as the TimesTen administrator. On Windows, the TimesTen instance administrator must be a member of the Administrators group. Each TimesTen instance can have a unique instance administrator name.
`DSN`	The data source name.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Deaf/Hard of Hearing Access to Oracle Support Services

To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at `http://www.fcc.gov/cgb/consumerfacts/trs.html`, and a list of phone numbers is available at `http://www.fcc.gov/cgb/dro/trsphonebk.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`

What's New

This section summarizes the new features and functionality of Oracle TimesTen In-Memory Database Release 11.2.1 that are documented in this guide, providing links into the guide for more information.

New features for release 11.2.1

This guide has information about the following new features:

- Access control
- Asynchronous materialized views
- SQL command cache
- Bitmap indexes
- Transaction log buffer file size
- Automatic client failover
- PL/SQL support
- Replication performance

Access control

Oracle TimesTen In-Memory Database release 11.2.1 has a new access control model. The previous TimesTen access control model has been removed. There is no backwards compatibility between the two models.

Users are defined at the database level rather than at the installation level. Privileges are defined at the object level. The system privileges in TimesTen release 7.0 and previous releases have been replaced with system privileges that are similar to Oracle database system privileges.

The main changes to access control in this release are as follows:

- Access control is always on. You can no longer install TimesTen with access control disabled.
- Only the instance administrator can create and destroy databases.
- Separate databases within an instance can have different users.
- Every object must have an owner that is a user in the database. You cannot create object `bob.t1` unless user `bob` exists in the database.

- Every object owner has access to their own objects. A user does not have access to objects owned by other users unless explicitly granted access by the object's owner or by a user with ADMIN privilege. Also, if the PUBLIC role has been granted access to a given object, then all database users have access to that object.
- Privileges are checked at prepare and when the statement is first executed. Subsequent executions of a statement require further privilege checks only when a revoke operation is executed in the database.
- You cannot create or alter a user by executing the CREATE USER `user` IDENTIFIED BY PASSWORD `password` or ALTER USER `user` IDENTIFIED BY PASSWORD `password` SQL statements across a client/server connection.
- You cannot drop a user with existing objects.
- Many of the utilities and built-in procedures require a certain privilege in order to execute. In addition, in order to modify or connect with certain first connection attributes, certain privileges are required. The required privilege for each is described with the utility, built-in procedure or first connection attribute description in the *Oracle TimesTen In-Memory Database Reference*.
- Only the instance administrator can execute the `ttRepAdmin -duplicate` utility. The instance administrator must have the same operating system user name on both source and target machines to execute `ttRepAdmin -duplicate`. For more details, see *Oracle TimesTen In-Memory Database Reference*.
- In previous versions, users could create an object with any owner name, even though no such user existed. For example, user `terry` could create the object `pat.table1` even though there was no user `pat` in the database. However, since every object now has an owner, when restoring from a TimesTen database from a release before 11.2.1 using the `ttMigrate` utility, TimesTen automatically creates the user `pat` for this object. The user `pat` will have no privileges and will have an internally generated password. For more information, see "Data Store Upgrades" in the *Oracle TimesTen In-Memory Database Installation Guide*.
- If you are using the `ttMigrate` utility to save or restore the entire TimesTen database, you must have the ADMIN privilege. However, if you are using `ttMigrate` to save or restore a few database objects, then you need only the privileges required to read or create those database objects. For more information, see the description for `ttMigrate` in the *Oracle TimesTen In-Memory Database Reference*.

For details on creating users and assigning privileges for access to database objects, see Chapter 4, "Managing Access Control".

Asynchronous materialized views

Materialized views can be refreshed asynchronously at either a specified time or through manual initiation. You can either have the deferred transactions updated incrementally or with a complete refresh. A materialized view log is created and associated with the asynchronous materialized view to facilitate the incremental refresh of data from the detail tables. For full details, see "Understanding materialized views" on page 7-11 and "Working with materialized views" on page 7-14.

SQL command cache

All commands executed—SQL statements, built-in procedures, and so on—are stored in the SQL Command Cache, which uses temporary memory. The commands are stored up until the limit of the SQL Command Cache is reached, then the new commands are stored after the last used commands are removed. You can retrieve one or more of these commands that are stored in the SQL Command Cache. For full details on the SQL command cache, see "Viewing SQL commands stored in the SQL Command Cache" on page 10-3.

You can also view the query plan information to monitor and troubleshoot your queries. For details, see "Viewing query plans associated with commands stored in the SQL Command Cache" on page 10-8.

Bitmap indexes

TimesTen supports bitmap indexes. See "Overview of index types" on page 7-22.

Transaction log buffer file size

The configuration for the transaction log buffer file size has been modified from LogBuffSize, which was defining the size in KBs, to LogBufMB, which defines the transaction log buffer size in MBs. This affects how you configure the data store size, as described in "Changing data store size" on page 1-18. Use the LogBufMB to configure performance, as described in "Increase LogBufMB if needed" on page 9-3.

Automatic client failover

You can configure automatic client failover for data stores that have active standby pair replication schemes. This enables the client to fail over automatically to the server on which the standby data store resides. See "Configuring automatic client failover" on page 2-10.

PL/SQL support

You can specify values for PL/SQL connection attributes in a data source name (DSN). See "Specifying PL/SQL connection attributes in a DSN" on page 1-14.

You can use the `ttIsql` utility to create and execute PL/SQL blocks. See "Creating and executing PL/SQL blocks" on page 6-10 and "Using OUT parameters" on page 6-11.

You can use the `ttIsql` utility to display PL/SQL objects. See "Displaying information about PL/SQL objects" on page 6-12.

Replication performance

Use the `RecoveryThreads` first connection attribute to increase performance of active standby pairs. See "Increase replication throughput for active standby pairs" on page 9-6.

1 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 6-2.
- Execute an application that uses the data store, as described in *Oracle TimesTen In-Memory Database Java Developer's Guide* and *Oracle TimesTen In-Memory Database C Developer's Guide*.

The main topics are:

- TimesTen ODBC and JDBC drivers
- Data source names
- Creating a DSN on Windows
- Creating a DSN on UNIX
- DSN examples
- Specifying the size of a data store
- Specifying a RAM policy
- Copying, migrating, backing up and restoring a data store
- Working with the ODBC.INI file

TimesTen ODBC and JDBC drivers

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.

An ODBC driver manager is included with Windows.

For UNIX, ODBC driver managers are not necessarily supplied with the operating system.

TimesTen supplies a driver manager for either Windows or UNIX with the Quick Start sample applications.

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

The following sections describe some basic concepts that will help you define TimesTen data stores:

- TimesTen ODBC drivers
- TimesTen JDBC driver and driver manager

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 11.2.1 Driver.
Windows	Debug	TimesTen Data Manager 11.2.1 Debug Driver.
Windows	Client	TimesTen Client 11.2.1 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.

Platform	Version	Location and name
HP-UX	Production	*install_dir*/lib/libtten.sl TimesTen Data Manager 11.2.1 Driver.
HP-UX	Debug	*install_dir*/lib/libttenD.sl TimesTen Data Manager 11.2.1 Debug Driver.
HP-UX	Client	*install_dir*/lib/libttclient.sl TimesTen Client 11.2.1 Driver.
Solaris Linux	Production	*install_dir*/lib/libtten.so TimesTen Data Manager 11.2.1 Driver.
Solaris Linux	Debug	*install_dir*/lib/libttenD.so TimesTen Data Manager 11.2.1 Debug Driver.
Solaris Linux	Client	*install_dir*/lib/libttclient.so TimesTen Client 11.2.1 Driver.
AIX	Production	*install_dir*/lib/libtten.a TimesTen Data Manager 11.2.1 Driver.
AIX	Debug	*install_dir*/lib/libttenD.a TimesTen Data Manager 11.2.1 Debug Driver.
AIX	Client	*install_dir*/lib/libttclient.a TimesTen Client 11.2.1 Driver.

TimesTen JDBC driver and driver manager

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.

The JDBC driver manager in the 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.

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

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. If a user tries to use a DSN that has connection attributes for which they do not have privileges, such as first connection attributes, they receive an error. For more information on first connection attribute privileges, see "Data Store Attributes" in the *Oracle TimesTen In-Memory Database Reference*.

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
- Data Manager and Client DSNs
- Connection attributes for Data Manager DSNs
- Thread programming with TimesTen

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, consisting of 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 `*install_dir*/info/sys.odbc.ini`. 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 in either the production version or the debug version are called "Data Manager DSNs." DSNs that use the TimesTen Client are called "Client DSNs."

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 maintaining 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 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 including both Client DSNs and Data Manager DSNs that are 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 "Working with the TimesTen Client and Server" on page 2-1.

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 the TimesTen database is loaded into memory. Only the instance administrator can load a database with first connection attribute settings. By default, TimesTen loads an idle database, which is a database with no connections, into memory when a first connection is made to it. These attributes persist for 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 "Working with the TimesTen Client and Server" on page 2-1 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 "Data Store Attributes" in the *Oracle TimesTen In-Memory Database Reference.*

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

UNIX 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.

Thread programming with TimesTen

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

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 "TimesTen ODBC and JDBC drivers" on page 1-1 to find out what you need to consider as you set up the data store.

For additional examples of setting up a data store, see "DSN examples" on page 1-12.

This section includes the following topics:

- Specify the ODBC driver
- Specify the DSN

- Specify the connection attributes

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 and driver manager" on page 1-3.

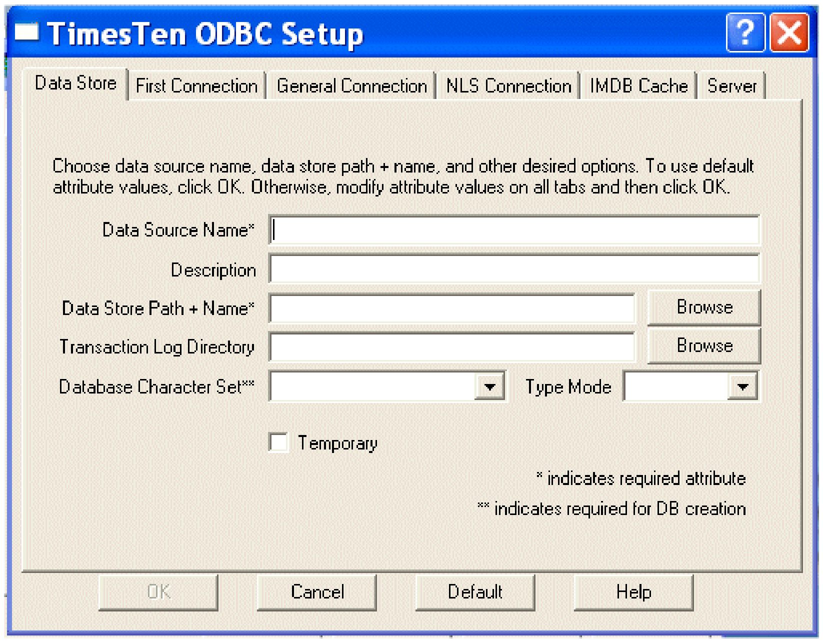
1. On the Windows Desktop, choose **Start > Settings > Control Panel > Administrative Tools > Data Sources (ODBC)**. This opens the ODBC Data Source Administrator.
2. Choose **User DSN** if you want to create a user DSN or **System DSN** if you want to create a system DSN. For a description of user and system DSNs, see "User and system DSNs" on page 1-4.
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 1-2.

Specify the DSN

On the Data Store tab of the TimesTen ODBC Setup dialog, specify a data source name, a data store path name, and a database character set. The Data Store Path Name cannot reference a mapped drive. See Figure 1–1.

Figure 1–1 Data store



Choose data source name, data store path + name, and other desired options. To use default attribute values, click OK. Otherwise, modify attribute values on all tabs and then click OK.

Data Source Name*
Description
Data Store Path + Name*
Transaction Log Directory
Database Character Set**
Type Mode
Temporary

* indicates required attribute
** indicates required for DB creation

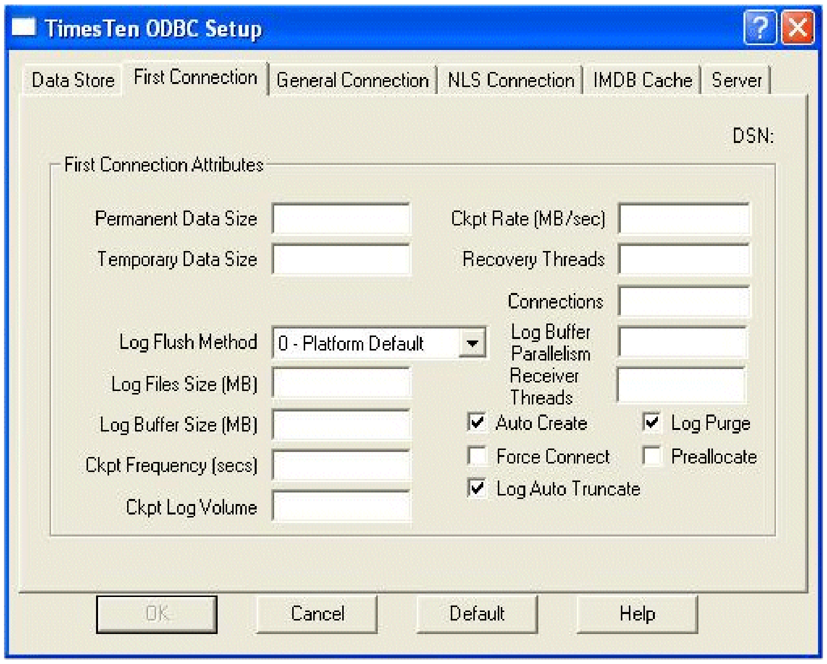
For an explanation of DSNs and data store path names, see "Data source names" on page 1-3. For an explanation of database character sets, see "Choosing a database character set" on page 5-2. The description field is optional.

Specify the connection attributes

Indicate the desired connection attributes under the **First Connection**, **General Connection**, and **NLS Connection** tabs of the TimesTen ODBC Setup dialog. See Figure 1–2, Figure 1–3,and Figure 1–4. If you are using the Cache Connect for Oracle feature, specify the connection attributes shown in Figure 1–5. If you are using a multithreaded client/server configuration, specify the connection attributes shown in Figure 1–6.

For a description of the connection attributes, see "Data Store Attributes" in *Oracle TimesTen In-Memory Database Reference*.

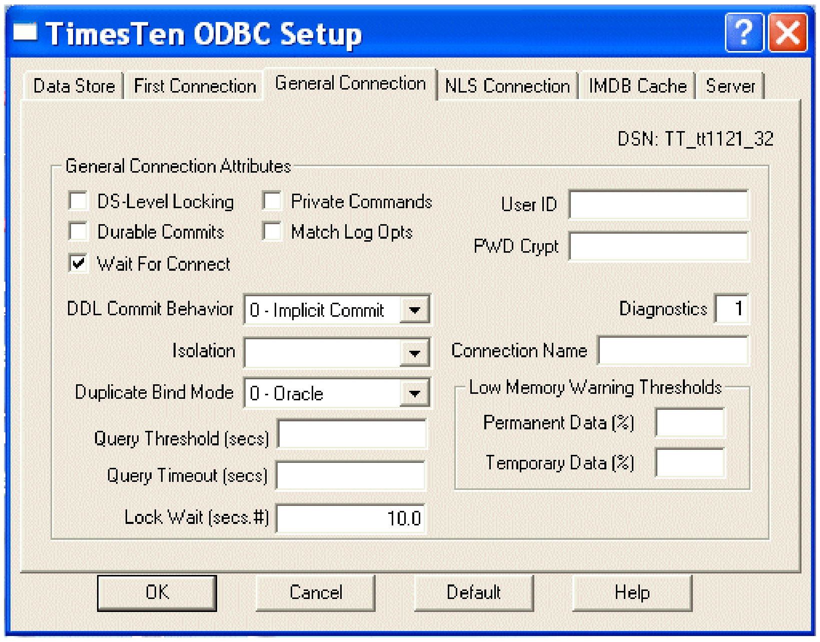
Figure 1–2 First Connection Attributes



Log Flush Method: 0 - Platform Default

[✓] Auto Create [✓] Log Purge
[ ] Force Connect [ ] Preallocate
[✓] Log Auto Truncate

Figure 1–3 General Connection Attributes



DSN: TT_tt1121_32

General Connection Attributes

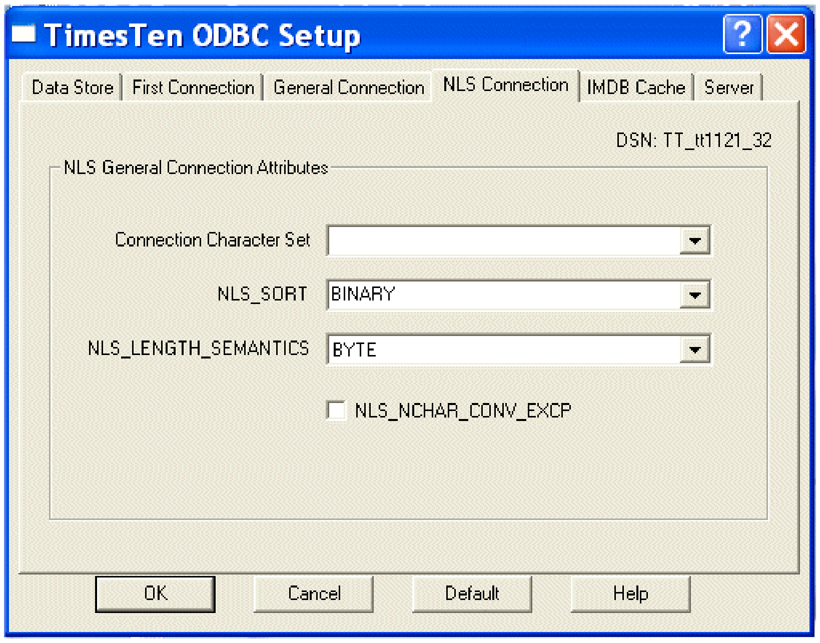
- [ ] DS-Level Locking
- [ ] Durable Commits
- [x] Wait For Connect
- [ ] Private Commands
- [ ] Match Log Opts

User ID	
PWD Crypt	
DDL Commit Behavior	0 - Implicit Commit
Diagnostics	1
Isolation	
Connection Name	
Duplicate Bind Mode	0 - Oracle
Query Threshold (secs)	
Query Timeout (secs)	
Lock Wait (secs.#)	10.0

Low Memory Warning Thresholds

Permanent Data (%)	
Temporary Data (%)	

Figure 1–4 NLS Connection Attributes



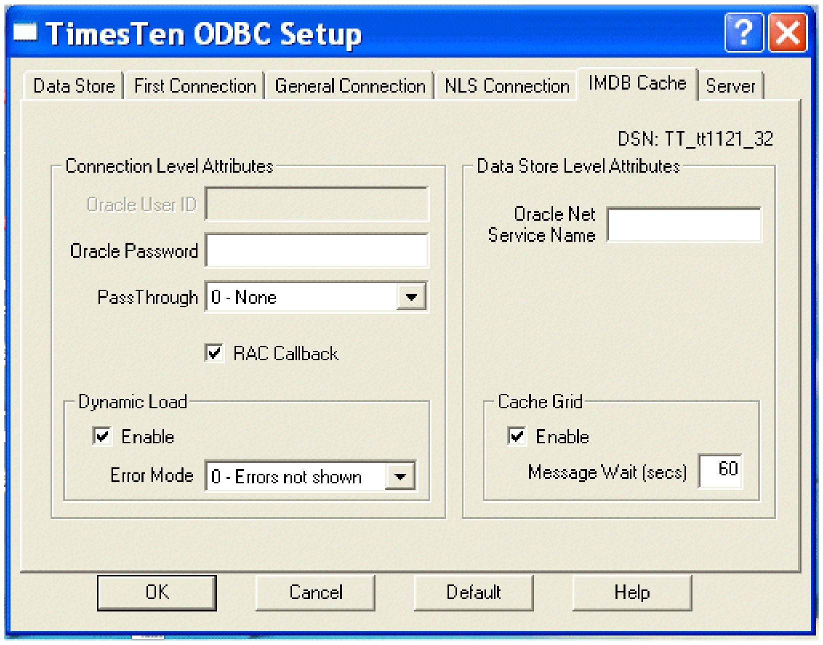
DSN: TT_tt1121_32

NLS General Connection Attributes

Connection Character Set	
NLS_SORT	BINARY
NLS_LENGTH_SEMANTICS	BYTE

- [ ] NLS_NCHAR_CONV_EXCP

Figure 1–5 IMDB Cache Attributes


TimesTen ODBC Setup

Data Store | First Connection | General Connection | NLS Connection | IMDB Cache | Server

DSN: TT_tt1121_32

Connection Level Attributes

Oracle User ID

Oracle Password

PassThrough: 0 - None

RAC Callback: ✓

Dynamic Load

Enable: ✓

Error Mode: 0 - Errors not shown

Data Store Level Attributes

Oracle Net Service Name

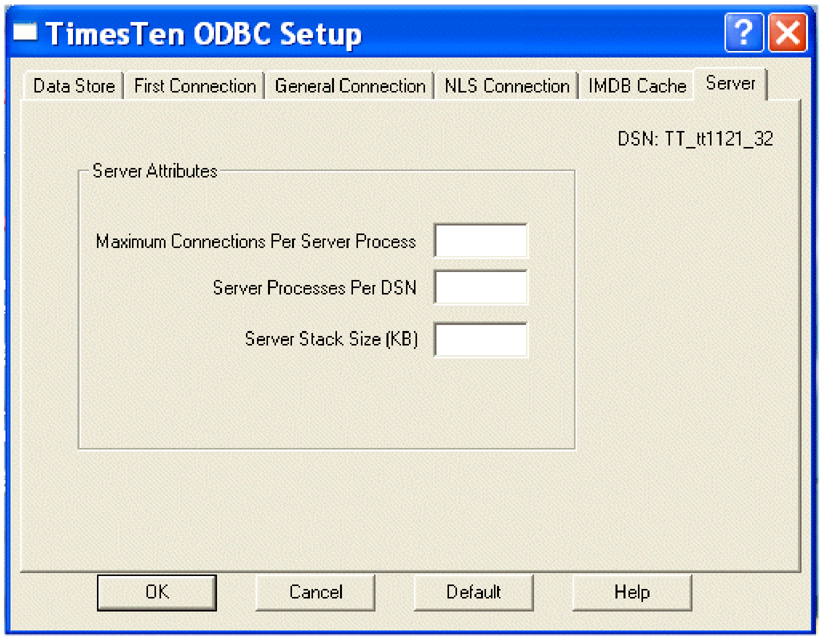
Cache Grid

Enable: ✓

Message Wait (secs): 60

OK | Cancel | Default | Help

Figure 1–6 Server Attributes


TimesTen ODBC Setup

Data Store | First Connection | General Connection | NLS Connection | IMDB Cache | Server

DSN: TT_tt1121_32

Server Attributes

Maximum Connections Per Server Process

Server Processes Per DSN

Server Stack Size (KB)

OK | Cancel | Default | Help

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, read "TimesTen ODBC and JDBC drivers" on page 1-1 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 1-12.

This section includes the following topics:

- Create a user ODBC.INI file
- Specify the DSN
- Specify the ODBC driver
- Specify the data store path name

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 *`install_dir`*`/info/sys.odbc`.ini file. This file is referred to as the "system ODBC.INI file."

The syntax for user and system ODBC.INI files are 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.

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 and driver manager" on page 1-3.

To set the TimesTen driver, specify the DRIVER attribute 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 1-2.

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 1-3.

Choose a database character set

Specify a database character set in the ODBC.INI file. For example:

```
DatabaseCharacterSet=US7ASCII
```

For more information, see "Choosing a database character set" on page 5-2.

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 "Data Store Attributes" in *Oracle TimesTen In-Memory Database Reference*. For examples, see "DSN examples" on page 1-12.

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
- Specifying PL/SQL connection attributes in a DSN
- 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.

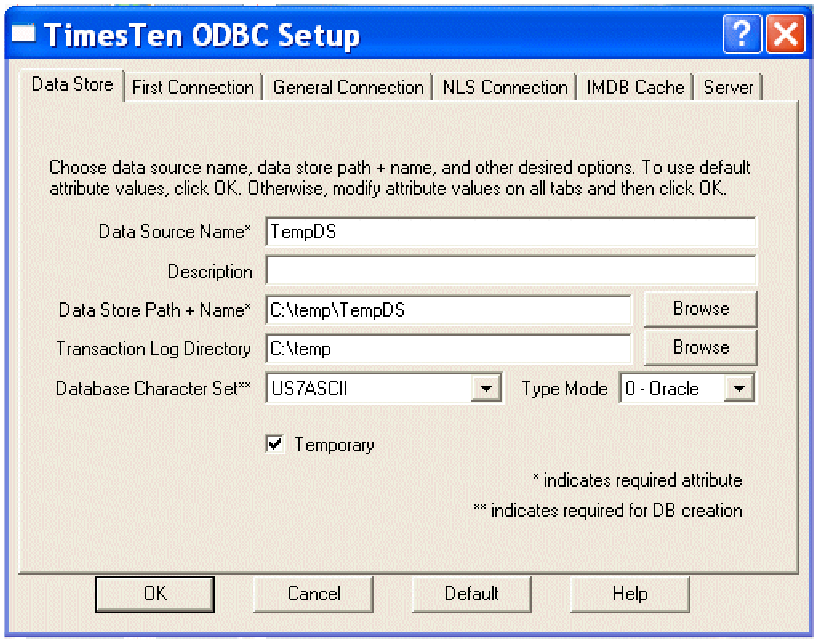
Setting up a temporary data store

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 1–7 and Figure 1–8.

Figure 1–7 Data Store


TimesTen ODBC Setup

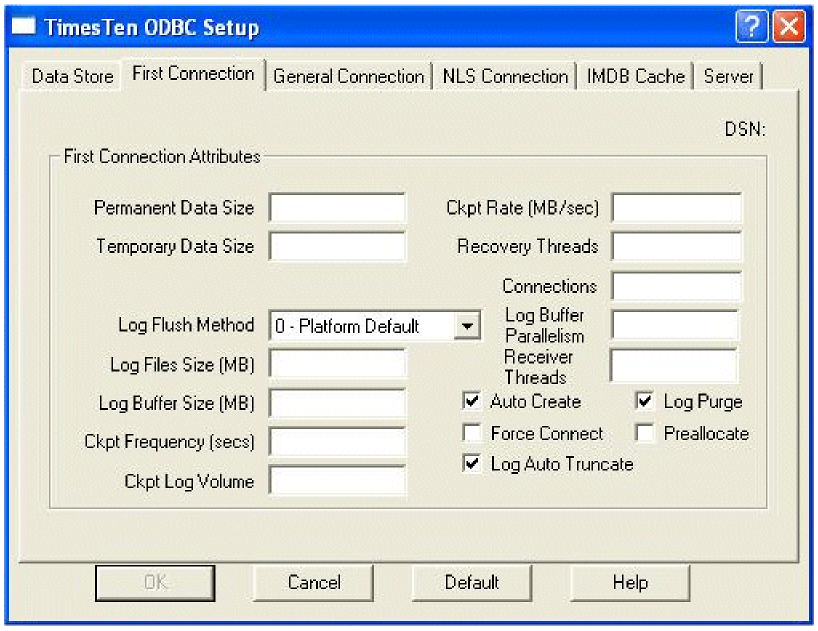
Data Store | First Connection | General Connection | NLS Connection | IMDB Cache | Server

Choose data source name, data store path + name, and other desired options. To use default attribute values, click OK. Otherwise, modify attribute values on all tabs and then click OK.

Data Source Name*: TempDS
Description:
Data Store Path + Name*: C:\temp\TempDS
Transaction Log Directory: C:\temp
Database Character Set**: US7ASCII
Type Mode: 0 - Oracle
[✓] Temporary

* indicates required attribute
** indicates required for DB creation

Figure 1–8 First Connection Attributes


TimesTen ODBC Setup

Data Store | First Connection | General Connection | NLS Connection | IMDB Cache | Server

DSN:

First Connection Attributes

Permanent Data Size:
Temporary Data Size:
Log Flush Method: 0 - Platform Default
Log Files Size (MB):
Log Buffer Size (MB):
Ckpt Frequency (secs):
Ckpt Log Volume:
Ckpt Rate (MB/sec):
Recovery Threads:
Connections:
Log Buffer Parallelism:
Receiver Threads:
[✓] Auto Create
[✓] Log Purge
[ ] Force Connect
[ ] Preallocate
[✓] Log Auto Truncate

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 in "TimesTen ODBC drivers" on page 1-2.

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
DatabaseCharacterSet=US7ASCII
```

Note: A temporary data store cannot be backed up.

Specifying PL/SQL connection attributes in a DSN

You can specify values for PL/SQL general connection attributes. For a complete list of PL/SQL connection attributes, see "Data Store Attributes" in the *Oracle TimesTen In-Memory Database Reference*.

The following are a few of the PL/SQL connection attributes:

- PLSCOPE_SETTINGS - Controls whether the PL/SQL compiler generates cross-reference information.
- PLSQL_OPTIMIZE_LEVEL - Sets the optimization level that is used to compile PL/SQL library units.
- PLSQL_MEMORY_ADDRESS - Specifies the virtual address, as a hexadecimal value, at which the PL/SQL shared memory segment is loaded into each process that uses the TimesTen direct drivers. This memory address must be identical in all connections to your database and in all processes that connect to your database.
- PLSQL_MEMORY_SIZE - Determines the size, in megabytes, of the PL/SQL shared memory segment.

This example creates the `PLdsn` DSN, enables PL/SQL and sets the PL/SQL shared memory segment size to 32 MB. enables cross references.

```
[PLdsn]
Datastore=/users/user1/PLdsn
PermSize=32
DatabaseCharacterSet=AL32UTF8
ConnectionCharacterSet=AL32UTF8
PLSQL=1
PLSQL_MEMORY_SIZE=32
```

For more information about these attributes, see "Data Store Attributes" in the *Oracle TimesTen In-Memory Database Reference* for information about PL/SQL connection attributes. For more examples, see "PL/SQL connection attributes" in *Oracle TimesTen In-Memory Database PL/SQL Developer's Guide*.

Creating multiple DSNs to a single data store

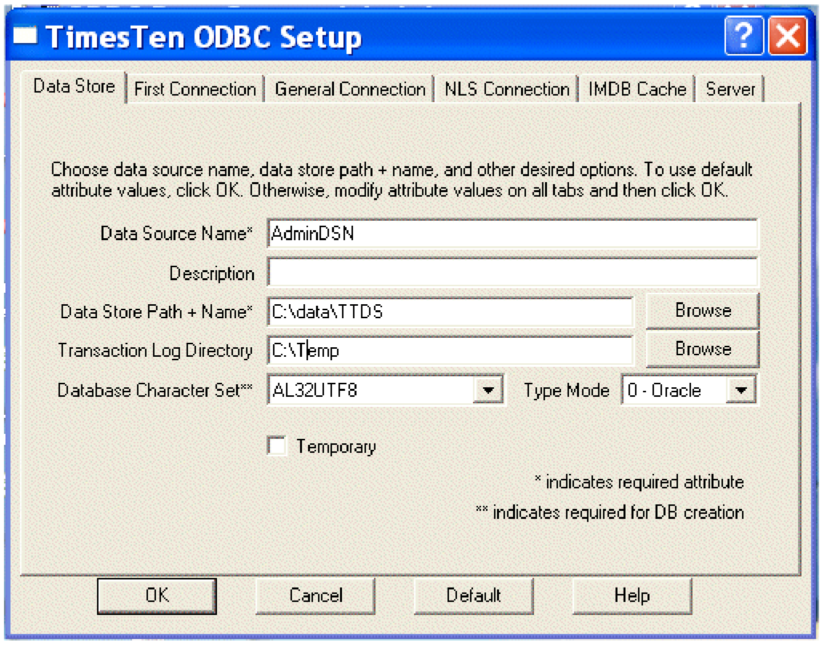
You can create two or more DSNs that refer to the same data store but have different connection attributes.

This example creates two DSNs, `AdminDSN` and `GlobalDSN`. The DSNs are identical except for their connection character sets. Applications that use the US7ASCII character set can connect to the TTDS data store by using `AdminDSN`. Applications that use multibyte characters can connect to the TTDS data store by using `GlobalDSN`.

WINDOWS

For Windows, use the ODBC Data Source Administrator to define AdminDSN as shown in Figure 1–9. `AdminDSN` is created with the AL32UTF8 database character set. Figure 1–10 shows that US7ASCII is the connection character set for `AdminDSN`.

Figure 1–9 Creating AdminDSN using TTDS data store



TimesTen ODBC Setup

Data Store | First Connection | General Connection | NLS Connection | IMDB Cache | Server

Choose data source name, data store path + name, and other desired options. To use default attribute values, click OK. Otherwise, modify attribute values on all tabs and then click OK.

Data Source Name*	AdminDSN
Description	
Data Store Path + Name*	C:\data\TTDS
Transaction Log Directory	C:\Temp
Database Character Set**	AL32UTF8
Type Mode	0 - Oracle

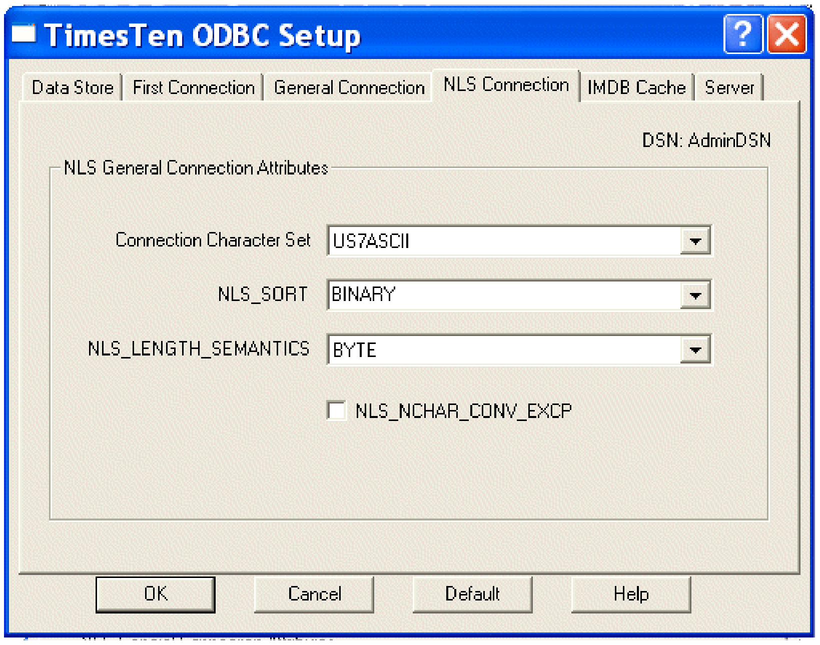
☐ Temporary

* indicates required attribute

** indicates required for DB creation

OK | Cancel | Default | Help

Figure 1–10 Setting the connection character set for AdminDSN



TimesTen ODBC Setup

Data Store | First Connection | General Connection | NLS Connection | IMDB Cache | Server

DSN: AdminDSN

NLS General Connection Attributes

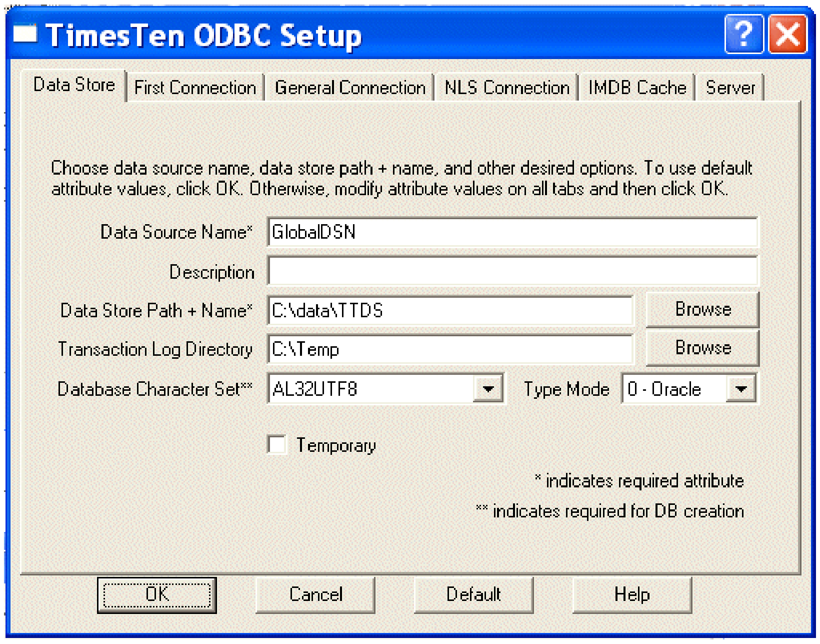
Connection Character Set	US7ASCII
NLS_SORT	BINARY
NLS_LENGTH_SEMANTICS	BYTE

☐ NLS_NCHAR_CONV_EXCP

OK | Cancel | Default | Help

`GlobalDSN` is also created with the AL32UTF8 database character set, as shown in Figure 1–11. Figure 1–12 shows that the connection character set for `GlobalDSN` is AL32UTF8.

Figure 1–11 Creating GlobalDSN using TTDS data store



Choose data source name, data store path + name, and other desired options. To use default attribute values, click OK. Otherwise, modify attribute values on all tabs and then click OK.

Data Source Name* GlobalDSN

Description

Data Store Path + Name* C:\data\TTDS

Transaction Log Directory C:\Temp

Database Character Set** AL32UTF8

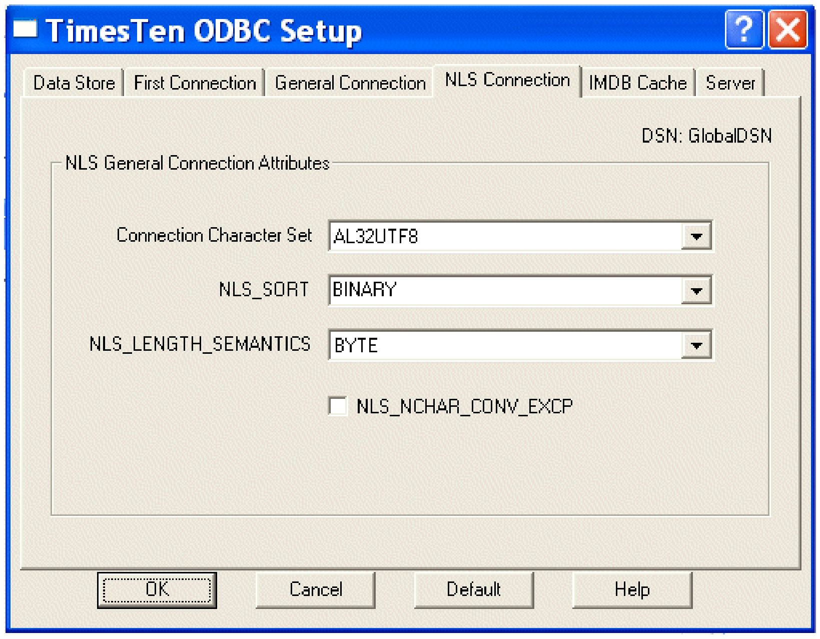
Type Mode 0 - Oracle

☐ Temporary

* indicates required attribute

** indicates required for DB creation

Figure 1–12 Setting the connection character set for GlobalDSN



DSN: GlobalDSN

NLS General Connection Attributes

Connection Character Set AL32UTF8

NLS_SORT BINARY

NLS_LENGTH_SEMANTICS BYTE

☐ NLS_NCHAR_CONV_EXCP

UNIX

The next example shows how to specify the DSNs on UNIX. It uses the TimesTen Data Manager ODBC driver for Solaris.

The text in square brackets is the data source name.

```
[AdminDSN]
Driver=install_dir/lib/libtten.so
Datastore=/data/TTDS
DatabaseCharacterSet=AL32UTF8
ConnectionCharacterSet=US7ASCII
```

```
[GlobalDSN]
Driver=install_dir/lib/libtten.so
DataStore=/data/TTDS
DatabaseCharacterSet=AL32UTF8
ConnectionCharacterSet=AL32UTF8
```

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 the `DataStore` attribute

On Microsoft Windows systems, the value of the `Driver` attribute should be the name of the TimesTen ODBC Driver. For example, the value can be TimesTen Data Manager 11.2.1.

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.

```
C:\ ttIsql
ttIsql <c> 1996-2009, Oracle. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.
Command> connect "Driver=TimesTen Data Managers 11.2.1; 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 "Data Store Attributes" in the *Oracle TimesTen In-Memory Database Reference* 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.

Modifying the size and unloading the data store is described in the following sections:

- Estimating and modifying the data partition sizes for the data store
- Unloading the data store from memory
- Monitoring PermSize and TempSize attributes

Estimating and modifying the data partition sizes for the data store

Procedures, tables, or rows cannot be created in the database if the permanent or temporary data partition is full. In order to have the correct size for your data store, set the appropriate size in the PermSize and TempSize connection attributes.

- PermSize connection attribute: The permanent data partition can be increased in size, but it cannot be decreased.
- TempSize connection attribute: The temporary data partition can be either increased or decreased in size for data stores that do not participate in replication.

To make size estimates, use the `ttSize` utility or run the application until you can make a reasonable estimate.

You must 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 + LogBufMB + 12MB overhead
```

Note: Additional shared segments may be created either for PL/SQL with the PLSQL_MEMORY_SIZE or for Client/Server with the `-serverShmSize` daemon.option.

When you are calculating the amount of PermSize to allocate, take into account that PL/SQL procedures, functions and packages occupy space in the permanent data partition. The amount of permanent data partition required by a stored PL/SQL unit depends on the size and complexity of the unit. Small procedures can take less than 3 KBytes, while larger ones can take considerably more. On average, reasonably complex units could be expected to use about 20 KBytes of permanent data partition space.

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

Unloading the data store from memory

Before you can change the size of either partition, you must first unload a data store from memory, close all active connections to the data store and set the RAM policy of the data store to `manual` or `inUse`.

- To unload the data store from memory, use the `ttStatus` utility to find processes connected to the data store and stop them. Once you have made the changes for data store size, reload it into memory.

  However, if the data store is configured for replication, stop the cache and replication agents, if they are running. 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 cache and replication agents.
- To set the RAM policy to `manual` or `inUse`, see "Specifying a RAM policy" on page 1-19 for information.

Monitoring PermSize and TempSize attributes

The TimesTen table SYS.MONITOR contains several columns that can be used to monitor usage of PermSize and TempSize. These columns include PERM_ALLOCATED_SIZE, TEMP_ALLOCATED_SIZE, PERM_IN_USE_SIZE, PERM_IN_USE_HIGH_WATER, and TEMP_IN_USE_SIZE. Each of these columns 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 the `ttBlockInfo` built-in procedure.

Receiving out-of-memory warnings

TimesTen provides two general connection attributes that determine when a low memory warning should be issued: PermWarnThreshold and TempWarnThreshold. 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 "Diagnostics through SNMP Traps" in the *Oracle TimesTen In-Memory Database Error Messages and SNMP Traps*.

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.

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:

- Migrate a data store between releases of TimesTen
- Migrate a data store between different hardware platforms
- Add rows of data to a table
- Take a snapshot of a data store and then restore it
- 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` 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.

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: Do 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 any existing backup 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	Comment
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 about using TimesTen's backup and restore facility, see "ttBackup" and "ttRestore" in *Oracle TimesTen In-Memory Database Reference*.

Working with the ODBC.INI file

This section includes the following topics:

- The user ODBC.INI file
- The system ODBC.INI file
- Searching for a DSN
- ODBC Data Sources
- Data Source specification
- ODBC.INI file example

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, containing 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 1-3.

The system ODBC.INI file

On UNIX, system DSNs are defined in the *`install_dir`*`/info/sys.odbc.ini` file. This file is referred to as the "system ODBC.INI file." A system DSN can be used any connection within the TimesTen installation.

Searching for a DSN

See "Searching for a DSN" on page 2-14 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 following 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 that connects to the data source.

Data Source specification

Each data source listed in the ODBC Data Sources section has its own data source specification. The data store specification for TimesTen Data Manager data stores has the format shown in Table 1–1.

Table 1–1 Data Source specification format

Component	Description
`[`*`data-source-name`*`]`	The *data-source-name* is required. It is the name of the data source, as specified in the ODBC Data Sources section of your `.odbc.ini` file.
`Driver=`*`driver-path-name`*	The TimesTen Data Manager driver that is linked with the data source. This is relevant when using a driver manager or for the server in a client/server scenario.
`DataStore=`*`data-store-path-name`*	The path name of the data store to access. The path name is required.
Optional attributes	See "Data Store Attributes" in *Oracle TimesTen In-Memory Database Reference* for information about attributes.

For example, the data source `sampledb_1121` could have a data source specification that includes the following:

```
[sampledb_1121]
Driver=install_dir/lib/libtten.so
DataStore=install_dir/info/DemoDataStore/sampledb_1121
...
```

The data store specification for TimesTen Client configurations has the format shown in Table 1–2.

Table 1–2 Data store specification for TimesTen Client configurations

Component	Description
`[`*`data-source-name`*`]`	The *`data-source-name`* is required. It is the name of the data source, as specified in the ODBC Data Sources section of your `.odbc.ini` file.
`TTC_Server=`*`server-name`*	The *`server-name`* is required. It is the DNS name, host name, IP address or shorthand name for the TimesTen Server.
`TTC_Server_DSN=`*`server-DSN`*	The *`server-DSN`* is required. It is the name of the data source to access on the TimesTen Server.
`TTC_Timeout=`*`value`*	Client connection timeout value in seconds.

Note: Most TimesTen Data Manager attributes are ignored for TimesTen Client data stores.

For example, the client/server data source `sampledbCS_1121` that connects to `sampledb_1121` on the TimesTen Server `ttserver` could have a data source specification that includes the following:

```
[sampledbCS_1121]
TTC_Server=ttserver
TTC_SERVER_DSN=sampledb_1121
TTC_Timeout=30
```

ODBC.INI file example

The following example shows portions of a UNIX `.odbc.ini` file:

```
...
[ODBC Data Sources]
sampledb_1121=TimesTen 11.2.1 Driver
...

[sampledb_1121]
Driver=install_dir/lib/libtten.so
DataStore=install_dir/info/DemoDataStore/sampledb_1121
PermSize=40
TempSize=32
PLSQL=1
DatabaseCharacterSet=US7ASCII
...

#######################################################################
# This following sample definitions should be in the .odbc.ini file
# that is used for the TimesTen 11.2.1 Client.
# The Server Name is set in the TTC_SERVER attribute.
# The Server DSN is set in the TTC_SERVER_DSN attribute.
########################################################################

[ODBC Data Sources]
sampledbCS_1121=TimesTen 11.2.1 Client Driver
...
```

```
[sampledbCS_1121]
TTC_SERVER=localhost
TTC_SERVER_DSN=sampledb_1121
...
```

2 Working with the TimesTen Client and Server

To access TimesTen data stores on remote machines, applications use the TimesTen Client driver to communicate with the TimesTen Server process. 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.

The TimesTen Server is a process that runs on a server machine. You can install the TimesTen Client on a separate or the same machine as the TimesTen Server. If you install the TimesTen Client on the same machine as the TimesTen Server, you can use them to access TimesTen data stores on the local machine. For example, this is useful when you want a 32-bit application to access a 64-bit data store on the same machine, for platforms that support both 32-bit and 64-bit applications.

Note: You can create a client/server connection between any combination of platforms that TimesTen supports.

You can link a client either directly with the TimesTen Client driver or with a driver manager. The latter is typical for Windows, which includes a driver manager, but not for UNIX. TimesTen supplies a driver manager for either Windows or UNIX with the Quick Start sample applications. Note that there are performance considerations in using a driver manager.

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

The main topics in this chapter are:

- Restrictions on Client/Server communication
- Client/Server communication
- Configuring TimesTen Client and Server
- Running the TimesTen Server daemon
- TimesTen Server connection attributes
- Defining Server DSNs
- TimesTen Client connection attributes

- Creating and configuring Client DSNs on Windows
- Creating and configuring Client DSNs on UNIX
- Accessing a remote data store on UNIX
- Working with the TTCONNECT.INI file

Restrictions on Client/Server communication

The following are the restrictions on client/server communication:

- XLA cannot be used over a client/server connection.
- On UNIX, some TimesTen utilities only work over direct connections, such as `ttAdmin`, `ttRepAdmin`, and `ttBackup`. The utilities that can be executed over a client connection on the UNIX platform are named with a suffix of `CS`, such as `ttIsqlCS`, `ttBulkCpCS`, `ttMigrateCS` and `ttSchemaCS`. These utilities have been linked with the ODBC driver manager, so they can be used with both client/server and direct linked drivers. Each utility listed in the *Oracle TimesTen In-Memory Database Reference* provides the name of the client/server version of that utility, if there is one.
- The `ttCacheUidPwdSet` built-in cannot be used over a client connection.
- You cannot connect with an external user defined on one host to a TimesTen data source on a remote host. There are no restrictions for connecting with internal users.
- Internal users can only be created or altered over a direct connection and not over a client connection to a TimesTen database. Thus, you can only execute the CREATE USER or ALTER USER statements using a direct connection to the TimesTen database. Once created, the user that connects from the client to the server must be granted the CREATE SESSION privilege or the connection will fail. For more information on how to create the user on the TimesTen database and how the administrator grants the CREATE SESSION privilege, see "Creating or identifying users to the database" on page 4-2 and "Granting privilege to connect to the database" on page 4-10.

Client/Server communication

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 reserve pool of Server processes. See "Prespawning TimesTen Server processes" on page 3-8 for details.

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

- TCP/IP Communication
- Shared memory communication
- UNIX domain socket communication

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 3-10. You must also define the Network Address of the logical server as `ttShmHost`. See "Creating and configuring Client DSNs on Windows" on page 2-6 or "Creating and configuring Client DSNs on UNIX" on page 2-13.

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 error.

UNIX domain socket communication

On 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. To use domain sockets, you must define the Network Address of the logical server as `ttLocalHost`. See "Creating and configuring Client DSNs on Windows" on page 2-6 or "Creating and configuring Client DSNs on UNIX" on page 2-13 for more information.

Configuring TimesTen Client and Server

Note: Before configuring the TimesTen Client and Server, read "TimesTen ODBC and JDBC drivers" on page 1-1 and "Data source names" on page 1-3.

The following sections describe how to connect a TimesTen application to a TimesTen data store using TimesTen Client and Server:

- Configuring client/server of the same TimesTen release
- Configuring cross-release Client/Server

Configuring client/server of the same TimesTen release

Perform the following to configure TimesTen when the client and server are of the same TimesTen release:

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*.

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. If you are using JDBC to connect to the database, install the Java Developer's Kit (JDK) on the client machine, where the Java application will be running and set up the environment variables, such as CLASSPATH and LIBRARY PATH, on the client machine. See "Setting the Java environment variables" in the *Oracle TimesTen In-Memory Database Java Developer's 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 2-5.
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 2-13 and "Creating and configuring Client DSNs on Windows" on page 2-6.
6. For OCI and Pro*C client/server connections, configure the application to use either `tnsnames.ora` or easy connect on the client as described in "Connecting to a TimesTen Datastore from OCI" in the *Oracle TimesTen In-Memory Database C Developer's Guide*.
7. Set TimesTen Client connection attributes. See "Data Store Attributes" in the *Oracle TimesTen In-Memory Database Reference*.
8. Link client/server applications as follows:
   - Link C and C++ client/server applications as described in "Linking options" in the *Oracle TimesTen In-Memory Database C Developer's Guide*.
   - Link OCI or Pro*C applications in the same manner as any OCI or Pro*C direct mode applications are linked, which is described in "TimesTen Support for Oracle Call Interface" and "TimesTen Support for the Oracle Pro*C/C++ Precompiler" in the *Oracle TimesTen In-Memory Database C Developer's Guide*.

Configuring cross-release Client/Server

A TimesTen Client can connect to a TimesTen Server from a different release and of a different bit level.

- A TimesTen Client 6.0 release may connect to a TimesTen Server 6.0 or 7.0 release of any bit level.

  Note: For a TimesTen 6.0 Client to connect successfully to a TimesTen Server release 7.0, the `-insecure-backwards-compat` option must be set in the `ttendaemon.options` file for the TimesTen Server.

- A TimesTen Client 7.0 or later release may connect to a TimesTen Server 7.0 or later release of any bit level.

The TimesTen Server loads a driver and a TimesTen database of its own release and bit level when the TimesTen Client connects to the Server. The TimesTen Data Manager must be installed on the Server host.

- If you are using a local client/server connection using UNIX Domain sockets through `ttLocalHost`, then the platforms for the client and the server must be UNIX. The bit level and the release level for the client and server hosts must be the same.
- If you are using a local client/server connection over a shared memory IPC using `ttShmHost`, then the platforms for the client and server can be either Windows or UNIX. The bit level may be different on the client and server hosts.

Running the TimesTen Server daemon

The TimesTen Server daemon is a subdaemon of the TimesTen daemon. If you have the TimesTen Server installed with no client installation, the TimesTern Server is automatically started and stopped when the TimesTen daemon or Service is started or stopped. You can explicitly start or shut down the daemon or Service with the `ttDaemonAdmin` utility.

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

The default ports for the 32-bit and 64-bit versions of TimesTen main and Server daemons are described in the "TimesTen Install" chapter in the *Oracle TimesTen In-Memory Database Installation Guide*. 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, 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 Service is run as user SYSTEM. On UNIX, the TimesTen Server is run as the instance administrator.

For instructions on modifying TimesTen Server daemon options, see "Modifying the TimesTen Server daemon options" on page 3-8.

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 the `syslog` facility.

See "Modifying informational messages" on page 3-5.

TimesTen Server connection attributes

By default, TimesTen creates only one connection to a Server per child process.

Because a TimesTen Server DSN corresponds to data stores that are accessed by a TimesTen Server daemon, a TimesTen Server DSN can be configured using regular connection attributes. In addition, you can specify connection attributes that allow you to specify multiple client connections to a single Server.

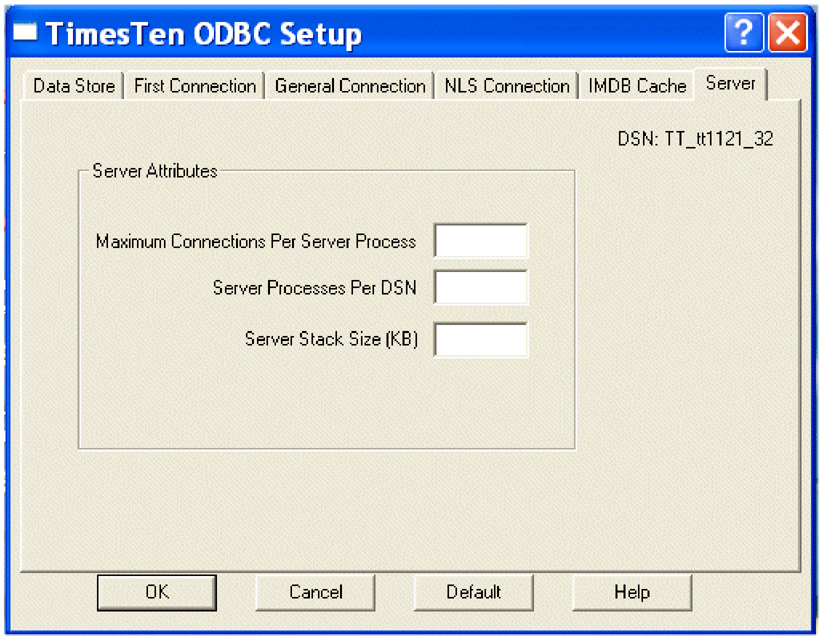
You can configure these attributes as part of the Server DSN definition or you can configure these attributes in the TimesTen daemon options file (`ttendaemon.options`). For a description of the TimesTen daemon options see "Specifying multiple connections to the TimesTen Server" on page 3-9.

The Server Connection attributes are MaxConnsPerServer, ServersPerDSN, and ServerStackSize. For a complete description of the TimesTen Server connection attributes, see "Data Store Attributes" in the *Oracle TimesTen In-Memory Database Reference*.

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 1-6 or "Creating a DSN on UNIX" on page 1-10. If you anticipate having more than one connection to the Server DSN, specify appropriate values for the Server attributes as needed. On Windows, these are specified on the **Server** tab.



TimesTen Client connection attributes

You can configure connection attributes used by the TimesTen Client driver as part of the Client DSN definition. Alternatively, 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 DataStore connection attribute is not allowed in the TimesTen Client connection string.

Use the TTC_SERVER_DSN attribute in either the connection string or the Client DSN for a client to specify which DSN it should use on the server. The server optionally uses the DRIVER attribute in the DSN to specify what driver to load.

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 "Data Store Attributes" in the *Oracle TimesTen In-Memory Database Reference*.

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
- Configuring automatic client failover
- 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:

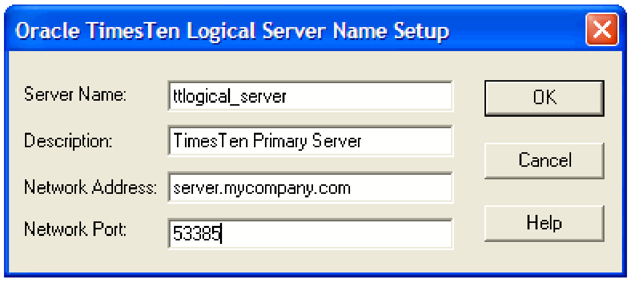
1. On the Windows Desktop, choose **Start > Settings > Control Panel > Administrative Tools > Data Sources (ODBC)**.

   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 11.2.1** 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:


Oracle TimesTen Logical Server Name Setup

Server Name: ttlogical_server
Description: TimesTen Primary Server
Network Address: server.mycompany.com
Network Port: 53385

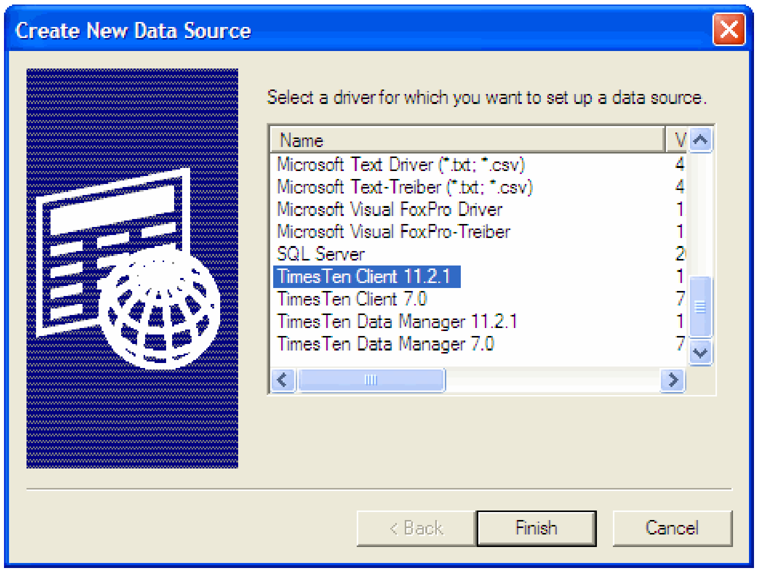
10. Click **OK**, then click **Close** in the TimesTen Logical Server List dialog to finish creating the logical server name.

Creating a Client DSN on Windows

WINDOWS

To define a TimesTen Client DSN:

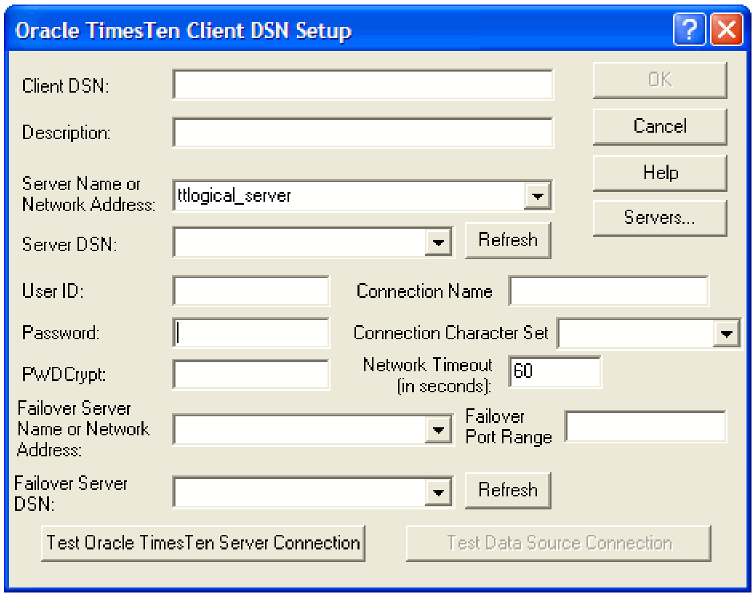
1. On the Windows Desktop, choose **Start > Settings > Control Panel > Administrative Tools > Data Sources (ODBC)**. This opens the ODBC Data Source Administrator.
2. Choose either **User DSN** or **System DSN**. For a description of User DSNs and System DSNs see "Data source names" on page 1-3.
3. Click **Add**. This opens the Create New Data Source dialog.


Create New Data Source

Select a driver for which you want to set up a data source.

Name	V
Microsoft Text Driver (*.txt; *.csv)	4
Microsoft Text-Treiber (*.txt; *.csv)	4
Microsoft Visual FoxPro Driver	1
Microsoft Visual FoxPro-Treiber	1
SQL Server	2
TimesTen Client 11.2.1	1
TimesTen Client 7.0	7
TimesTen Data Manager 11.2.1	1
TimesTen Data Manager 7.0	7

4. Choose **TimesTen Client 11.2.1**. Click **Finish**. This opens the Oracle TimesTen Client DSN Setup dialog.


Oracle TimesTen Client DSN Setup
Client DSN:
Description:
Server Name or Network Address:
ttlogical_server
Server DSN:
Refresh
User ID:
Connection Name
Password:
Connection Character Set
PWDCrypt:
Network Timeout (in seconds):
60
Failover Server Name or Network Address:
Failover Port Range
Failover Server DSN:
Refresh
Test Oracle TimesTen Server Connection
Test Data Source Connection
OK
Cancel
Help
Servers...

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 2-7.

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.

9. In the **Connection Character Set** field, choose a character set that matches your terminal settings or your data source. The default connection character set is

US7ASCII. For more information, see "ConnectionCharacterSet" in *Oracle TimesTen In-Memory Database Reference*.

Configuring automatic client failover

You can configure automatic client failover for data stores that have active standby pair replication schemes. This enables the client to fail over automatically to the server on which the standby data store resides.

See "Automatic client failover" in *Oracle TimesTen In-Memory Database C Developer's Guide* for information about connection option persistence after failover.

In the Oracle TimesTen Client DSN Setup dialog, complete the following fields:

1. In the **Failover 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. 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 the 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 2-7.
2. In the **Failover Server DSN** field, enter the Server DSN corresponding to the standby data store.
   - 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 **Failover 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.
3. Optionally, specify the **Failover Port Range** for the port for failover notifications. By default, TimesTen uses a port chosen by the operating system. To accommodate firewalls between the client and server systems, you can specify one value for the port number or a range of port numbers.

Setting the timeout interval and authentication

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

To set the timeout interval and authentication:

1. In the **User ID** field of the Oracle TimesTen Client DSN Setup dialog box, enter a user name that is defined on the server machine.
2. In the **Password** field, enter the password that corresponds to the user ID. Alternatively, you can enter an encrypted password in the **PwdCrypt** field.

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 time out. 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

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`.

Note: These examples reference the 32-bit sample DSNs. This is indicated by the extension `_32`. On 64-bit platforms, the sample DSNs are appended with `_64`.

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 2-5, verify that the Server DSN, `RunData1121_32`, 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 `53385`. This is the default port number for the TimesTen Server on 32-bit platforms for TimesTen Release 11.2.1. This value should correspond to the value displayed by `ttStatus` in Step 1.

   See "Creating and configuring a logical server name" on page 2-7 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, `RunData_tt1121_32`. In the TimesTen Client DSN Setup dialog, enter the following values:

- In the **Client DSN** field, enter `RunDataCS_tt1121_32`.
- 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 `RunData1121_32`.

5. Run the client application from the machine `client.mycompany.com` using the Client DSN, `RunDataCS_tt1121`. The example below uses the `ttIsqlCS` program installed with TimesTen Client.

```
ttIsqlCS connStr "DSN=RunDataCS_tt1121_32"
```

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

Consider that the Network Address of the TimesTen Server is `server.mycompany.com` and the Server is listening on Port 53385. 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 `53385` 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=53385"
```

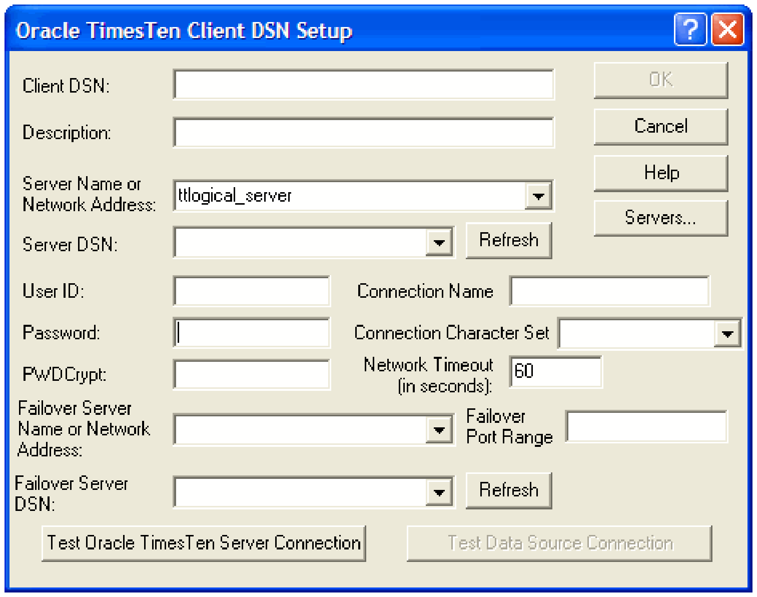
3. Alternatively, define the Server in the connection string. In this case you do not need to define a Client DSN, nor a logical server name.

```
ttIsqlCS -connStr "TTC_Server=server.mycompany.com;
TTC_Server_DSN=Server_DSN; TCP_Port=53385"
```

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.



Server Name or Network Address: ttlogical_server

Network Timeout (in seconds): 60

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 DSN or the system ODBC.INI file for system DSNs. For a description of user and system DSNs, see "Data source names" on page 1-3.

This section includes the following topics:

- Searching for a DSN
- Creating and configuring a logical server name
- Creating a Client DSN

- Configuring automatic client failover

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. The `sys.odbc.ini` file in the daemon home directory, if the SYSODBCINI environment variable is not set.

Creating and configuring a logical server name

Define logical server names 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`
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`

TimesTen searches for the logical server in this order:

1. In the file specified by the SYSTTCONNECTINI environment variable, if it is set
2. In the *`daemon_home_dir`*`/sys.ttconnect.ini` file

Example 2–1 Defining a logical server name

This example from a TTCONNECT.INI file defines a logical server name, `ttserver_logical`, for a TimesTen Server daemon running on the machine `server.mycompany.com` and listening on port `53385`. The instance name of the TimesTen installation is `tt1121`.

```
[ttserver_logical]
Description=TimesTen Server 11.2.1
Network_Address=server.mycompany.com
TCP_Port=53385
```

Example 2–2 Using UNIX domain sockets for communication

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_tt1121_32]
Description=
  Local TimesTen Server TimesTen release 11.2.1 through domain sockets
Network_Address=ttLocalHost
TCP_PORT=53385
```

Example 2–3 Configuring shared memory for inter-process communication

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_tt1121]
Description= Local TimesTen Server TimesTen release 11.2.1 through shared memory
Network_Address=ttShmHost
TCP_PORT=53385
```

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_tt1121_32` 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_tt1121_32=TimesTen Client 11.2.1
```

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_tt1121_32`.

```
[RunDataCS_tt1121_32]
TTC_Server=ttserver_logical
TTC_Server_DSN=RunData_tt1121_32
```

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

Configuring automatic client failover

You can configure automatic client failover for data stores that have active standby pair replication schemes. This enables the client to fail over automatically to the server on which the standby data store resides.

In the ODBC.INI file, set TTC_SERVER2 to the server on which the standby data store resides. Set TTC_SERVER_DSN2 to the name of the standby data store.

For example:

```
[MYDSN FAILOVER]
TTC_SERVER=localhost
TTC_SERVER_DSN=MYDSN
TTC_Timeout=60000
ConnectionCharacterSet=AL32UTF8
TTC_SERVER2=localhost
TTC_SERVER_DSN2=MYDSNSTANDBY
```

Accessing a remote data store on UNIX

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_tt1121_32` 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 `tt1121_32`.

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_tt1121_32` exists in the system ODBC.INI file on `server.mycompany.com`.

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

   ```
   [RunData_tt1121_32]
   Driver=install_dir/lib/libtten.so
   DataStore=install_dir/server/RunData_tt1121_32
   ```

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=53385
   ```

   See "Creating and configuring Client DSNs on UNIX" on page 2-13 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_tt1121_32**.

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

   ```
   [RunDataCS_tt1121_32]
   TTC_SERVER=ttserver_logical
   TTC_SERVER_DSN=RunData_tt1121_32
   ```

   See "User and system DSNs" on page 1-4 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_tt1121_32`. The example below uses the `ttIsql` program that is installed with TimesTen Client.

```
ttIsqlCS -connStr "DSN=RunDataCS_tt1121_32"
```

The next 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 `53385`. 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 `53385` as the Network Port. Define a Client DSN with `logical_server` as the server name, `Server_DSN` as the Server DSN. 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=53385"
   ```

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=53385"
   ```

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 53385
     ```

   - 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 3-8.

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.

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 `install_dir/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 shown in Table 2–1.

Table 2–1 TimesTen Server format in the TTCONNECT.INI file

Component	**Description**
[*ServerName*]	Short name of the TimesTen you wish to define
`Description=`*`description`*	Description of the TimesTen Server
`Network_Address=`*`network-address`*	The DNS name, host name or IP address of the machine on which the TimesTen Server is running.
`TCP_Port=`*`port-number`*	The TCP/IP port number where the TImesTen Server is running. Default for TimesTen release 11.2.1 is 53385 for 32-bit platforms and 53389 for 64-bit platforms.

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=53385
```

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:

```
[LocalHost1121]
Description=Shm TimesTen Client/Server
Network_Address=ttLocalHost
TCP_Port=53385
```

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:

```
[ShmHost1121]
Description=Shm TimesTen Client/Server
Network_Address=ttShmHost
TCP_Port=53385
```

3 Working with the Oracle TimesTen Data Manager Daemon

The Oracle TimesTen Data Manager daemon, which is the Oracle TimesTen Data Manager service on Windows, starts when TimesTen is installed. The daemon operates continually in the background.

The TimesTen daemon performs the following functions:

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

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
- Starting and stopping the daemon on UNIX
- Shutting down a TimesTen application
- Managing TimesTen daemon options
- Managing TimesTen Client/Server daemon options

Starting and stopping the Oracle TimesTen Data Manager Service on Windows

WINDOWS

The Oracle TimesTen Data Manager service starts 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 the Windows Administrative Tools as follows:

1. Open Administrative Tools:

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

2. Double-click **Services**. All currently available services are displayed.
3. Select **TimesTen Data Manager 11.2.1**, 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 the instance administrator to start and stop the TimesTen daemon.

The instance administrator must manually start and stop the daemon, after each system reboot, 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.

User `root` can start the daemon by executing the daemon startup script. The following table shows the location of the daemon startup script by platform.

Platform	**Location of daemon startup script**
Linux	`/etc/init.d/tt_`*`instance_name`*
Solaris	`/etc/init.d/tt_`*`instance_name`*
HP-UX	`/sbin/init.d/tt_`*`instance_name`*
AIX	`/etc/init.d/tt_`*`instance_name`*

Shutting down a TimesTen application

A TimesTen application consists of a data store which has been allocated to shared memory, some client connections, and possibly replication and cache agents for communication with other TimesTen or Oracle databases.

To shut down a TimesTen application, complete the following tasks:

1. Disconnect all client processes gracefully.
2. Shut down all replication and cache agents.
3. Unload the data store from shared memory if it was manually loaded.
4. Stop the TimesTen daemon.

Managing TimesTen daemon options

The `ttendaemon.options` file contains TimesTen daemon options. During installation, the installer sets some of these options to correspond to your responses to the installation prompts.

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:

```
install_dir/info/
```

The features that the `ttendaemon.options` file controls are as follows:

- The network interfaces on which the daemon listens
- The minimum and maximum number of TimesTen subdaemons that can exist for the TimesTen instance
- Whether or not the TimesTen Server daemon is started
- Whether or not you use shared memory segments for client/server inter-process communication
- The number of Server processes that are prespawned on your system
- The location and size of support and user logs
- Backward compatibility
- The maximum number of users for a TimesTen instance
- Data access across NFS mounted systems. This is for Linux only.
- The TNS_ADMIN value for the Oracle Database. This option cannot be modified in this file.

Use the `ttmodinstall` utility to make changes to the `ttendaemon.options` file for most commonly changed options. See "`ttmodinstall`" in *Oracle TimesTen In-Memory Database Reference*. If you cannot use `ttmodinstall` to change a particular option and must modify the `ttendaemon.options` file directly, stop the TimesTen daemon before you change the file. Restart the TimesTen daemon after you have finished changing the file. To change 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
- Modifying informational messages
- Changing the allowable number of subdaemons
- Allowing data store access over NFS-mounted systems
- Enabling Linux large page support
- Shared memory daemon option for HP-UX ccNUMA systems

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.

The `-listenaddr` parameter exists for situations where a server has multiple network addresses and multiple network cards. In this case it is possible to limit the network addresses on which the TimesTen daemon is listening to a subset of the server's network addresses. This is done by making entries only for those addresses on which the daemon listens. These possibilities exist:

- Given a situation where a server has a "public" network address that is accessible both inside and outside the local network and a "private" address that is accessible only within the local network, adding a `-listenaddr` entry containing only the private address blocks all communications to TimesTen coming on the public address.
- By specifying only the local host, the TimesTen main daemon can be cut off from all communications coming from outside the server and communicate only with local clients and subdaemons.

There is no relationship between TimesTen replication and the `-listenaddr` parameter and there is no requirement for enabling the `-listenaddr` parameter when replication is enabled. If replication is going to be used in an environment where `-listenaddr` is enabled, then the replication nodes need to know the allowable network addresses to use. However, if no `-listenaddr` parameter is enabled replication still works.

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, TimesTen always listens on the loopback address.

Listening on IPv6

By default, TimesTen uses the IPv4 protocol. To enable the daemon to listen on IPv6, you must enter on a separate line in the `ttendaemon.options` file:

```
-enableIPv6
```

and

```
-listenaddr6 address
```

- You can specify an IPv6 address with the `-listenaddr6` option to enable IPv6.
- Specifying the `-enableIPv6` option with one or more `-listenaddr` or `-listenaddr6` options adds the IPv6 loopback interface to the list.
- If you specify the `-enableIPv6` option without specifying any addresses with the `-listenaddr` or `-listenaddr6` options, then the daemon listens on any IPv6 interface as well as any IPv4 interface.

The address specified with this option can be either a host name or a numerical IP address. See "Determining the daemon listening address" on page 3-3 for specifics on the `-listenaddr` option

If one or more `-listenaddr` options are provided, the daemons listen on the specified IPv4 interfaces, with the IPv4 loopback address being added to the list if not specified. If only `-enableIPv6` is specified, the daemons listen on both the IPv4 ANY interface and the IPv6 ANY interface.

You can specify both `-listenaddr` and `-listenaddr6` options. If you specify one or more `-listenaddr6` options, the daemons listen on the specified IPv4 or IPv6 interfaces, with both the IPv4 and IPv6 loopback interfaces being added if not specified. If the name resolver returns multiple IPv4 and/or IPv6 addresses for a name, the daemons listen on all of the names.

Modifying 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.

By default, informational messages are stored in:

- A user error log that contains information you may need to see. Generally, these messages contain information on actions you may need to take.
- A support log containing everything in the user error log plus information of use by TimesTen Customer Support.

The following options specify the locations and size of the support and user logs, as well as the number of files to keep stored on your system.

Option	Description
`-supportlog` *path* `-f` *path*	Specifies the location for the support log file. The default file is *daemon_home*/`ttmesg.log`
`-maxsupportlogfiles` *num*	The TimesTen main daemon automatically rotates the files once they get to a specific size. This option specifies the number of support log files to keep. The default is 10.
`-maxsupportlogsize` *nBytes*	Specifies the maximum size of the support log file. The default is 1MB.
`-userlog` *logfile* or `-userlog [syslog]`	Specifies the location and name of the user log file. The default file is *daemon_home*/`tterrors.log`. You may specify [`syslog`] on UNIX systems as the path or the Event Log on Windows, in which case the output is sent to the system [`syslog`] or Event Log.
`-maxuserlogfiles` *num*	The TimesTen main daemon automatically rotates the files once they get to a specific size. This option specifies the number of user log files to keep. The default is 10.
`-maxuserlogsize` *nBytes*	Specifies maximum size of the user log. Default is 1MB.
`-showdate`	On UNIX systems only, indicates that the date should be prepended to all messages

WINDOWS

If you have specified the Event Log as the location for your log messages, to view them 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**, up or down arrows, depending on your version of Windows.

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

To specify the `syslog` facility used to log TimesTen Daemon and subdaemon messages on UNIX, 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 `-verbose` in the `ttendaemon.options` file.

Changing the allowable number of subdaemons

TimesTen uses subdaemons to perform the following:

- Manage data stores.
- Flush the log buffer to disk.
- Perform periodic checkpoints.
- Implement the aging policies of various tables.
- Find and break deadlocks.
- Rollback transactions for abnormally terminated direct-mode applications.
- Perform required background processing for the database.

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.

Allowing data store access over NFS-mounted systems

By default, TimesTen systems cannot access data across NFS-mounted systems. On Linux x86 64-bit systems, you can access checkpoint and log files on NFS-mounted systems.

To enable data access on NFS-mounted systems, on a separate line of the `ttendaemon.options` file, add:

```
-allowNetworkFiles
```

Note: TimesTen does not support the storage of trace files or user and support logs across NFS-mounted systems

---

Enabling Linux large page support

LINUX

To enable Linux large page support on TimesTen, on a separate line of the `ttendaemon.options` file, add:

```
-linuxLargePageAlignment Size_in_MB
```

The *`Size_in_MB`* is the `Hugepagesize` value in `/proc/meminfo`, specified in MB instead of KB.

Shared memory daemon option for HP-UX ccNUMA systems

HP-UX

HP-UX ccNUMA systems have non-uniform memory latency depending on the data location. Accessing data in a remote cell takes longer than accessing data in a local cell. To ensure the best results for TimesTen operations, set the IPC_MEM_LOCAL and confine the TimesTen processes to the local cell.

To set the locality hint for the shared memory segment, on a separate line of the `ttendaemon.options` file, add:

```
-shmLocalityHint locality_hint
```

Legal values for *`locality_hint`* are:

- IPC_MEM_LOCAL
- IPC_MEM_INTERLEAVED
- IPC_MEM_FIRST_TOUCH
- IPC_MEM_STRIPED

Only one value string can be specified at a time. If specified, TimesTen attempts to create the shared memory segment for all data stores in the instance with the appropriate locality hint.

Note: This option only takes effect if the instance administrator has permission to access the memory resource.

The semantics of the hints are described in the man page for `shmget()`. The default behavior is to create the segment without the hint. Expect the default behavior if the daemon option is not specified or if it is specified incorrectly. To see whether a segment has been created with the hint, use the HP-UX `pstat()` facility. See the HP-UX man page for `pstat()`.

Managing TimesTen Client/Server daemon options

This section includes the following topics:

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:

- 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 reserve 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 to create *`number`* processes. If this option is not specified, no processes are prespawned and kept in the reserve pool.

When a new connection is requested, if there are no items in the server pool, a new process is spawned, as long as you have not met the operating system limit.

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, even if there are no processes remaining in the reserve pool.

Changes to the TimesTen Server daemon take effect when the Server is restarted.

Specifying multiple connections to the TimesTen Server

By default, TimesTen creates only one connection to a Server per child process. You can set multiple connects to a single TimesTen Server, either by using the Server connection attributes described in the *Oracle TimesTen In-Memory Database Reference* or by setting the TimesTen daemon options described in this section. These options allow you to set the number of connections to a TimesTen server, the number of servers for each DSN and the size of each connection to the server.

Note: In the case that you have set both the Server connection attributes and these daemon options, the value of the Server connection attributes takes precedence.

Configuring the maximum number of client connections per child server process

To run a child server process in multithreaded mode so that a single server process can service multiple client connections to a data store, add the following line to the `ttendaemon.options` file:

```
-maxConnsPerServer NumberOfClientConnections
```

The possible values of *`NumberOfClientConnections`* range from 1 to 2047, inclusive. The default value is 1, which indicates that the child server process runs in multi-process mode and, therefore, can service only one client connection.

Configuring the desired number of child server processes spawned for a server DSN

To specify the desired number of child server processes to be spawned for a particular server DSN, add the following line to the `ttendaemon.options` file:

```
-serversPerDSN NumberOfChildServerProcesses
```

The possible values of *`NumberOfChildServerProcesses`* range from 1 to 2047, inclusive. The default value is 1.

Client connections to a particular server DSN are evenly distributed in round-robin fashion to the child server processes that are spawned and assigned to the DSN. The number of child server processes assigned to the server DSN is greater than *`NumberOfChildServerProcesses`* if the number of client connections to the DSN is greater than the maximum number of client connections per child server process multiplied by the desired number of child server processes spawned for a server DSN.

Configuring the thread stack size of the child server processes

To set the size of the child server process thread stack for each client connection, add the following line to the `ttendaemon.options` file:

```
-serverStackSize ThreadStackSize
```

`ThreadStackSize` is specified in KB. The default is 128 KB on 32-bit systems and 256 KB on 64-bit systems. The *`ThreadStackSize`* setting is ignored if the maximum number of client connections per child server process is 1 because the sole client connection will be serviced by the main thread of the child server process.

Note: 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 *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:

- The maximum size allowed is 1 gigabyte.

- 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. Modify the value of `-serverShmSize` in the `ttendaemon.options` file.
2. Use the `ttDaemonAdmin` utility to restart the TimesTen Server. Only the instance administrator can 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.

Communicating with older releases of TimesTen

To allow client/server and replication communication between a TimesTen Client Release 6.0 and a TimesTen Server Release 7.0, add the following on a separate line in the `ttendaemon.options` file:

```
-insecure-backwards-compat
```

Note: By using the `-insecure-backwards-compat` option, you are reducing the security of your TimesTen installation. You should only use the option when absolutely necessary, and discontinue its use when you no longer need it.

4

Managing Access Control

The TimesTen Access Control provides authentication for each user and authorization for all objects in the database. Authentication is provided with the correct user password. Management of authorization for all objects in the database is provided by granting appropriate privileges to specific users.

The following sections describe the TimesTen authentication and authorization:

- Managing users to control authentication
- Providing authorization to objects through privileges

Managing users to control authentication

For users to access and manipulate data within the database, you must create users and provide appropriate passwords. When you create a user, you should also grant the appropriate privileges for connecting to the database or for access to objects in the database. For more information on granting privileges, see "Providing authorization to objects through privileges" on page 4-4.

The following sections describe how to create and manage your users:

- Overview of users
- Creating or identifying users to the database
- Changing the password of the internal user
- Dropping users from the database

Overview of users

There are three types of users in the TimesTen database:

- Instance administrator: The instance administrator is the user who installed the TimesTen instance. This user has full privileges for everything within the TimesTen instance. For information on creating this user, see "TimesTen Installation" in the *Oracle TimesTen In-Memory Database Installation Guide*.

  Note: In addition to the instance administrator, there are four system users created during the TimesTen install. These system users are used internally by TimesTen as follows: SYSTEM for internal use, SYS for system objects, GRID for cache grid objects and TTREP for replication objects.

- Internal user: An internal user is created within TimesTen for use within the TimesTen database. An internal user authenticates with a password for a particular database in which it was defined.

  TimesTen user names are case-insensitive, of type TT_CHAR and limited to 30 characters. For details on all user naming conventions, see "Names and parameters" in the *Oracle TimesTen In-Memory Database SQL Reference*.

  You can create an internal user with the CREATE USER statement, which is described in the *Oracle TimesTen In-Memory Database SQL Reference*.

- External user: An external user is created within the operating system. External users are assumed to have been authenticated by the operating system at login time, so there is no stored password within the database. One cannot connect as an external user from a different host from which the TimesTen database is installed. On the same host, we use the operating system credentials of the client to enable the client to connect as that particular external user. For example, if an external user logs into the UNIX system, they can connect to the TimesTen database without specifying a password since they already provided it during the login, as long as the external user has been granted the correct privileges. The external user must also be in the TimesTen users group and have the correct permissions granted to it, as described in the *Oracle TimesTen In-Memory Database Installation Guide*.

  You cannot connect with an external user defined on one host to a TimesTen data source on a remote host. External users can only be used to connect to the local TimesTen data source, because the local operating system authenticates the external user.

  While the external user is created within the operating system, you still need to identify the user to the database as an external user with the IDENTIFIED EXTERNALLY clause of the CREATE USER statement. For details on this SQL statement, see "CREATE USER" in the *Oracle TimesTen In-Memory Database SQL Reference*.

  UNIX external user names are case sensitive. Windows external user names are not. When connecting from UNIX platforms, TimesTen automatically converts the external user name to upper case, rendering it case insensitive.

  If you do not want to use cleartext passwords to log into TimesTen, then use the `PWDCrypt` attribute to create a hash of the password. The only reason to use this attribute is if the password is used for logging into other entities, such as an Oracle Database. The `PWDCrypt` version of the password can always be used to connect to TimesTen, but you cannot convert it back to the original password in order to connect to Oracle.

Note: Both the instance administrator and all external users must be in the TimesTen users group specified during the install. For more details, see "TimesTen Installation" in the *Oracle TimesTen In-Memory Database Installation Guide*.

Creating or identifying users to the database

Only the instance administrator or a user with the ADMIN privilege can create the internal user or identify the external user with the `CREATE USER` statement. For security purposes, you can only create or alter the internal user with the CREATE USER or ALTER USER statements using a direct connection to the TimesTen database. Thus, executing CREATE USER or ALTER USER from a client-server application or through passthrough execution is not allowed. You can use the ALTER USER statement to change a user from an internal to an external user or from an external to an internal user. The full syntax for the `CREATE USER` statement is detailed in the "SQL Statements" chapter in the *Oracle TimesTen In-Memory Database SQL Reference*.

Note: For details on a user with the ADMIN privilege, see "Granting administrator privileges" on page 4-10.

To create an internal user, provide the username and password in the CREATE USER statement. The following example creates the internal user `TERRY` with the password `"secret"`:

```
CREATE USER TERRY IDENTIFIED BY "secret";
User created.
```

To identify an external user, provide the username in the `CREATE USER IDENTIFIED EXTERNALLY` statement. The following example identifies the external user `PAT` to the TimesTen database:

```
CREATE USER PAT IDENTIFIED EXTERNALLY;
User created.
```

To change the external user `PAT` to an internal user, perform the following `ALTER USER` statement:

```
ALTER USER PAT IDENTIFIED BY "secret";
```

To change the internal user `PAT` to an external user, perform the following `ALTER USER` statement:

```
ALTER USER PAT IDENTIFIED EXTERNALLY;
```

You can see what users have been created by executing a SELECT statement on the following system views:

- SYS.ALL_USERS lists all users of the database that are visible to the current user.
- SYS.USER_USERS describes the current user of the database.
- SYS.DBA_USERS describes all users of the database. To perform a select statement on this view, you must have the appropriate privileges granted.

For example, to see the current user, perform the following:

```
SELECT * FROM sys.user_users;
< PAT, 4, OPEN, <NULL>, <NULL>, USERS, TEMP, 2009-02-25 12:00:17.027100, <NULL>,
<NULL> >
1 row found.
```

For more details on these views, see "System and Replication Tables" in the *Oracle TimesTen In-Memory Database SQL Reference*.

Changing the password of the internal user

Only the internal user has a password that can be modified within the database. A user can alter their own password. A user with the ADMIN privilege can alter the password of any user. These users can change the password with the IDENTIFIED BY clause of the ALTER USER statement.

For example, to change the password for internal user `TERRY` to `"12345"` from its current setting, perform the following:

```
ALTER USER TERRY IDENTIFIED BY "12345";
User altered.
```

Dropping users from the database

You cannot drop a user unless you have first deleted all objects owned by that user. You can drop any user created or identified in the database with the DROP USER statement.

The following DROP USER statement drops the user `TERRY` from the database:

```
DROP USER TERRY;
User dropped.
```

You cannot drop any user that owns objects or the instance administrator. The following error occurs if you try to drop the instance administrator:

```
drop user instadmin;
15103: System-defined users and roles cannot be dropped
The command failed.
```

Note: Currently, we do not support DROP USER CASCADE.

Providing authorization to objects through privileges

When multiple users can access database objects, authorization can be controlled to these objects with privileges. Every object has an owner. Privileges control if a user can modify an object owned by another user. Privileges are granted or revoked either by the instance administrator, a user with the ADMIN privilege or, for privileges to a certain object, by the owner of the object.

The following sections describe authorization to objects through the use of privileges:

- Privileges overview
- Granting or revoking system privileges
- Granting or revoking object privileges
- Granting or revoking multiple privileges with a single SQL statement
- Granting or revoking privileges for cache groups
- Viewing user privileges
- Privileges needed for utilities, built-in procedures and first connection attributes
- Privilege checking rules for parent-child tables

Privileges overview

TimesTen provides user authorization to objects in the database through privileges. Users must be granted privileges for access to database resources or objects. These privileges restrict what operations users may perform on those objects. A user has all privileges on all objects in their own schema, and these privileges cannot be revoked. A user can be granted privileges for objects in other users' schemas.

TimesTen evaluates each user's privileges when the SQL statement is executed. Each SQL statement can be executed by an arbitrary user. For example:

```
SELECT * from PAT.TABLE1;
```

If this statement is executed by Pat, then no extra privileges are necessary because Pat owns this object. However, if another user, such as Terry, executes this statement, then Terry must have be granted the SELECT privilege for PAT.TABLE1.

Privileges provide the following:

- Define what data users, applications, or functions can access or what operations they can perform.
- Prevent users from adversely affecting system performance or from consuming excessive system resources. For example, a privilege restricting the creation of indexes is provided not because of an authorization concern, but because it may affect DML performance and occupies space.

Some examples of privileges include the right to perform the following:

- Connect to the database and create a session
- Create a table
- Select rows from a table that is owned by another user
- Perform any cache group operation

In addition, a user may need certain privileges in order to perform the following:

- Execute certain TimesTen built-in procedures, which are documented in the *Oracle TimesTen In-Memory Database Reference*.
- Execute certain TimesTen command-line utilities, which are documented in the *Oracle TimesTen In-Memory Database Reference*.
- Initiate a connection with first connection attributes, which are documented in the *Oracle TimesTen In-Memory Database Reference*.
- The privilege required for executing each SQL statement is documented in the statement description in the *Oracle TimesTen In-Memory Database SQL Reference*

There are two levels of privileges:

- System privileges: These privileges enable system-wide functionality, such as access to all objects. Granting system privileges can enable a user to perform standard administrator tasks or access to objects in other users' schemas. These privileges extend beyond a single object. Restrict them only to trusted users.
- Object privileges: Each type of object has privileges associated with it.

A subset of these privileges are automatically granted to each user upon creation through the PUBLIC role. Privilege hierarchy rules apply to all privileges granted to a user.

Grant privileges to users so that they can accomplish tasks required for their job. We recommend that you are intentional about who you grant privileges, so that they have only the exact privileges that they need to perform necessary operations.

Privileges are checked at prepare time and when the statement is first executed for each SQL statement. Subsequent executions of that statement require further privilege checks only when a revoke operation is executed in the database.

System privileges

A system privilege enables a user the ability to perform system-level activities across multiple objects in the database. It confers the right to perform a particular operation in the database or to perform an operation on a type of object. For example, the privilege to create or modify an object in another user's schema in the database requires a system privilege to be granted to the user.

Only the instance administrator or a user with the ADMIN privilege can grant a system privilege to a user. The instance administrator always has full system and object privileges, which cannot be revoked at any time.

Note: The instance administrator can perform all operations. So, any operation that can be performed by a user with ADMIN privileges can also be performed by the instance administrator.

Some of the system privileges include ADMIN, SELECT ANY TABLE, CREATE SESSION and CREATE ANY SEQUENCE. For more details on granting or revoking system privileges, see "Granting or revoking system privileges" on page 4-9.

Object privileges

An object privilege enables a user to perform defined operations on a specific object. Separate object privileges are available for each object type.

Every object owner has access and full privileges to their own objects. A user does not have access to objects owned by other users unless explicitly granted access by the object's owner or by a user with ADMIN privilege. If the PUBLIC role has been granted access to a given object, then all database users have access to that object. A user with ADMIN privileges cannot revoke an owner's privileges on the owner's object.

Note: Some objects, such as cache group and replication objects, require system level privileges before a user can perform certain operations.

Object access control requires that a user either be the owner of an object or granted the appropriate object privilege to perform operations on the object. Object privileges are granted or revoked by the instance administrator, a user with the ADMIN privilege or the user who is the owner of the object.

For more details on granting or revoking object privileges, see "Granting or revoking object privileges" on page 4-12.

PUBLIC role

A role called PUBLIC is automatically created in each TimesTen database. By default, TimesTen grants specific privileges to this role. Every user created within the TimesTen database are granted each privilege that is granted to the PUBLIC role. That is, when the instance administrator or a user with the ADMIN privilege creates a user, the privileges associated with the PUBLIC role are granted to each of these users. Each subsequent privilege that is granted to the PUBLIC role is also automatically granted to all users simultaneously. A user with the ADMIN privilege can add or remove default privileges for all users by granting or revoking privileges from the PUBLIC role. When the user revokes a privilege from PUBLIC, it is revoked from each user, except for those users who have this privilege granted to them explicitly.

Note: The only exception to this behavior is that any privileges that were granted to PUBLIC by user SYS cannot be revoked. The privileges that were granted as part of database creation are shown when you execute the following SQL statement:

```
SELECT * FROM DBA_TAB_PRIVS WHERE GRANTOR = 'SYS'
```

In the following example, user Pat is granted the SELECT ANY TABLE privilege and PUBLIC is granted the SELECT ANY TABLE privilege. Then, all system privileges are displayed from the SYS.DBA_SYS_PRIVS view. For more information on this view, see "Viewing user privileges" on page 4-19. Revoking SELECT ANY TABLE from PUBLIC does not remove SELECT ANY TABLE from Pat, which is shown again through the SYS.DBA_SYS_PRIVS view.

```
Command> GRANT SELECT ANY TABLE TO PAT;
Command> GRANT SELECT ANY TABLE TO PUBLIC;
Command> SELECT * FROM SYS.DBA_SYS_PRIVS;
< SYS, ADMIN, NO >
< PUBLIC, SELECT ANY TABLE, NO >
< SYSTEM, ADMIN, NO >
< PAT, ADMIN, NO >
< PAT, SELECT ANY TABLE, NO >
5 rows found.
Command> REVOKE SELECT ANY TABLE FROM PUBLIC;
Command> select * from sys.dba_sys_privs;
< SYS, ADMIN, NO >
< SYSTEM, ADMIN, NO >
< PAT, ADMIN, NO >
< PAT, SELECT ANY TABLE, NO >
4 rows found.
```

If you must, you may create a database that grants the ADMIN privilege to PUBLIC. This grants the ADMIN privilege to all users who will then have unrestricted access to all database objects and be able to perform administrative tasks except for tasks that must be performed by the instance administrator. This is never recommended as a long-term approach, since it results in an insecure database. See "Data Store Upgrades" in the *Oracle TimesTen In-Memory Database Installation Guide* for full details on when and for what purposes to use this approach.

Note: For a full description of the default privileges assigned to the PUBLIC role, see "SQL Statements" in the *Oracle TimesTen In-Memory Database SQL Reference*.

The PUBLIC role also grants access to certain objects, system tables and views. By default, in a newly created TimesTen database, PUBLIC has SELECT and EXECUTE privileges on various system tables and views and PL/SQL functions, procedures and packages. You can see the list of privileges granted to PUBLIC, and subsequently all users, by querying the SYS.DBA_TAB_PRIVS view. In the following query, the privilege granted to PUBLIC is in the fifth column.

```
Command> DESC SYS.DBA_TAB_PRIVS;
View SYS.DBA_TAB_PRIVS:
  Columns:
    GRANTEE                         VARCHAR2 (30) INLINE
    OWNER                           VARCHAR2 (30) INLINE
    TABLE_NAME                      VARCHAR2 (30) INLINE
    GRANTOR                         VARCHAR2 (30) INLINE
    PRIVILEGE                       VARCHAR2 (40) INLINE NOT NULL
    GRANTABLE                       VARCHAR2 (3) INLINE NOT NULL
    HIERARCHY                       VARCHAR2 (3) INLINE NOT NULL
1 view found.

Command> SELECT * FROM SYS.DBA_TAB_PRIVS WHERE GRANTEE='PUBLIC';
< PUBLIC, SYS, TABLES, SYS, SELECT, NO, NO >
< PUBLIC, SYS, COLUMNS, SYS, SELECT, NO, NO >
< PUBLIC, SYS, INDEXES, SYS, SELECT, NO, NO >
< PUBLIC, SYS, USER_COL_PRIVS, SYS, SELECT, NO, NO >
< PUBLIC, SYS, PUBLIC_DEPENDENCY, SYS, SELECT, NO, NO >
< PUBLIC, SYS, USER_OBJECT_SIZE, SYS, SELECT, NO, NO >
< PUBLIC, SYS, STANDARD, SYS, EXECUTE, NO, NO >
< PUBLIC, SYS, UTL_IDENT, SYS, EXECUTE, NO, NO >
< PUBLIC, SYS, TT_DB_VERSION, SYS, EXECUTE, NO, NO >
< PUBLIC, SYS, PLITBLM, SYS, EXECUTE, NO, NO >
< PUBLIC, SYS, DBMS_OUTPUT, SYS, EXECUTE, NO, NO >
< PUBLIC, SYS, DBMS_SQL, SYS, EXECUTE, NO, NO >
< PUBLIC, SYS, DBMS_STANDARD, SYS, EXECUTE, NO, NO >
< PUBLIC, SYS, DBMS_PREPROCESSOR, SYS, EXECUTE, NO, NO >
< PUBLIC, SYS, UTL_RAW, SYS, EXECUTE, NO, NO >
< PUBLIC, SYS, DBMS_UTILITY, SYS, EXECUTE, NO, NO >
< PUBLIC, SYS, DBMS_RANDOM, SYS, EXECUTE, NO, NO >
...
57 rows found.
```

Privilege hierarchy rules

There is a hierarchy for all of the privileges. The higher level privileges confer related lower level privileges. For example, the ADMIN privilege confers all privileges. The SELECT ANY TABLE privilege confers the SELECT privilege on any individual table.

Whenever a user needs a privilege for an operation, you can verify if the user already has the privilege if either the user is the owner of the object or has a higher level privilege that confers the necessary privileges for that operation. For example, if the user Pat needs to have the SELECT privilege for Terry.Table2, you can check the following:

- Is Pat the owner of the object? If so, owners have all object privileges on their objects
- Has Pat been granted the SELECT ANY TABLE privilege? This privilege means Pat would have SELECT ON any table, view, materialized view.
- Has Pat been granted the ADMIN privilege, which would mean that Pat can perform any valid SQL operation.

If you grant a privilege that is included in a higher level privilege, no error occurs. However, when you revoke privileges, they must be revoked in the same unit as granted. The following sequence of grant and revoke statements for user PAT grants the ability to update any table as well as an update privilege on a specific table:

```
GRANT UPDATE ANY TABLE TO PAT;
GRANT UPDATE ON HR.employees TO PAT;
REVOKE UPDATE ON HR.employees FROM PAT;
```

The UPDATE ANY TABLE privilege grants the ability to update any table in the database. The second grant is specific for UPDATE privilege to the `HR.employees` table. The second grant is unnecessary as the UPDATE ANY TABLE provides access to all tables, including employees, but it does not result in an error. You can revoke the second grant, but it will not affect the first grant of the UPDATE ANY TABLE system privilege. Thus, PAT can still update the `HR.employees` table.

You must revoke in the same unit as was granted. The following example grants the UPDATE ANY TABLE system privilege to PAT. A user tries to revoke the ability to update the `HR.employees` table from the user. But, the UPDATE ANY TABLE privilege is a system privilege and the UPDATE privilege is an object privilege. The execution of the REVOKE statement for a unit that was not granted fails with an error.

```
GRANT UPDATE ANY TABLE TO PAT;
REVOKE UPDATE ON HR.employees FROM PAT;
15143: REVOKE failed: User PAT does not have object privilege UPDATE on
HR.EMPLOYEES
The command failed.
```

The full details of the privilege hierarchy is described in the "Privileges" chapter in the *Oracle TimesTen In-Memory Database SQL Reference*.

Granting or revoking system privileges

To grant or revoke a system privilege, use the GRANT or REVOKE statements. Only the instance administrator or a user with the ADMIN privilege can grant or revoke system privileges. The GRANT or REVOKE syntax for system privileges includes the system privilege and the user who receives that privilege. Both the syntax for the GRANT and REVOKE statements and the required privileges for executing each SQL statement are described in the "SQL Statements" chapter in the *Oracle TimesTen In-Memory Database SQL Reference*.

Note: How to grant and revoke object privileges is described in "Granting or revoking object privileges" on page 4-12.

The most powerful system privilege is ADMIN. When you grant a user the ADMIN privilege, you enable this user to perform any operation for any database object.

An individual user can view their own system privileges in the SYS.USER_SYS_PRIVS system view. A user with the ADMIN privilege can view all system privileges for all users in the SYS.DBA_SYS_PRIVS system table. These system views are described in "Viewing user privileges" on page 4-19.

The following sections describe some of the system privileges available in TimesTen:

- Granting administrator privileges
- Granting ALL PRIVILEGES
- Granting privilege to connect to the database
- Granting additional system privileges
- Enabling users to perform operations on any database object type
- Granting or revoking privileges for cache groups

Note: For a full list of all system privileges, see "Privileges" in the *Oracle TimesTen In-Memory Database SQL Reference*.

Granting administrator privileges

The ADMIN privilege confers all system and object privileges, which allows these users to perform all administrative tasks and valid database operations. For all objects, a user with the ADMIN privilege can perform create, alter, drop, select, update, insert, or delete operations. In addition, a user with the ADMIN privilege can perform replication tasks, checkpointing, backups, migration, user creation and deletion, and so on. Only a user with the ADMIN privilege can grant or revoke all privileges.

Only a user with the ADMIN privilege may view all system tables and views by default. Only a user with the ADMIN privilege can create, alter or drop replication schemas or active standby pairs. The following views and packages can only be accessed by users with the ADMIN privilege:

- The SYS.DBA_TAB_PRIVS view
- The SYS.DBA_SYS_PRIVS view
- The SYS.UTL_RECOMP package

Note: For more information on viewing privileges for users from system tables or views, see "Viewing user privileges" on page 4-19.

To grant the ADMIN privilege to the user `TERRY`, execute the following statement:

```
GRANT ADMIN TO TERRY;
```

If you have the ADMIN privilege, then you can grant privileges to other users. For example, a user with the ADMIN privilege can grant the SELECT privilege to `TERRY` on the `departments` table owned by `PAT`, as follows:

```
GRANT SELECT ON PAT.departments TO TERRY;
```

Note: Since Pat is the owner of departments, Pat may also grant the SELECT object privilege to Terry.

Granting ALL PRIVILEGES

The ALL PRIVILEGES grants every system privilege to a user. If you want a user to have most of the system privileges, you can grant ALL PRIVILEGES to a user and then revoke only those system privileges that you do not want them to have. The following example grants all system privileges to user PAT. Then, revokes the ADMIN and DROP ANY TABLE privileges to disallow Pat the ability to perform all administration tasks or to drop any tables.

```
GRANT ALL PRIVILEGES TO PAT;
REVOKE ADMIN, DROP ANY TABLE FROM PAT;
```

You may also REVOKE ALL PRIVILEGES that were granted to a user. This removes all system privileges from the user, except what the user inherits from the PUBLIC role, as demonstrated below for user PAT:

```
REVOKE ALL PRIVILEGES FROM PAT;
```

Granting privilege to connect to the database

TimesTen databases are accessed through Data Source Names (DSNs). If a user tries to use a DSN that has connection attributes for which they do not have privileges, such as first connection attributes, they receive an error.

For a complete description of first connection attributes, see "Data Store Attributes" in the *Oracle TimesTen In-Memory Database Reference.*

All users must be granted the CREATE SESSION system privilege by a user with the ADMIN privilege in order to connect to the database. The CREATE SESSION system privilege provides the authorization to connect to the database. The following example grants the CREATE SESSION privilege to PAT:

```
GRANT CREATE SESSION TO PAT;
```

A user with the ADMIN privilege can grant CREATE SESSION privilege to all users by granting this privilege to the PUBLIC role. This allows all users to connect to the database.

```
GRANT CREATE SESSION TO PUBLIC;
```

Granting additional system privileges

In addition to the ADMIN privilege, there are a few system privileges that confer a superset of abilities. The following provides a brief description of these privileges:

- XLA: XLA readers can have global impact on the system. They create extra log volume, and can cause long log holds if they do not advance their bookmarks. You must have the XLA system privilege to connect as an XLA reader.
- CACHE_MANAGER: The CACHE_MANAGER privilege is used for cache group administrator operations. See "Granting or revoking privileges for cache groups" on page 4-17 for details.

Enabling users to perform operations on any database object type

When you want to grant or revoke privileges for a user, you can grant or revoke privileges for a single object or for that type of object anywhere in the database.

> **Note:** To grant or revoke privileges for a single object, use object privileges, which are described in "Granting or revoking object privileges" on page 4-12.

The system privileges that contain the ANY keyword enable the user to perform the functions on all objects of the same type in the database. These system privileges are CREATE ANY *object_type*, DROP ANY *object_type*, ALTER ANY *object_type*, SELECT ANY *object_type*, UPDATE ANY TABLE, INSERT ANY TABLE, DELETE ANY TABLE, and EXECUTE ANY PROCEDURE.

> **Note:** For a full description of these privileges, see "Privileges" in the *Oracle TimesTen In-Memory Database SQL Reference*. For details on the cache group system privileges that contain the ANY keyword, see "Granting or revoking privileges for cache groups" on page 4-17.

The following sections provide more details for the CREATE ANY *object_type*, DROP ANY *object_type*, and ALTER ANY *object_type* system privileges:

- Creating a table, index, view, materialized view, sequence, procedure, function or package
- Dropping a table, view, materialized view, sequence, procedure, function or package

- Altering a table, view, materialized view, sequence, procedure, function or package

Creating a table, index, view, materialized view, sequence, procedure, function or package To create a table, view, materialized view, sequence, procedure, function or package within the user's namespace or another user's namespace, you must have the appropriate CREATE *object_type* or CREATE ANY *object_type* system privileges.

The following describes the CREATE and CREATE ANY system privileges:

- The CREATE *object_type* privilege grants a user the ability to create that object, but only in the user's own schema. After creation, the user owns this object and thus, automatically has been granted all privileges for that object.

  Other privileges are required if a user wants to create cache groups.

- The CREATE ANY *object_type* privilege grants a user the ability to create any object of that type in the database, even in another user's schema. The object types include table, index, view, materialized view, sequence and procedure. The CREATE ANY *object_type* privileges are CREATE ANY TABLE, CREATE ANY INDEX, CREATE ANY VIEW, CREATE ANY MATERIALIZED VIEW, CREATE ANY SEQUENCE, and CREATE ANY PROCEDURE.

> **Note:** Be cautious in granting the CREATE ANY PROCEDURE or EXECUTE ANY PROCEDURE privileges. They can be misused, especially in combination with each other.

The following example grants the privilege to create any table in other users' schemas to user TERRY:

```
GRANT CREATE ANY TABLE TO TERRY;
```

The following example grants the privilege to create a table within the user's own schema:

```
GRANT CREATE TABLE TO TERRY;
```

Dropping a table, view, materialized view, sequence, procedure, function or package Grant the DROP ANY *object_type* system privilege in order for a user to drop an object of *object_type* that the user does not own. For example, granting PAT this privilege enables Pat to drop the `employees` table that is owned by the user HR. A user always has the right to drop a table they own. The DROP ANY *object_type* privilege enables a user to drop any object of the specified type in the database, except for cache groups that require other privileges.

Altering a table, view, materialized view, sequence, procedure, function or package ALTER ANY PROCEDURE allows users to alter any procedure, function or package in the database. The ALTER ANY *object_type* privilege is necessary to modify the properties of objects that the user does not own. For example, granting the ALTER ANY PROCEDURES privilege to Pat enables PAT to alter the `employees` object owned by HR. A user always has the right to alter a table they own.

Granting or revoking object privileges

To grant or revoke an object privilege, use the GRANT or REVOKE statements. The syntax for the object-level GRANT or REVOKE statement requires the name of the object on which the grant or revoke is applied. The syntax for the GRANT and

REVOKE statements is described in the "SQL Statements" chapter in the *Oracle TimesTen In-Memory Database SQL Reference*.

The following sections describe and provide examples on the object privileges for all object types, except for the cache admin objects. The cache object privileges are described in "Granting or revoking privileges for cache groups" on page 4-17.:

Note:

- Each SQL statement may require a certain privilege. The required privileges are documented with each statement description in the "SQL Statements" chapter in the *Oracle TimesTen In-Memory Database SQL Reference*.
- For a full list of all object privileges, see "Privileges" in the *Oracle TimesTen In-Memory Database SQL Reference*.

- Grant all object privileges
- Object privileges for tables
- Object privileges for views
- Object privileges for sequences
- Object privileges for materialized views
- Object Privileges needed when creating foreign key with REFERENCES clause
- Object privileges for functions, procedures and packages

Grant all object privileges

You can grant all privileges for an object to a user with the ALL keyword. This essentially grants a user the right to perform any operation on the object.

There are no specific object privileges for DROP or ALTER. These operations cannot be granted for individual objects; instead, granting the appropriate system privilege enables a user other the owner of an object to DROP or ALTER that object.

For example, `GRANT ALL ON employees TO PAT` grants all privileges for the employees table to user PAT. It is possible to revoke individual privileges after granting all object privileges. For instance, the following is a valid sequence of operations:

```
GRANT ALL ON HR.employees TO PAT;
REVOKE DELETE ON HR.employees FROM PAT;
```

You may also REVOKE ALL object privileges that were granted to a user for the object. This removes all privileges for the object from the user, as demonstrated below for user PAT:

```
REVOKE ALL ON HR.employees FROM PAT;
```

Both the object owner and a user with the ADMIN privilege can perform the GRANT ALL and REVOKE ALL statements.

Object privileges for tables

For a user to perform operations on tables that they do not own, they must be granted the appropriate object privilege for that table. This includes privileges for tables within cache groups. The object privileges for tables include SELECT, UPDATE, DELETE, INSERT, INDEX and REFERENCES.

The following object privileges may be appropriate not only for authorization, but also for performance reasons:

- The INDEX privilege enables the user to create an index on the table. Creating an index consumes additional space and impacts the performance of DML on the table. A specific grant for INDEX is required for a user to create an index.
- The REFERENCES privilege enables the user to create a foreign key dependency on the table. Foreign key dependencies impact the performance of DML operations on the parent. For more details on the REFERENCES privilege, see "Object Privileges needed when creating foreign key with REFERENCES clause" on page 4-15.

The following example grants the SELECT object privilege for the employees table in the HR schema to the user PAT:

```
GRANT SELECT ON HR.employees TO PAT;
```

The next example shows an example of how to grant the UPDATE privilege on the `employees` table owned by the user `HR` to the user Pat:

```
GRANT UPDATE ON HR.employees TO PAT;
```

Object privileges for views

For a user to create a view, that user must be granted the CREATE VIEW or CREATE ANY VIEW privilege. For a user to select from a view that they do not own, they need to be granted the SELECT object privilege for that view. Furthermore, the view itself needs to be valid; that is, the owner of the view must be granted the SELECT object privilege for all of the objects referenced by the view.

When user Pat creates a view owned by Pat and that view only references objects owned by Pat, then Pat is only required to be granted the CREATE VIEW privilege for this operation. If user Pat creates a view owned by Terry that references objects owned by Terry, then Pat is required to be granted the CREATE ANY VIEW privilege for this operation. For example:

```
CREATE VIEW PAT.VIEW1 as select * from PAT.TABLE1;
```

In this example, if Pat executes this statement, then Pat only needs to be granted the CREATE VIEW privilege.

If user Pat creates a view, and the view references a table owned by Terry, then Pat needs to be granted CREATE VIEW privilege and the SELECT object privilege on all of the objects referenced by the view. The owner of the view, not the view creator, must be granted the SELECT object privilege on the objects referenced by the view. Therefore, in this example, Pat must be granted the SELECT object privilege on TABLE2 that is owned by Terry. Once these privileges are granted, Pat can execute the following:

```
CREATE VIEW PAT.VIEW2 as select * from TERRY.TABLE2;
```

However, if a third user, Joe, executes this statement, then Joe must be granted the CREATE ANY VIEW privilege to create the view. Even though Joe is executing the statement, Pat, as the owner of the view, is still required to be granted the SELECT object privilege in order to perform the select on Terry's table.

TimesTen validates all views referenced at execution time. TimesTen will notify which privileges are not in place in order to perform the given operation.

For example:

```
CREATE VIEW PAT.VIEW2 as select * from TERRY.TABLE2;
CREATE VIEW JOE.VIEW4 as select * from PAT.VIEW2, TERRY.TABLE4;
```

If Pat is executing these statements, the following privileges must be granted:

- CREATE ANY VIEW privilege so that Pat can create the view in Pat’s own schema as well as a view in Joe's schema.
- USER Joe must be granted the SELECT object privilege on Terry.Table4.
- USER Joe must be granted the SELECT object privilege on Pat.View2
- USER Pat must be granted the SELECT object privilege on Terry.Table2

When validating all references, TimesTen also validates that PAT.VIEW2 is still valid by verifying that Pat has the SELECT object privilege on TERRY.TABLE2. When you select from a view, TimesTen validates that the view itself is still valid, as well as any views referenced by that view.

Object privileges for sequences

For a user to perform operations on sequences that they do not own, they must be granted the SELECT object privilege. The SELECT privilege on a sequence allows the user to perform all operations on a sequence, including NEXTVAL, even though it ultimately updates the sequence.

For example, to grant SELECT privilege on the `employees_seq` sequence in the HR schema to the user PAT, issue the following statement:

```
GRANT SELECT ON HR.employees_seq TO PAT;
```

The user PAT can subsequently generate the next value of the sequence with the following statement:

```
SELECT HR.employees_seq.NEXTVAL FROM DUAL;
< 207 >
1 row found.
```

Object privileges for materialized views

For a user to perform operations on materialized views that they do not own, they must be granted the appropriate table object privilege. The object privileges for materialized views include SELECT, INDEX and REFERENCES.

Object Privileges needed when creating foreign key with REFERENCES clause

The REFERENCES clause in the CREATE or ALTER TABLE statements creates a foreign key dependency from the new child table column (TABLE1.COL1) on the parent table column (TABLE2.PK) as shown in the following operation:

```
ALTER TABLE PAT.TABLE1 ADD CONSTRAINT FK1
    FOREIGN KEY (COL1) REFERENCES PAT.TABLE2 (PK);
```

In this example, the user executing the SQL must have ALTER ANY TABLE privilege. Since Pat owns both tables, no additional privileges are needed since Pat owns both tables.

However, if the REFERENCES clause refers to a table not owned by this user, then the REFERENCES object privilege on the table not owned by the user is required before execution is allowed. For example:

```
ALTER TABLE PAT.TABLE1 ADD CONSTRAINT FK1
```

```
    FOREIGN KEY (COL1) REFERENCES TERRY.TABLE2 (PK);
```

In this example, the user executing this SQL must have ALTER ANY TABLE privilege. As in the previous example, if the user executing this SQL is Pat, the ALTER ANY TABLE privilege is not required because a table's owner can always modify their own table. In addition, the user Pat must be granted the REFERENCES privilege on TERRY.TABLE2 in order for Pat to create a foreign key involving a table owned by Terry.

A user who creates or alters a child table needs the REFERENCES object privilege on the parent table to create a foreign key dependency. The REFERENCES privilege implicitly grants SELECT privileges for a user creating a foreign key on the parent table. However, this implicit grant does not mean that the user has the SELECT privilege on the parent table, so any SELECT statements will fail if the only privilege on the parent table is the REFERENCES privilege.

Object privileges for functions, procedures and packages

For a user to perform operations on PL/SQL functions, procedures or packages that they do not own, they must be granted the EXECUTE object privilege. When you grant a user EXECUTE privilege on a package, this automatically grants EXECUTE privilege on its component procedures and functions.

This privilege grants the right to the following:

- Execute the procedure or function.
- Access any program object declared in the specification of a package.
- Compile the object implicitly during a call to a currently invalid or uncompiled function or procedure.

The EXECUTE privilege does not allow the user to create, drop or alter any procedure, function or package. This requires appropriate system privileges. For example, to explicitly compile using ALTER PROCEDURE or ALTER FUNCTION, the user must be granted the ALTER ANY PROCEDURE system privilege. For details on the system privileges for functions, procedures or packages, see "Enabling users to perform operations on any database object type" on page 4-11.

Granting or revoking multiple privileges with a single SQL statement

You can grant multiple object privileges in the same GRANT or REVOKE statement for the same database object for one or more users. For example, the following grants Terry the SELECT and UPDATE object privileges on the HR.employees table in the same SQL statement.

```
GRANT SELECT, UPDATE ON HR.employees TO TERRY;
```

You can also grant multiple system privileges to one or more users with the same GRANT or REVOKE statement. The following example grants multiple system privileges to both Terry and Pat.

```
GRANT CREATE ANY TABLE, CREATE SESSION TO TERRY, PAT;
```

You cannot combine system and object privileges in the same GRANT or REVOKE statement.

Granting or revoking privileges for cache groups

In order for a user to be able to perform activities involving any cache group, the user must have the appropriate cache group privileges. There are system and object privileges for cache groups, where system privileges confer abilities beyond a singular object.

Cache group privileges are required for performing cache group operations. However, tables within a cache group require the granting of the object table privileges.

Note: Passthrough does not require any privileges to be granted, since the privilege checking will be performed by the Oracle Database with the user credentials. For details on passthrough, see the *Oracle In-Memory Database Cache User's Guide*.

The following sections provide an overview of cache group privileges:

- Cache manager privilege
- Cache group system privileges
- Cache group object privileges

For a full list of all system and object privileges for cache groups, as well as table privileges, see "Privileges" in the *Oracle TimesTen In-Memory Database SQL Reference*.

Cache manager privilege

The cache group system privileges provide a user the ability to affect cache group objects across the database. The CACHE_MANAGER system privilege is the administrator privilege for cache groups. If a user has been granted the CACHE_MANAGER privilege, then this user may perform any operation for cache groups.

You must have the CACHE_MANAGER privilege to perform the initial load of a read-only cache group or to change the state of autorefresh on a read-only cache group. The initial load implicitly alters the state of the cache group autorefresh from paused to on.

The following grants the CACHE_MANAGER privilege to PAT:

```
GRANT CACHE_MANAGER TO PAT;
```

Note: An asynchronous writethrough (AWT) cache group combines both cache groups and replication. The CACHE_MANAGER privilege provides all of the privileges that you need for creating AWT cache groups.

Cache group system privileges

In order to create a cache group, a user must have the CREATE CACHE GROUP or CREATE ANY CACHE GROUP system privilege. To drop or alter a cache group that is not owned by the user, the user must be granted the DROP ANY CACHE GROUP or ALTER ANY CACHE GROUP as appropriate. For example, the following confers the privilege for a user to alter any cache group in the database:

```
GRANT ALTER ANY CACHE GROUP TO PAT;
```

When the user creates or drops a cache group that includes a table, the user must also be granted either the CREATE ANY TABLE or CREATE TABLE privileges, depending on if the table is owned by the user or not.

For example, user Pat creates a cache group CACHEGRP based on a table T1 owned by TERRY as follows:

```
CREATE CACHE GROUP PAT.CACHEGRP AS SELECT * FROM TERRY.T1;
```

In order for Pat to execute this statement, Pat must be granted both CREATE CACHE GROUP and CREATE ANY TABLE. The second privilege is required in order for Pat to create the TERRY.T1 table, which is in another user's schema. If Pat wants to drop this cache group, Pat would need to be granted DROP ANY TABLE as Pat does not own the table.

User Pat creates a cache group CACHEGRP based on table T1 owned by Pat as follows:

```
CREATE CACHE GROUP PAT.CACHEGRP AS SELECT * FROM PAT.T1;
```

In order for Pat to execute this statement, Pat must be granted both CREATE CACHE GROUP and CREATE TABLE. The second privilege is required in order for Pat to create the PAT.T1 table, which is in Pat's schema. Since Pat owns both the cache group and the table, Pat does not need any additional privileges to drop the cache group.

User Pat creates a cache group TERRY.CACHEGRP based on table T1 owned by Terry as follows:

```
CREATE CACHE GROUP TERRY.CACHEGRP AS SELECT * FROM TERRY.T1;
```

In order for Pat to execute this statement, Pat must be granted both CREATE ANY CACHE GROUP and CREATE ANY TABLE. These privileges are required in order for Pat to create the cache group and table as owned by TERRY. If Pat wants to drop this cache group, Pat would also need to be granted DROP ANY CACHE GROUP and DROP ANY TABLE since Pat does not own either object.

Other system privileges for cache group operations are for performing the following on objects not owned by the user:

- FLUSH ANY CACHE GROUP: Allows users to flush any cache group in the database.
- LOAD ANY CACHE GROUP: Allows users to load any cache group in the database.
- UNLOAD ANY CACHE GROUP: Allows users to unload any cache group in the database.
- REFRESH ANY CACHE GROUP: Allows users to refresh any cache group in the database.

Cache group object privileges

The CACHE_MANAGER privilege is sufficient for performing DML operations on a cache group. The ANY system privileges described in the previous section enable the user to perform that operation on any object of that type. Alternatively, you can grant finer-grained privileges that only allow particular DML operations on a singular object.

The object privileges for cache group operations are granted to a user for performing the operation on a single, defined object. The following are the object privileges for cache group objects:

- FLUSH: Allows the user to flush a cache group owned by another user.
- LOAD: Allows the user to load a cache group owned by another user.
- UNLOAD: Allows the user to unload a cache group owned by another user.
- REFRESH: Allows the user to refresh a cache group owned by another user.

For example, the following example grants PAT the cache group object privilege to perform a FLUSH on the cache group CACHEGRP that is owned by Terry:

```
GRANT FLUSH ON TERRY.CACHEGRP TO PAT;
```

For details on cache group operations, see the *Oracle In-Memory Database Cache User's Guide*.

Viewing user privileges

You can view the privileges granted to each user through the following views:

Table 4–1 System privilege views

View name	Description
SYS.USER_SYS_PRIVS	Returns all of the system privileges granted to the current user.
SYS.DBA_SYS_PRIVS	Returns the list of system privileges granted to all users and inherited from the PUBLIC role. Requires the ADMIN privilege to select from this view.
SYS.USER_TAB_PRIVS	Returns all of the object privileges granted to the current user.
SYS.ALL_TAB_PRIVS	Returns the results of both USER_TAB_PRIVS and the object privileges inherited from the PUBLIC role for a user. This shows all object privileges granted to a user.
SYS.DBA_TAB_PRIVS	Returns the object privileges granted to all users and inherited from the PUBLIC role. Requires the ADMIN privilege to select from this view.

For example, perform the following to see all of the system privileges granted to all users:

```
Command> SELECT * FROM SYS.DBA_SYS_PRIVS;
< SYS, ADMIN, YES >
< SYSTEM, ADMIN, YES >
< TERRY, ADMIN, YES >
< TERRY, CREATE ANY TABLE, NO >
< PAT, CACHE_MANAGER, NO >
5 rows found.
```

Note: For details on these views, see "System and Replication Tables" in the *Oracle TimesTen In-Memory Database SQL Reference*.

Privileges needed for utilities, built-in procedures and first connection attributes

Many of the utilities and built-in procedures require a certain privilege in order to execute. In addition, in order to modify or connect with certain first connection attributes, certain privileges are required. First connection attributes are set when a database is first loaded, and only the instance administrator can load a database with first connection attribute settings. The required privilege for each is described with the utility, built-in procedure or first connection attribute description in the *Oracle TimesTen In-Memory Database Reference*.

Privilege checking rules for parent-child tables

If you have tables related by foreign key constraints, then the following applies:

- If ON DELETE CASCADE is specified on a foreign-key constraint for a child table, a user can delete rows from the parent table resulting in deletions from the child table without requiring an explicit DELETE privilege on the child table. However, a user must have the DELETE privilege on the parent table for this to occur automatically.
- When you perform an insert or update on a child table, TimesTen determines if there is a foreign key constraint violation on the parent table resulting from the change to the child table. In this case, a user is required to have the INSERT or UPDATE privilege on the child table, but not a SELECT privilege on the parent table.
- A user who creates a child table needs the REFERENCES object privilege on the parent table to create a foreign key dependency. See "Object Privileges needed when creating foreign key with REFERENCES clause" on page 4-15 for more details.

5 Globalization Support

This chapter describes TimesTen globalization support features. It includes the following topics:

Overview of globalization support features

TimesTen globalization support includes the following features:

- Character set support

  You must choose a database character set when you create a data store. See "Supported character sets" on page 5-10 for a list of supported character sets. You can also choose a connection character set for a session. See "Connection character set" on page 5-4.

- Length semantics

  You can specify byte semantics or character semantics for defining the storage measurement of character data types. See "Length semantics and data storage" on page 5-3.

- Linguistic sorts and indexes. You can sort data based on linguistic rules. See "Linguistic sorts" on page 5-4. You can use linguistic indexes to improve performance of linguistic sorts. See "Using linguistic indexes" on page 5-6.

- SQL string and character functions

  TimesTen provides SQL functions that return information about character strings. TimesTen also provides SQL functions that return a character from an encoded value. See "SQL string and character functions" on page 5-6.

Note: This release of TimesTen does not support session language and territory.

Choosing a database character set

TimesTen uses the database character set to define the encoding of data stored in character data types, such as CHAR and VARCHAR2.

Use the DatabaseCharacterSet data store attribute to specify the database character set during data store creation. You cannot alter the database character set after data store creation, and there is no default value for DatabaseCharacterSet. See "Supported character sets" on page 5-10 for a list of supported character sets.

Consider the following questions when you choose a character set for a data store:

- What languages does the data store need to support now and in the future?
- Is the character set available on the operating system?
- What character sets are used on clients?
- How well does the application handle the character set?
- What are the performance implications of the character set?

If you are using Oracle In-Memory Database Cache (IMDB Cache) to cache Oracle tables, you must create the data store with the same database character set as the Oracle Database.

This section includes the following topics:

- Character sets and languages
- Client operating system and application compatibility
- Performance and storage implications
- Character sets and replication

Character sets and languages

Choosing a database character set determines what languages can be represented in the database.

A group of characters, such as alphabetic characters, ideographs, symbols, punctuation marks, and control characters, can be encoded as a character set. An encoded character set assigns unique numeric codes to each character in the character repertoire. The numeric codes are called code points or encoded values.

Character sets can be single-byte or multibyte. Single-byte 7-bit encoding schemes can define up to 128 characters and normally support just one language. Single-byte 8-bit encoding schemes can define up to 256 characters and often support a group of related languages. Multibyte encoding schemes are needed to support ideographic scripts used in Asian languages like Chinese or Japanese because these languages use thousands of characters. These encoding schemes use either a fixed number or a variable number of bytes to represent each character. Unicode is a universal encoded character set that enables information from any language to be stored using a single character set. Unicode provides a unique code value for every character, regardless of the platform, program, or language.

Client operating system and application compatibility

The database character set is independent of the operating system. On an English operating system, you can create and run a database with a Japanese character set. However, when an application in the client operating system accesses the database, the client operating system must be able to support the database character set with appropriate fonts and input methods. For example, you cannot insert or retrieve Japanese data on the English Windows operating system without first installing a Japanese font and input method. Another way to insert and retrieve Japanese data is to use a Japanese operating system remotely to access the database server.

If all client applications use the same character set, then that character set is usually the best choice for the database character set. When client applications use different character sets, the database character set should be a superset of all the application character sets. This ensures that every character is represented when converting from an application character set to the database character set.

Performance and storage implications

For best performance, choose a character set that avoids character set conversion and uses the most efficient encoding for the languages desired. Single-byte character sets result in better performance than multibyte character sets, and they also are the most efficient in terms of space requirements. However, single-byte character sets limit how many languages you can support.

Character sets and replication

All data stores in a replication scheme must have the same database character set. No character set conversion occurs during replication.

Length semantics and data storage

In single-byte character sets, the number of bytes and the number of characters in a string are the same. In multibyte character sets, a character or code point consists of one or more bytes. Calculating the number of characters based on byte lengths can be difficult in a variable-width character set. Calculating column lengths in bytes is called **byte semantics**, while measuring column lengths in characters is called **character semantics**.

Character semantics is useful for defining the storage requirements for multibyte strings of varying widths. For example, in a Unicode database (AL32UTF8), suppose that you need to define a VARCHAR2 column that can store up to five Chinese characters together with five English characters. Using byte semantics, this column requires 15 bytes for the Chinese characters, which are three bytes long, and 5 bytes for the English characters, which are one byte long, for a total of 20 bytes. Using character semantics, the column requires 10 characters.

The expressions in the following list use byte semantics. Note the BYTE qualifier in the CHAR and VARCHAR2 expressions.

- CHAR (5 BYTE)
- VARCHAR2(20 BYTE)

The expressions in the following list use character semantics. Note the CHAR qualifier in the VARCHAR2 expression.

- VARCHAR2(20 CHAR)

- SUBSTR(*string*, 1, 20)

By default, the CHAR and VARCHAR2 character data types are specified in bytes, not characters. Therefore, the specification CHAR(20) in a table definition allows 20 bytes for storing character data.

The NLS_LENGTH_SEMANTICS general connection attribute determines whether a new column of character data type uses byte or character semantics. It enables you to create CHAR and VARCHAR2 columns using either byte-length or character-length semantics without having to add the explicit qualifier. NCHAR and NVARCHAR2 columns are always character-based. Existing columns are not affected.

The default value for NLS_LENGTH_SEMANTICS is BYTE. Specifying the BYTE or CHAR qualifier in a data type expression overrides the NLS_LENGTH_SEMANTICS value.

Connection character set

The database character set determines the encoding of CHAR and VARCHAR2 character data types. The connection character set is used to describe the encoding of the incoming and outgoing application data, so that TimesTen can perform the necessary character set conversion between the application and the database. For example, this allows a non-Unicode application to communicate with a Unicode (AL32UTF8) database.

The ConnectionCharacterSet general connection attribute sets the character encoding for the connection, which can be different than the database character set. The connection uses the connection character set for information that passes through the connection, such as parameters, SQL query text, results and error messages. Choose a connection character set that matches the application environment or the character set of your data source.

Best performance results when the connection character set and the database character set are the same because no conversion occurs. When the connection character set and the database character set are different, data conversion is performed in the ODBC layer. Characters that cannot be converted to the target character set are changed to replacement characters.

The default connection character set is US7ASCII. This setting applies to both direct and client connections.

Linguistic sorts

Different languages have different sorting rules. Text is conventionally sorted inside a database according to the binary codes used to encode the characters. Typically, this does not produce a sort order that is linguistically meaningful. A linguistic sort handles the complex sorting requirements of different languages and cultures. It enables text in character data types, such as CHAR, VARCHAR2, NCHAR, and NVARCHAR2, to be sorted according to specific linguistic conventions.

A linguistic sort operates by replacing characters with numeric values that reflect each character's proper linguistic order. TimesTen offers two kinds of linguistic sorts: monolingual and multilingual.

This section includes the following topics:

- Monolingual linguistic sorts
- Multilingual linguistic sorts

- Case-insensitive and accent-insensitive linguistic sorts
- Performing a linguistic sort
- Using linguistic indexes

Monolingual linguistic sorts

TimesTen compares character strings in two steps for monolingual sorts. The first step compares the major value of the entire string from a table of major values. Usually, letters with the same appearance have the same major value. The second step compares the minor value from a table of minor values. The major and minor values are defined by TimesTen. TimesTen defines letters with accent and case differences as having the same major value but different minor values.

Monolingual linguistic sorting is available only for single-byte and Unicode database character sets. If a monolingual linguistic sort is specified when the database character set is non-Unicode multibyte, then the default sort order is the binary sort order of the database character set.

For a list of supported sorts, see "Supported linguistic sorts" on page 5-12.

Multilingual linguistic sorts

TimesTen provides multilingual linguistic sorts so that you can sort data for multiple languages in one sort. Multilingual linguistic sort is based on the ISO/OEC 14651 - International String Ordering and the Unicode Collation algorithm standards. This framework enables the database to handle languages that have complex sorting rules, such as those in Asian languages, as well as providing linguistic support for databases with multilingual data.

In addition, multilingual sorts can handle canonical equivalence and supplementary characters. Canonical equivalence is a basic equivalence between characters or sequences of characters. For example, ç is equivalent to the combination of c and ,.

For example, TimesTen supports a monolingual French sort (FRENCH), but you can specify a multilingual French sort (FRENCH_M). _M represents the ISO 14651 standard for multilingual sorting. The sorting order is based on the GENERIC_M sorting order and can sort accents from right to left. TimesTen recommends using a multilingual linguistic sort if the tables contain multilingual data. If the tables contain only French, then a monolingual French sort may have better performance because it uses less memory. It uses less memory because fewer characters are defined in a monolingual French sort than in a multilingual French sort. There is a trade-off between the scope and the performance of a sort.

For a list of supported multilingual sorts, see "Supported linguistic sorts" on page 5-12.

Case-insensitive and accent-insensitive linguistic sorts

Operations inside a database are sensitive to the case and the accents of the characters. Sometimes you might need to perform case-insensitive or accent-insensitive comparisons.

To specify a case-insensitive or accent-insensitive sort:

- Append _CI to a TimesTen sort name for a case-insensitive sort. For example:

  BINARY_CI: accent-sensitive and case-insensitive binary sort

  GENERIC_M_CI: accent-sensitive and case-insensitive GENERIC_M sort

- Append _AI to a TimesTen sort name for an accent-insensitive and case-insensitive sort. For example:

  BINARY_AI: accent-insensitive and case-insensitive binary sort

  FRENCH_M_AI: accent-insensitive and case-insensitive FRENCH_M sort

Performing a linguistic sort

The NLS_SORT data store connection attribute indicates which collating sequence to use for linguistic comparisons. The NLS_SORT value affects the SQL string comparison operators and the ORDER BY clause.

You can use the ALTER SESSION statement to change the value of NLS_SORT:

```
ALTER SESSION SET NLS_SORT=SWEDISH;
SELECT product_name
  FROM product
  ORDER BY product_name;

PRODUCT NAME
------------
aerial
Antenne
Lcd
ächzen
Ähre
```

You can also override the NLS_SORT setting by using the NLSSORT SQL function to perform a linguistic sort:

```
SELECT * FROM test ORDER BY NLSSORT(name,'NLS_SORT=SPANISH');
```

For more extensive examples of using NLSSORT, see "NLSSORT" in the *Oracle TimesTen In-Memory Database SQL Reference*.

Using linguistic indexes

You can create a linguistic index to achieve better performance during linguistic comparisons. A linguistic index requires storage for the sort key values.

To create a linguistic index, use a statement similar to the following:

```
CREATE INDEX german_index ON employees
(NLSSORT(employee_id, 'NLS_SORT=GERMAN'));
```

The optimizer chooses the appropriate index based on the values for NLSSORT and NLS_SORT.

You must create multiple linguistic indexes if you want more than one linguistic sort on a column. For example, if you want both GERMAN and GERMAN_CI sorts against the same column, create two linguistic indexes.

For more information, see "CREATE INDEX" in the *Oracle TimesTen In-Memory Database SQL Reference*.

SQL string and character functions

The following table summarizes SQL functions that operate on character strings:

SQL function	Description
ASCIISTR	Takes as its argument either a string or an expression that resolves to a string in any character set. It returns the ASCII version of the string in the database character set. Non-ASCII characters are converted to Unicode escapes.
INSTR INSTRB INSTR4	Determines the first position, if any, at which one string occurs within another string. INSTRB uses bytes instead of characters. INSTR4 uses UCS4 code points.
LENGTH LENGTHB LENGTH4	Returns the length of a character string in an expression as number of characters. LENGTHB uses bytes instead of characters. LENGTH4 uses UCS4 code points.
LOWER and UPPER	The LOWER function converts expressions of type CHAR, NCHAR, VARCHAR2 or NVARCHAR2 to lowercase. The UPPER function converts expressions of type CHAR, NCHAR, VARCHAR2 or NVARCHAR2 to uppercase. Character semantics is supported for CHAR and VARCHAR2 types. The data type of the result is the same as the data type of the expression.
RTRIM	Removes trailing spaces from CHAR, VARCHAR2, NCHAR or NVARCHAR2 strings.
SUBSTR SUBSTRB SUBSTR4	Returns a VARCHAR2 or NVARCHAR2 string that represents a substring of a CHAR or NCHAR string. The returned substring is a specified number of characters, beginning from a designated starting point. SUBSTRB uses bytes instead of characters. SUBSTR4 uses UCS4 code points.
UNISTR	Takes as its argument a string that resolves to data of type NVARCHAR2. It returns the value in UTF-16 format. Unicode escapes are supported.

The following functions return characters:

- CHR: Returns the character with the specified binary value in the database character set.
- NCHR: Returns the character with the specified Unicode value.

See "Expressions" in the *Oracle TimesTen In-Memory Database SQL Reference* for more information including examples.

Setting globalization support attributes

The globalization support attributes are summarized in the following table:

Parameter	Description
DatabaseCharacterSet	Indicates the character encoding used by a data store.
ConnectionCharacterSet	Determines the character encoding for the connection, which may be different from the database character set.
NLS_SORT	Indicates the collating sequence to use for linguistic comparisons.
NLS_LENGTH_SEMANTICS	Sets the default length semantics.

Parameter	Description
NLS_NCHAR_CONV_EXCP	Determines whether an error is reported when there is data loss during an implicit or explicit data type conversion between NCHAR/NVARCHAR2 data and CHAR/VARCHAR2 data.

DatabaseCharacterSet must be set during data store creation. There is no default. See "Choosing a database character set" on page 5-2.

The rest of the attributes are set during connection to a data store. For more information about ConnectionCharacterSet, see "Connection character set" on page 5-4.

You can use the ALTER SESSION statement to change the following attributes during a session:

- NLS_SORT
- NLS_LENGTH_SEMANTICS
- NLS_NCHAR_CONV_EXCP

For more information, see "ALTER SESSION" in the *Oracle TimesTen In-Memory Database SQL Reference* and "Data Store Attributes" in *Oracle TimesTen In-Memory Database Reference*.

Backward compatibility using TIMESTEN8

TIMESTEN8 is a restricted database character set that specifies behavior from TimesTen releases before 7.0. It is supported for backward compatibility only.

TIMESTEN8 has the following restrictions:

- There is no support for character set conversion of any kind. This includes:
  - Conversions between the application and the database. If DatabaseCharacterSet is TIMESTEN8, then ConnectionCharacterSet must also be TIMESTEN8.
  - Conversions between CHAR/VARCHAR2 data and NCHAR/NVARCHAR2 data.
- Sorting for CHAR and VARCHAR data types is limited to binary ordering. `NLS_SORT=BINARY` is the only sort allowed.
- TIMESTEN8 is not supported in IMDB Cache.

During database creation, customers should select the database character set matching the actual encoding of the data being stored in CHAR and VARCHAR2 columns whenever possible. Select TIMESTEN8 only when backwards compatibility to existing TimesTen data is required.

Globalization support during migration

The `ttMigrate` utility saves one or more migrate objects from a TimesTen data store into a binary data file or restores the objects from the binary data files into a TimesTen data store. Migrate objects include tables, cache group definitions, views and sequences.

This section includes the following topics:

- Object migration and character sets

- Migration and length semantics
- Migrating linguistic indexes
- Migrating cache group tables

See also "Copying, migrating, backing up and restoring a data store" on page 1-20 of this guide and the description of `ttMigrate` in *Oracle TimesTen In-Memory Database Reference*.

Object migration and character sets

`ttMigrate` tags each object it saves with the object's character set. By default, `ttMigrate` stores object data in the database character set, but you can choose a different character set by using the `-saveAsCharset` option. You can specify this option in create mode (`-c`) or append mode (`-a`).

When you use `ttMigrate` to restore an object, its data is implicitly converted to the database character set of the target data store if needed. Character set conversion can result in data loss if the database character set of the target data store cannot represent all of the data that it receives.

If you know that the data is in encoded in the database character set of the target data store, you can use the `-noCharsetConversion` option. This option can be specified only in restore mode (`-r`). If you use the `-noCharsetConversion` option, `ttMigrate` treats the data as if it is in the database character set of the target data store.

When you restore untagged character data from a data store that was created before release 7.0 into a data store from release 7.0 and later, `ttMigrate` treats the data as if it is in the database character set of the target data store.

`ttMigrate` issues a warning whenever there is an implicit or explicit character set conversion while saving or restoring data.

Migration and length semantics

`ttMigrate` saves length semantic information about CHAR and VARCHAR2 columns. It restores the length semantic information when restoring objects into data stores created in TimesTen release 7.0 or later.

When objects are migrated back into a TimesTen release before 7.0, columns with character semantics are converted to byte semantics and the column length is adjusted to match the byte length of the original columns.

When objects are migrated from a release before 7.0 to release 7.0 and later, byte semantics is used.

Migrating linguistic indexes

`ttMigrate` supports migration of linguistic indexes into TimesTen releases that support them. When migrating back to a TimesTen release before 7.0, `ttMigrate` issues a warning indicating that the linguistic indexes cannot be restored. Migration of the table proceeds without the linguistic indexes.

Migrating cache group tables

You cannot restore cache group tables containing NCHAR/NVARCHAR2 columns to a release before 7.0. Releases before 7.0 do not allow these data types in cache group tables.

Supported character sets

The tables in this section describe the character sets supported in TimesTen.

Asian character sets

Name	Description
JA16EUC	EUC 24-bit Japanese
JA16EUCTILDE	The same as JA16EUC except for the way that the wave dash and the tilde are mapped to and from Unicode
JA16SJIS	Shift-JIS 16-bit Japanese
JA16SJISTILDE	The same as JA16SJIS except for the way that the wave dash and the tilde are mapped to and from Unicode
KO16KSC5601	KSC5601 16-bit Korean
KO16MSWIN949	Microsoft Windows Code Page 949 Korean
TH8TISASCII	Thai Industrial Standard 620-2533 - ASCII 8-bit
VN8MSWIN1258	Microsoft Windows Code Page 1258 8-bit Vietnamese
ZHS16CGB231280	CGB2312-80 16-bit Simplified Chinese
ZHS16GBK	GBK 16-bit Simplified Chinese
ZHS32GB18030	GB18030-2000
ZHT16BIG5	BIG5 16-bit Traditional Chinese
ZHT16HKSCS	Microsoft Windows Code Page 950 with Hong Kong Supplementary Character Set HKSCS-2001. Character set conversion to and from Unicode is based on Unicode 3.0.
ZHT16MSWIN950	Microsoft Windows Code Page 950 Traditional Chinese
ZHT32EUC	EUC 32-bit Traditional Chinese

European character sets

Name	Description
BLT8CP921	Latvian Standard LVS8-92(1) Windows/UNIX 8-bit Baltic
BLT8ISO8859P13	ISO 8859-13 Baltic
BLT8MSWIN1257	Microsoft Windows Code Page 1257 8-bit Baltic
BLT8PC775	IBM-PC Code Page 775 8-bit Baltic
CEL8ISO8859P14	ISO 8859-13 Celtic
CL8ISO8859P5	ISO 8859-5 Latin/Cyrillic
CL8KOI8R	RELCOM Internet Standard 8-bit Latin/Cyrillic
CL8KOI8U	KOI8 Ukrainian Cyrillic
CL8MSWIN1251	Microsoft Windows Code Page 1251 8-bit Latin/Cyrillic
EE8ISO8859P2	ISO 8859-2 East European
EL8ISO8859P7	ISO 8859-7 Latin/Greek
ET8MSWIN923	Microsoft Windows Code Page 923 8-bit Estonian

Name	Description
EE8MSWIN1250	Microsoft Windows Code Page 1250 8-bit East European
EL8MSWIN1253	Microsoft Windows Code Page 1253 8-bit Latin/Greek
EL8PC737	IBM-PC Code Page 737 8-bit Greek/Latin
EE8PC852	IBM-PC Code Page 852 8-bit East European
LT8MSWIN921	Microsoft Windows Code Page 921 8-bit Lithuanian
NE8ISO8859P10	ISO 8859-10 North European
NEE8ISO8859P4	ISO 8859-4 North and North-East European
RU8PC866	IBM-PC Code Page 866 8-bit Latin/Cyrillic
SE8ISO8859P3	ISO 8859-3 South European
US7ASCII	ASCII 7-bit American
US8PC437	IBM-PC Code Page 437 8-bit American
WE8ISO8859P1	ISO 8859-1 West European
WE8ISO8859P15	ISO 8859-15 West European
WE8MSWIN1252	Microsoft Windows Code Page 1252 8-bit West European
WE8PC850	IBM-PC Code Page 850 8-bit West European
WE8PC858	IBM-PC Code Page 858 8-bit West European

Middle Eastern character sets

Name	Description
AR8ADOS720	Arabic MS-DOS 720 Server 8-bit Latin/Arabic
AR8ASMO8X	ASMO Extended 708 8-bit Latin/Arabic
AR8ISO8859P6	ISO 8859-6 Latin/Arabic
AR8MSWIN1256	Microsoft Windows Code Page 1256 8-Bit Latin/Arabic
AZ8ISO8859P9E	ISO 8859-9 Latin Azerbaijani
IW8ISO8859P8	ISO 8859-8 Latin/Hebrew
IW8MSWIN1255	Microsoft Windows Code Page 1255 8-bit Latin/Hebrew
TR8MSWIN1254	Microsoft Windows Code Page 1254 8-bit Turkish
TR8PC857	IBM-PC Code Page 857 8-bit Turkish
WE8ISO8859P9	ISO 8859-9 West European & Turkish

TimesTen character set

Name	Description
TIMESTEN8	TimesTen legacy character semantics

Universal character sets

Name	Description
AL16UTF16	Unicode 4.0 UTF-16 Universal character set. This is the implicit TimesTen national character set.
AL32UTF8	Unicode 4.0 UTF-8 Universal character set
UTF8	Unicode 3.0 UTF-8 Universal character set, CESU-8 compliant

Supported linguistic sorts

The tables in this section list the supported values for the NLS_SORT general connection attribute and the NLS_SORT SQL function.

Monolingual linguistic sorts

Basic name	Extended name
ARABIC	-
ARABIC_MATCH	-
ARABIC_ABJ_SORT	-
ARABIC_ABJ_ MATCH	-
ASCII7	-
AZERBAIJANI	XAZERBAIJANI
BENGALI	-
BIG5	-
BINARY	-
BULGARIAN	-
CANADIAN FRENCH	-
CATALAN	XCATALAN
CROATIAN	XCROATIAN
CZECH	XCZECH
CZECH_ PUNCTUATION	XCZECH_PUNCTUATION
DANISH	XDANISH
DUTCH	XDUTCH
EBCDIC	-
EEC_EURO	-
EEC_EUROPA3	-
ESTONIAN	-
FINNISH	-
FRENCH	XFRENCH

Basic name	Extended name
GERMAN	XGERMAN
GERMAN_DIN	XGERMAN_DIN
GBK	-
GREEK	-
HEBREW	-
HKSCS	-
HUNGARIAN	XHUNGARIAN
ICELANDIC	-
INDONESIAN	-
ITALIAN	-
LATIN	-
LATVIAN	-
LITHUANIAN	-
MALAY	-
NORWEGIAN	-
POLISH	-
PUNCTUATION	XPUNCTUATION
ROMANIAN	-
RUSSIAN	-
SLOVAK	XSLOVAK
SLOVENIAN	XSLOVENIAN
SPANISH	XSPANISH
SWEDISH	-
SWISS	XSWISS
THAI_DICTIONARY	-
TURKISH	XTURKISH
UKRAINIAN	-
UNICODE_BINARY	-
VIETNAMESE	-
WEST_EUROPEAN	XWEST_EUROPEAN

Multilingual linguistic sorts

Sort name	Description
CANADIAN_M	Canadian French sort supports reverse secondary, special expanding characters.
DANISH_M	Danish sort supports sorting uppercase characters before lowercase characters.
FRENCH_M	French sort supports reverse sort for secondary.

Sort name	Description
GENERIC_M	Generic sorting order which is based on ISO14651 and Unicode canonical equivalence rules but excluding compatible equivalence rules.
JAPANESE_M	Japanese sort supports SJIS character set order and EUC characters which are not included in SJIS.
KOREAN_M	Korean sort Hangul characters are based on Unicode binary order. Hanja characters based on pronunciation order. All Hangul characters are before Hanja characters.
SPANISH_M	Traditional Spanish sort supports special contracting characters.
THAI_M	Thai sort supports swap characters for some vowels and consonants.
SCHINESE_RADICAL_M	Simplified Chinese sort is based on radical as primary order and number of strokes order as secondary order.
SCHINESE_STROKE_M	Simplified Chinese sort uses number of strokes as primary order and radical as secondary order.
SCHINESE_PINYIN_M	Simplified Chinese Pinyin sorting order.
TCHINESE_RADICAL_M	Traditional Chinese sort based on radical as primary order and number of strokes order as secondary order.
TCHINESE_STROKE_M	Traditional Chinese sort uses number of strokes as primary order and radical as secondary order. It supports supplementary characters.

6 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:

- Managing XLA bookmarks

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

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:

```
C:\>ttIsql

ttIsql (c) 1996-2009, Oracle. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

Command>
```

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.

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

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

ttIsql (c) 1996-2009, Oracle. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

Command> connect "DSN=MY_DSN"
Connection successful: DSN=MY_DSN;DataStore=E:\ds\MY_DSN;
DRIVER=E:\WINNT\System32\TTdv1121.dll;
(Default setting AutoCommit=1)

Command> run "create.sql"

CREATE TABLE LOOKUP (KEY NUMBER 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:

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

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

Customizing the ttIsql command prompt

You can customize the `ttIsql` command prompt by using the `set` command with the `prompt` attribute:

```
Command> set prompt MY_DSN;
MY_DSN
```

You can specify a string format (`%c`) that returns the name of the current connection:

```
Command> set prompt %c;
con1
```

If you want to embed spaces, you must quote the string:

```
Command> set prompt "MY_DSN %c> ";
MY_DSN con1>
```

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.

```
Command> help connect disconnect

Arguments in <> are required.
Arguments in [] are optional.

Command Usage: connect [DSN|connection_string] [as <connection_id>]
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. A connection ID may
optionally be specified, for use in referring to the connection when multiple
connections are enabled. The DSN is used as the default connection ID. If that ID
is already in use, the connection will be assigned the ID "conN", where N is some
number larger than 0.
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";
-or- connect RunData as rundata1;

Command Usage: disconnect [all]
Command Aliases: (none)
Description: Disconnects from the currently connected data source or all
connections when the "all" argument is included. 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: NO
Requires autocommit turned off: NO
Reports elapsed execution time: YES
Works only with a TimesTen data source: NO
Example: disconnect;
```

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.

To view the list of attributes that can be set or shown by using `ttIsql`, enter:

```
Command> help attributes
```

Using ttIsql's 'editline' feature for 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.

To disable the 'editline' feature in `ttIsql`, use the `ttIsql` command `set editline off`.

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

- Emacs binding
- vi binding

Emacs binding

To use the emacs binding, 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. Move 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.

Keystroke	Action
<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 ("`h` ") to view the list of previously executed commands. For example:

```
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:

```
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:

```
Command> ! auto;

autocommit 0
Command>
```

Saving and clearing ttIsql's command history

You can save the list of commands that `ttIsql` stores by using the `savehistory` command:

```
Command> savehistory history.txt;
```

If the output file already exists, use the `-a` option to append the new command history to the file or the `-f` option to overwrite the file. The next example shows how to append new command history to an existing file.

```
Command> savehistory -a history.txt;
```

You can clear the list of commands that `ttIsql` stores by using the `clearhistory` command:

```
Command> clearhistory;
```

Working with character sets

The `ttIsql` utility supports the character sets listed in "Supported character sets" on page 5-10. The ability of `ttIsql` to display characters depends on the native OS locale settings of the terminal on which you are using `ttIsql`.

To override the locale-based output format, use the `ncharencoding` option or the `-N` option. The valid values for these options are LOCALE (the default) and ASCII. If you choose ASCII and `ttIsql` encounters a Unicode character, it displays it in escaped format.

You do not need to have an active connection to change the output method.

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 Guide* and "Setting a timeout value for executing SQL statements" in the *Oracle TimesTen In-Memory Database C Developer's Guide*.

Note: TimesTen rollback and query timeout features do not stop Cache Connect operations that are being processed on Oracle. This includes passthrough statements, flushing, manual loading, manual refreshing, synchronous writethrough, propagating and dynamic loading.

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

```
E:\>ttIsql
```

```
ttIsql (c) 1996-2009, Oracle. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

Command> connect "DSN=MY_DSN";
Connection successful: DSN=MY_DSN;DataStore=E:\ds\MY_DSN;DRIVER=E:\WINNT\System32\
TTdv1121.dll;
(Default setting AutoCommit=1)
Command> autocommit 0;
Command> CREATE TABLE LOOKUP (KEY NUMBER 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 `ttIsql` commands that display information on data store structures. The most useful commands are summarized below:

- `describe` - Displays information on tables, views, sequences, PL/SQL functions, PL/SQL procedures, PL/SQL packages, prepared SQL statements and built-in 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.

Use the `describe` command 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, a PL/SQL function, PL/SQL procedure or PL/SQL package. The `describe` command requires a semicolon character to terminate the command.

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

Table USER.T1:
  Columns:
   *KEY                           NUMBER 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                   NUMBER
  Columns:
    KEY NUMBER                    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. The argument to the `cachegroups` command is the name of the cache group that you want to display information for.

```
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.

```
Command> monitor;
TIME_OF_1ST_CONNECT: Mon Feb 23 11:32:49 2009
DS_CONNECTS: 11
DS_DISCONNECTS: 0
DS_CHECKPOINTS: 0
DS_CHECKPOINTS_FUZZY: 0
DS_COMPACTS: 0
PERM_ALLOCATED_SIZE: 40960
```

```
PERM_IN_USE_SIZE: 5174
PERM_IN_USE_HIGH_WATER: 5174
TEMP_ALLOCATED_SIZE: 18432
TEMP_IN_USE_SIZE: 4527
TEMP_IN_USE_HIGH_WATER: 4527
SYS18: 0
TPL_FETCHES: 0
TPL_EXECS: 0
CACHE_HITS: 0
PASSTHROUGH_COUNT: 0
XACT_BEGINS: 2
XACT_COMMITS: 1
XACT_D_COMMITS: 0
XACT_ROLLBACKS: 0
LOG_FORCES: 0
DEADLOCKS: 0
LOCK_TIMEOUTS: 0
LOCK_GRANTS_IMMED: 17
LOCK_GRANTS_WAIT: 0
SYS19: 0
CMD_PREPARES: 1
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: 64
LOG_FS_READS: 0
LOG_FS_WRITES: 0
LOG_BUFFER_WAITS: 0
CHECKPOINT_BYTES_WRITTEN: 0
CURSOR_OPENS: 1
CURSOR_CLOSES: 1
SYS3: 0
SYS4: 0
SYS5: 0
SYS6: 0
CHECKPOINT_BLOCKS_WRITTEN: 0
CHECKPOINT_WRITES: 0
REQUIRED_RECOVERY: 0
SYS11: 0
SYS12: 1
TYPE_MODE: 0
SYS13: 0
SYS14: 0
SYS15: 0
SYS16: 0
SYS17: 0
SYS9:
```

Creating and executing PL/SQL blocks

You can create and execute PL/SQL blocks from the `ttIsql` command line.

Set `serveroutput` on to display results generated from the PL/SQL block:

```
Command> set serveroutput on
```

Create an anonymous block that puts a text line in the output buffer. Note that the block must be terminated with a slash (/).

```
Command> BEGIN
       > DBMS_OUTPUT.put_line(
       >   'Welcome!');
       > END;
       > /
Welcome!
PL/SQL procedure successfully completed.
Command>
```

See "Introduction to PL/SQL in the TimesTen Database" in *Oracle TimesTen In-Memory Database PL/SQL Developer's Guide* for more examples.

Using OUT parameters

You can pass data back to applications from PL/SQL by using OUT parameters. This example returns information about how full a TimesTen data store is.

Create the `tt_space_info` PL/SQL procedure and use SQL to provide values for the `permpct`, `permmaxpct`, `tempct`, and `tempmaxpct` parameters.

```
Command> CREATE OR REPLACE PROCEDURE tt_space_info
       >   (permpct    OUT PLS_INTEGER,
       >    permmaxpct OUT PLS_INTEGER,
       >    temppct    OUT PLS_INTEGER,
       >    tempmaxpct OUT PLS_INTEGER) AS
       >    monitor    sys.monitor%ROWTYPE;
       > BEGIN
       >   SELECT * INTO monitor FROM sys.monitor;
       >   permpct := monitor.perm_in_use_size * 100 / monitor.perm_allocated_
size;
       >   permmaxpct := monitor.perm_in_use_high_water * 100 / monitor.perm_
allocated_size;
       >   temppct := monitor.temp_in_use * 100 / monitor.temp_allocated_size;
       >   tempmaxpct := monitor.temp_in_use_high_water * 100 / monitor.temp_
allocated_size;
       > END;
       >/

Procedure created.
```

Declare the variables and call `tt_space_info`. The parameter values are passed back to `ttIsql` so they can be printed:

```
Command> VAR permpct NUMBER
Command> VAR permpctmax NUMBER
Command> VAR temppct NUMBER
Command> VAR temppctmax NUMBER
Command> BEGIN
       >   tt_space_info(:permpct, :permpctmax, :temppct, :temppctmax);
       > END;
       >/

PL/SQL procedure successfully completed.

Command> PRINT permpct;
PERMPCT              : 4
```

```
Command> PRINT permpctmax;
PERMPCTMAX           : 4

Command> PRINT temppct;
TEMPPCT              : 11

Command> PRINT temppctmax;
TEMPPCTMAX           : 11
```

You can also pass back a statement handle that can be executed by a PL/SQL statement with an OUT refcursor parameter. The PL/SQL statement can choose the query associated with the cursor. The following example opens a refcursor, which randomly chooses between ascending or descending order.

```
Command> VARIABLE ref REFCURSOR;
Command> BEGIN
    >    IF (mod(dbms_random.random(), 2) = 0) THEN
    >     open :ref for select object_name from SYS.ALL_OBJECTS order by 1 asc;
    >    ELSE
    >     open :ref for select object_name from SYS.ALL_OBJECTS order by 1 desc;
    >    end if;
    >   END;
    >   /

PL/SQL procedure successfully completed.
```

To fetch the result set from the refcursor, use the PRINT command:

```
Command> PRINT ref
REF           :
< ACCESS$ >
< ALL_ARGUMENTS >
< ALL_COL_PRIVS >
< ALL_DEPENDENCIES >
...
143 rows found.
```

Or if the result set was ordered in descending order, the following would print:

```
Command> PRINT ref
REF          :
< XLASUBSCRIPTIONS >
< WARNING_SETTINGS$ >
< VIEWS >
...
143 rows found.
```

Displaying information about PL/SQL objects

You can use `ttIsql` to list PL/SQL functions, procedures and packages in a data store. Commands prefixed by `all` display all objects. For example, the `functions` command lists PL/SQL functions that are owned by the user, whereas `allfunctions` lists all PL/SQL functions. You can optionally specify patterns for object owners and object names.

Use these commands to list PL/SQL objects:

- `functions` and `allfunctions` - Lists PL/SQL functions
- `procedures` and `allprocedures` - Lists PL/SQL procedures

- `packages` and `allpackages` - Lists PL/SQL packages

This example shows that user `terry` creates a procedure called `proc1` while connected to `myDSN`. Note that a slash character (/) is entered on a new line following the PL/SQL statements.

The `procedures` command and the `allprocedures` command show that it is the only PL/SQL procedure in the data store.

```
$ ttisql myDSN
Copyright (c) 1996-2009, Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.
connect "DSN=myDSN";
Connection successful:
DSN=myDSN;UID=terry;DataStore=/scratch/terry/myDSN;DatabaseCharacter
Set=AL32UTF8;ConnectionCharacterSet=US7ASCII;PermSize=32;TypeMode=0;
(Default setting AutoCommit=1)
Command> create or replace procedure proc1 as begin null; end;
       > /
Procedure created.
Command> procedures;
  TERRY.PROC1
1 procedure found.
Command> allprocedures;
  TERRY.PROC1
1 procedure found.
```

Now connect to the same DSN as `pat` and create a procedure called `q`. The `allprocedures` command shows the PL/SQL procedures created by `terry` and `pat`.

```
$ ttisql "dsn=myDSN;uid=PAT"
Copyright (c) 1996-2009, Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.
connect "dsn=myDSN;uid=PAT";
Connection successful:
DSN=myDSN;UID=PAT;DataStore=/scratch/terry/myDSN;DatabaseCharacterSet=AL32UTF8;
ConnectionCharacterSet=US7ASCII;PermSize=32;TypeMode=0;
(Default setting AutoCommit=1)
Command> create or replace procedure q as begin null; end;
       > /
Procedure created.
Command> procedures;
  PAT.Q
1 procedure found.
Command> allprocedures;
  TERRY.PROC1
  PAT.Q
2 procedures found.
```

Viewing and setting data store attributes

You can view and set data store attributes with the `ttIsql` `show` and `set` commands. For a list of the attributes that you can view and set with `ttIsql`, see "Data Store Attributes" in *Oracle TimesTen In-Memory Database Reference*.

To view the setting for the Passthrough attribute, enter:

```
Command> show passthrough;
PassThrough = 0
```

To change the Passthrough setting, enter:

```
Command> set passthrough 1;
```

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 "The TimesTen Query Optimizer" on page 10-1.

- `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 range indexes.
- `tryttree` - Enables or disables the use of range 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 autocommit feature must be turned off. Use `ttIsql`'s `autocommit` built-in command to turn autocommit off.

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

```
Command> CREATE TABLE T1 (A NUMBER);
Command> CREATE TABLE T2 (B NUMBER);
Command> CREATE TABLE T3 (C NUMBER);
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.

```
Command> timing 1;
Command> connect "DSN=MY_DSN";
Connection successful: DSN=MY_DSN;DataStore=E:\ds\MY_DSN;DRIVER=E:\WINNT\System32\
TTdv1121.dll;
(Default setting AutoCommit=1)
Execution time (SQLDriverConnect) = 1.2626 seconds.
Command>
```

In the example above, the `SQLDriverConnect` call took about 1.26 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.

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

```
Command> connect "DSN=MY_DSN";
Connection successful: DSN=MY_DSN;DataStore=E:\ds\MY_DSN;DRIVER=E:\WINNT\Sys
tem32\TTdv1121.dll;
(Default setting AutoCommit=1)
Command> timing 1;
Command> create table t1 (key number not null primary key, value char(20));
Execution time (SQLExecute) = 0.007247 seconds.
Command> prepare insert into t1 values (:f, :g);
Execution time (SQLPrepare) = 0.000603 seconds.
Command> exec;

Type '?' for help on entering parameter values.
Type '*' to end prompting and abort the command.
Type '-' to leave the parameter unbound.
Type '/' to leave the remaining parameters unbound and execute the command.

Enter Parameter 1 'F' (NUMBER) > 1;
Enter Parameter 2 'G' (CHAR) > 'abc';
1 row inserted.
Execution time (SQLExecute) = 0.000454 seconds.
Command> exec;

Type '?' for help on entering parameter values.
Type '*' to end prompting and abort the command.
Type '-' to leave the parameter unbound.
Type '/' to leave the remaining parameters unbound and execute the help command.

Enter Parameter 1 'F' (NUMBER) > 2;
Enter Parameter 2 'G' (CHAR) > 'def';
1 row inserted.
Execution time (SQLExecute) = 0.000300 seconds.
Command> free;
Command> select * from t1;
< 1, abc                  >
< 2, def                  >
2 rows found.
Execution time (SQLExecute + Fetch Loop) = 0.000226 seconds.
Command> disconnect;
Disconnecting...
Execution time (SQLDisconnect) = 2.911396 seconds.
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. The `ttIsql` utility 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 Reference* or type `help` at the `ttIsql` command prompt.

The example below prepares and executes a SELECT statement with a predicate containing one NUMBER 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.

```
Command> connect "DSN=MY_DSN";
Connection successful: DSN=MY_DSN;DataStore=E:\ds\MY_DSN;DRIVER=E:\WINNT\Sys
tem32\TTdv1121.dll;
(Default setting AutoCommit=1)
The command succeeded.
Command> CREATE TABLE T1 (KEY NUMBER 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.
Type '?;' for help on entering parameter values.
Type '*;' to abort the parameter entry process.

Enter Parameter 1 (NUMBER) >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>
```

Note: For information about using `ttIsql` with PL/SQL host variables, see "Introduction to PL/SQL in the TimesTen Database" in *Oracle TimesTen In-Memory Database PL/SQL Developer's Guide*.

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 the `ttIsql` command line options, see the *Oracle TimesTen In-Memory Database Reference*.

```
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.

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;dssize;"
```

In this example, `ttIsql` will automatically connect to a DSN called `MY_DSN`, turn off autocommit and display the size of the data store as shown below:

```
C:\>ttIsql

ttIsql (c) 1996-2009, Oracle. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

Command> connect "DSN=MY_DSN";
Connection successful: DSN=MY_DSN;DataStore=E:\ds\MY_
DSN;DRIVER=E:\WINNT\System32\TTdv1121.dll;
```

```
(Default setting AutoCommit=1)
Command> autocommit 0;

Command> alltables;
  SYS.ACCESS$
  SYS.ARGUMENT$
  SYS.CACHE_GROUP
  SYS.COLUMNS
  SYS.COLUMN_HISTORY
  SYS.COL_STATS
  SYS.DEPENDENCY$
  SYS.DIR$
  SYS.DUAL
  SYS.ERROR$
  SYS.IDL_CHAR$
  SYS.IDL_SB4$
  SYS.IDL_UB1$
  SYS.IDL_UB2$
  SYS.INDEXES
  SYS.MONITOR
...
59 tables found.
Command>
```

Managing XLA bookmarks

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

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

```
Command> xlabookmarkdelete;

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> xlabookmarkdelete xlaSimple;
Command>
```

7

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 SQL Reference*.

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

This chapter includes the following topics:

- Data store overview
- Understanding tables
- Understanding views
- Understanding materialized views
- Understanding indexes
- Understanding rows

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 7-2.
- **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 7-11.
- **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 7-10.

- **Indexes**. Indexes on one or more columns of a table may be created for faster access to tables. See "Understanding indexes" on page 7-21.
- **Rows**. Every table consists of 0 or more rows. A row is a formatted list of values. See "Understanding rows" on page 7-23.
- **System tables**. System tables contain TimesTen metadata, such as a table of all tables. See "System and Replication Tables" in the *Oracle TimesTen In-Memory Database SQL Reference*.

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

Data store users and owners

When Access Control is enabled, the TimesTen Data Manager authenticates user names with passwords. TimesTen Client/Server also authenticates 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 `ttCkptBlocking` built-in procedures described in the *Oracle TimesTen In-Memory Database Reference*.
- **Temporary data stores** are not stored to disk. A temporary data store is automatically destroyed when no applications are connected to it. TimesTen removes all disk-based files, when the last application disconnects.

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.

The following sections describes tables, its columns and how to manage them:

- Overview of tables
- Working with tables

- Implementing aging in your tables

Overview of tables

This section includes the following topics:

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

Column overview

When you create the columns in the table, the column names are case-insensitive.

Each column has the following:

- 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 drop the table and then recreate it with the new definitions.

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 VARBINARY, NVARCHAR and VARCHAR 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, VARCHAR, NVARCHAR and VARBINARY columns whose declared column length is > 128 bytes are stored out of line. Columns whose declared column length is <= 128 bytes are stored inline.

The maximum sizes of in-line and out-of-line portions of a row are listed in "Estimating table size" on page 7-6.

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 for any column type
- A constant value
- SYSDATE for DATE and TIMESTAMP columns
- USER for CHAR columns
- CURRENT_USER for CHAR columns
- SYSTEM_USER for CHAR columns

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 the *Oracle TimesTen In-Memory Database SQL Reference*.

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 ("."), such as `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. 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 "Names and parameters" in the *Oracle TimesTen In-Memory Database SQL Reference* 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 9-11.

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 range 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 7-21. Once a row is inserted, its primary key columns cannot be modified, except to change a range index to a hash index.

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 SQL Reference* for more information.

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 or uniquely indexed columns in the other table. Foreign keys use a range index on the referencing columns. See "CREATE TABLE" in the *Oracle TimesTen In-Memory Database SQL Reference* for more information.

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 SQL Reference*.

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

Working with tables

To perform any operation that creates, drops or manages a table, the user must have the appropriate privileges, which are described along with the syntax for all SQL statements in the "SQL Statements" chapter in the *Oracle TimesTen In-Memory Database SQL Reference*.

This section includes the following topics:

- Creating a table
- Dropping 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 SQL Reference*. TimesTen converts table names to upper case characters.

Example 7–1 Create a table

The following SQL statement creates a table, called `NameID`, with two columns: `CustId` and `CustName` of two different data types.

```
CREATE TABLE NameID (CustId TT_INTEGER, CustName VARCHAR2(50));
```

Example 7–2 Create a table with a hash index

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 7-4. The `UNIQUE HASH ON custId PAGES` value indicates that there are 30 pages in the hash index. This number is used to determine the number of buckets that are to be allocated for the table's hash index. Bucket count = (PAGES * 256) / 20. Therefore the number of buckets allocated for the hash index is 384: (30 * 256) / 20 = 384

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

Dropping a table

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

Example 7–3 Drop a table

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. 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.

Implementing aging in your tables

You can define an aging policy for one or more tables in your data store. An aging policy refers to the type of aging and the aging attributes, as well as the aging state (ON or OFF). You can specify one of the following types of aging policies: usage-based or time-based. Usage-based aging removes least recently used (LRU) data within a specified data store usage range. Time-based aging removes data based on the specified data lifetime and frequency of the aging process. You can define both usage-based aging and time-based aging in the same data store, but you can define only one type of aging on a specific table.

You can define an aging policy for a new table with the CREATE TABLE statement. You can add an aging policy to an existing table with the ALTER TABLE statement if the table does not already have an aging policy defined. You can change the aging policy by dropping aging and adding a new aging policy.

You cannot specify aging on the following types of tables:

- Global temporary tables
- Detail tables for materialized views

You can also implement aging in cache groups. See "Implementing aging on a cache group" in the *Oracle In-Memory Database Cache User's Guide*.

This section includes the following topics:

Usage-based aging

Usage-based aging enables you to maintain the amount of memory used in a data store within a specified threshold by removing the least recently used (LRU) data.

Define LRU aging for a new table by using the AGING LRU clause of the CREATE TABLE statement. Aging begins automatically if the aging state is ON.

Use the `ttAgingLRUConfig` built-in procedure to specify the LRU aging attributes. The attribute values apply to all tables in the data store that have an LRU aging policy. If you do not call the `ttAgingLRUConfig` built-in procedure, then the default values for the attributes are used.

Note: The `ttAgingLRUConfig` built-in procedure requires that the user have ADMIN privilege if you want to modify any attributes. You do not need any privileges for viewing existing attributes. For more information, see "Built-In Procedures" in the *Oracle TimesTen In-Memory Database Reference*.

The following table summarizes the LRU aging attributes:

LRU Aging Attribute	Description
LowUsageThreshhold	The percent of the data store PermSize at which LRU aging is deactivated.
HighUsageThreshhold	The percent of the data store PermSize at which LRU aging is activated.
AgingCycle	The number of minutes between aging cycles.

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

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

- The row is used to build the result set of a SELECT statement.
- The row has been flagged to be updated or deleted.
- The row is used to build the result set of an INSERT SELECT statement.

You can use the ALTER TABLE statement to perform the following tasks:

- Enable or disable the aging state on a table that has an aging policy defined by using the ALTER TABLE statement with the SET AGING {ON | OFF} clause.
- Add an LRU aging policy to an existing table by using the ALTER TABLE statement with the ADD AGING LRU [ON | OFF] clause.
- Drop aging on a table by using the ALTER TABLE statement with the DROP AGING clause.

Use the `ttAgingScheduleNow` built-in procedure to schedule when aging starts. For more information, see "Scheduling when aging starts" on page 7-9.

To change aging from LRU to time-based on a table, first drop aging on the table by using the ALTER TABLE statement with the DROP AGING clause. Then add time-based aging by using the ALTER TABLE statement with the ADD AGING USE clause.

> **Note:** When you drop LRU aging or add LRU aging to tables that are referenced in commands, TimesTen marks the compiled commands invalid. The commands need to be recompiled.

Time-based aging

Time-based aging removes data from a table based on the specified data lifetime and frequency of the aging process. Specify a time-based aging policy for a new table with the AGING USE clause of the CREATE TABLE statement. Add a time-based aging policy to an existing table with the ADD AGING USE clause of the ALTER TABLE statement.

The AGING USE clause has a `ColumnName` argument.`ColumnName` is the name of the column that is used for time-based aging. For brevity, we will call it the *timestamp column*. The timestamp column must be defined as follows:

- ORA_TIMESTAMP, TT_TIMESTAMP, ORA_DATE or TT_DATE data type
- NOT NULL

Your application updates the values of the timestamp column. If the value of this column is unknown for some rows and you do not want the rows to be aged, then define the column with a large default value. You can create an index on the timestamp column for better performance of the aging process.

> **Note:** You cannot add or modify a column in an existing table and then use that column as a timestamp column because you cannot add or modify a column and define it to be NOT NULL.

You cannot drop the timestamp column from a table that has a time-based aging policy.

If the data type of the timestamp column is ORA_TIMESTAMP, TT_TIMESTAMP, or ORA_DATE, you can specify the lifetime in days, hours, or minutes in the LIFETIME clause of the CREATE TABLE statement. If the data type of the timestamp column is TT_DATE, specify the lifetime in days.

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

Use the CYCLE clause to indicate how often the system should examine the rows to remove data that has exceeded the specified lifetime. If you do not specify CYCLE, aging occurs every five minutes. If you specify 0 for the cycle, then aging is continuous. Aging begins automatically if the state is ON.

Use the ALTER TABLE statement to perform the following tasks:

- Enable or disable the aging state on a table with a time-based aging policy by using the SET AGING {ON|OFF} clause.

- Change the aging cycle on a table with a time-based aging policy by using the SET AGING CYCLE clause.
- Change the lifetime by using the SET AGING LIFETIME clause.
- Add time-based aging to an existing table with no aging policy by using the ADD AGING USE clause.
- Drop aging on a table by using the DROP AGING clause.

Use the `ttAgingScheduleNow` built-in procedure to schedule when aging starts. For more information, see "Scheduling when aging starts" on page 7-9

To change the aging policy from time-based aging to LRU aging on a table, first drop time-based aging on the table. Then add LRU aging by using the ALTER TABLE statement with the ADD AGING LRU clause.

Aging and foreign keys

Tables that are related by foreign keys must have the same aging policy.

If LRU aging is in effect and a row in a child table has been recently accessed, then neither the parent row nor the child row will be deleted.

If time-based aging is in effect and a row in a parent table is a candidate for aging out, then the parent row and all of its children will be deleted.

If a table has ON DELETE CASCADE enabled, the setting is ignored.

Scheduling when aging starts

Use the `ttAgingScheduleNow` built-in procedure to schedule the aging process. The aging process starts as soon as you call the procedure unless there is already an aging process in progress, in which case it will begin when that aging process has completed.

When you call `ttAgingScheduleNow`, the aging process starts regardless of whether the state is ON or OFF.

The aging process starts only once as a result of calling `ttAgingScheduleNow` does not change the aging state. If the aging state is OFF when you call `ttAgingScheduleNow`, then the aging process starts, but it does not continue after the process is complete. To continue aging, you must call `ttAgingScheduleNow` again or change the aging state to ON.

If the aging state is already set to ON, then `ttAgingScheduleNow` resets the aging cycle based on the time ttAgingScheduleNow was called.

You can control aging externally by disabling aging by using the ALTER TABLE statement with the SET AGING OFF clause. Then use `ttAgingScheduleNow` to start aging at the desired time.

Use `ttAgingScheduleNow` to start or reset aging for an individual table by specifying its name when you call the procedure. If you do not specify a table name, then `ttAgingScheduleNow` will start or reset aging on all of the tables in the data store that have aging defined.

Aging and replication

For active standby pairs, implement aging on the active master data store. Deletes that occur as a result of aging will be replicated to the standby master data store and the read-only subscribers. If a failover to the standby master data store occurs, aging is enabled on the data store after its role changes to ACTIVE.

For all other types of replication schemes, implement aging separately on each node. The aging policy must be the same on all nodes.

If you implement LRU aging on a multimaster replication scheme used as a hot standby, LRU aging may provide unintended results. After a failover, you may not have all of the desired data because aging occurs locally.

Understanding views

A *view* is a logical table that is based on one or more 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 *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.

To perform any operation that creates, drops or manages a view, the user must have the appropriate privileges, which are described along with the syntax for all SQL statements in the "SQL Statements" chapter in the *Oracle TimesTen In-Memory Database SQL Reference*.

This section includes the following topics:

- Creating a view
- Dropping a view
- Restrictions on views and their detail tables

Creating a view

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

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

This selects columns from the detail tables to be used in 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 "SQL Statements" in the *Oracle TimesTen In-Memory Database SQL Reference*, 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
- Arguments
- Temporary tables
- 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.

Dropping a view

The DROP VIEW statement deletes the specified view.

The following statement drops the `CustOrder` view:

```
DROP VIEW CustOrder;
```

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.

Understanding materialized views

The following sections describes materialized views and how to manage them:

- Overview of materialized views
- Working with materialized views

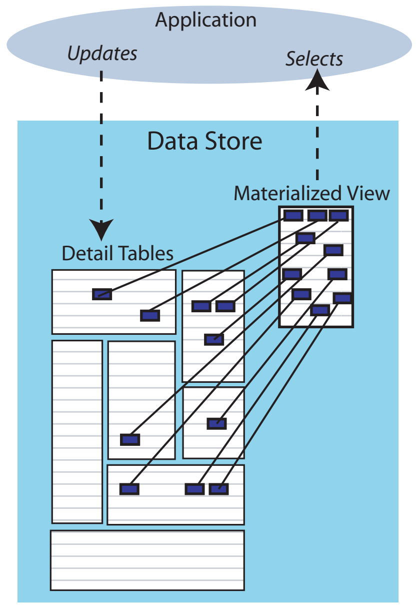
Overview of materialized views

A materialized view is a read-only table that maintains a summary of data selected from one or more regular TimesTen tables. The TimesTen tables queried to make up the result set for the materialized view are called detail tables.

Note: Materialized views are not supported on cache tables.

Figure 7–1 shows a materialized view created from detail tables. An application updates the detail tables and can select data from the materialized view.

Figure 7–1 Materialized view



There are two types of materialized views based upon how the result set for the materialized view is updated.

- Synchronous materialized view
- Asynchronous materialized view

In addition, learn when to use each type of materialized views in the section: "When to use synchronous or asynchronous materialized views" on page 7-13.

Synchronous materialized view

The synchronous materialized view, by default, updates the result set data from the detail tables at the time of the detail table transaction. Every time data is updated in the detail tables, the result set is updated. Thus, the synchronous materialized view is never out of sync with the detail tables. However, this can affect your performance. A single transaction, the user transaction, executes the updates for both the detail table and any synchronous materialized views.

Asynchronous materialized view

The materialized view is populated and it is in sync with the detail tables at creation. When the detail tables are updated, the asynchronous materialized views are not updated immediately. At any moment, they can be out of sync with the corresponding detail tables. The asynchronous materialized view defers updates to the result set as a trade-off for performance. You decide when and how the result set is refreshed either manually by the user or automatically within a pre-configured interval. The asynchronous materialized view is always refreshed in its own transaction, not within the user transaction that updates the detail tables. Thus, the user transaction is not blocked by any updates for the asynchronous materialized view.

The asynchronous refresh may use either of the following refresh method configurations:

- `FAST`, which updates only the incremental changes since the last update.
- `COMPLETE`, which provides a full refresh.

To facilitate a `FAST` refresh, you must create a materialized view log to manage the deferred incremental transactions for each detail table used by the asynchronous materialized view. Each detail table requires only one materialized view log for managing all deferred transactions, even if it is included in more than one `FAST` asynchronous materialized view.

The detail table cannot be dropped if there is an associated materialized view or materialized view log.

Note: When you use XLA in conjunction with asynchronous materialized views, you cannot depend on the ordering of the DDL statements. In general, there are no operational differences between the XLA mechanisms used to track changes to a table or a materialized view. However, for asynchronous materialized views, be aware that the order of XLA notifications for an asynchronous view is not necessarily the same as it would be for the associated detail tables, or the same as it would be for asynchronous view. For example, if there are two inserts to a detail table, they may be done in the opposite order in the asynchronous materialized view. Furthermore, updates may be treated as a delete followed by an insert, and multiple operations, such as multiple inserts or multiple deletes, may be combined. Applications that depend on ordering should not use asynchronous materialized views.

When to use synchronous or asynchronous materialized views

The following sections provide guidelines on when to use synchronous or asynchronous materialized views:

- Joins and aggregate functions turn into super locks
- Freshness of the materialized view
- Overhead cost

Joins and aggregate functions turn into super locks If a synchronous materialized view has joins or uses aggregate functions, there is a super lock effect. For example, if you have a single table with a synchronous materialized view that aggregates on average 1000 rows into 1. When you update a row in the detail table of the synchronous materialized view, you lock that row for the remainder of the transaction. Any other transaction that attempts to update that row blocks and waits until the transaction commits.

But since there is a synchronous materialized view on that table, the materialized view is also updated. The single row in the materialized view is locked and updated to reflect the change. However, there are 999 other rows from the base table that also aggregate to that same materialized view row. These 999 other base table rows are also effectively locked because if you try to update any of them, you will block and wait while retrieving the lock on the materialized view row. This is referred to as a super lock.

The same effect occurs across joins. If you have a synchronous materialized view that joins five tables and you update a row in any one of the five tables, you will have a super lock on all the rows in the other four tables that join to the one that you updated.

Obviously, the combination of joins and aggregate functions compound the problem for synchronous materialized views. However, asynchronous materialized views with COMPLETE refresh diminish the super lock because the locks on the asynchronous materialized view rows with COMPLETE refresh are only held during the refresh process. The super locks with synchronous materialized views will be held until the updating transaction commits. Thus, if you have short transactions, then super locks on synchronous materialized view are not a problem. However, if you have long transactions, use asynchronous materialized views with COMPLETE refresh that minimize the effect of any super lock.

Freshness of the materialized view Synchronous materialized views are always fresh and they always return the latest data. Asynchronous materialized views can become stale after an update until refreshed. If you must have the most current data all the time, use synchronous materialized views. However, you may consider using asynchronous if your application does not need the most current data.

For example, you may execute a series of analytical queries each with variations. In this case, you can use an asynchronous materialized view to isolate the differences that result from the query variations from the differences that result from newly arrived or updated data.

Overhead cost An asynchronous materialized view is not updated in the user transaction, which updates the detail tables. The refresh of an asynchronous materialized view is always performed in an independent transaction. This means that the user is free to execute any other transaction. By comparison, for synchronous materialized views, a single transaction executes the updates for both the detail table and any synchronous materialized views, which does affect your performance.

While the asynchronous materialized view logs for asynchronous materialized views with FAST refresh incur overhead, it is generally less overhead than the cost of updating a synchronous materialized view. This is especially true even if the asynchronous materialized view is complicated with joins. For asynchronous materialized views with COMPLETE refresh, there is no overhead at the time of updating the detail table.

You can defer asynchronous materialized view maintenance cost. The asynchronous materialized view log costs less than the incremental maintenance of the synchronous materialized view because the asynchronous materialized view logs perform simple inserts, whereas synchronous materialized view maintenance has to compute the delta for the materialized view and joins and then apply results in an update operation. Updates are more expensive than inserts. The cost difference reduces if the synchronous materialized view is simple in structure.

Working with materialized views

This section includes the following topics:

- Creating a materialized view
- Dropping a materialized view or a materialized view log
- Restrictions on materialized views and detail tables

- Performance implications of materialized views

Creating a materialized view

To create a materialized view, use the SQL statement CREATE MATERIALIZED VIEW. In order to create a materialized view, the user must have the appropriate privileges, which are described along with the syntax for all SQL statements in the "SQL Statements" chapter in the *Oracle TimesTen In-Memory Database SQL Reference*.

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 7-4.

The materialized view examples are based on the following two tables:

```
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,
  book char(100),
  PRIMARY KEY (orderId),
  FOREIGN KEY (custId) REFERENCES Customer(custId));
```

The following sections provide details and examples for creating materialized views:

- Creating a synchronous materialized view
- Creating an asynchronous materialized view
- The SELECT query in the CREATE MATERIALIZED VIEW statement

Creating a synchronous materialized view A synchronous materialized view is automatically updated each time the detail tables are updated. You can create a synchronous materialized view with the CREATE MATERIALIZED VIEW statement.

The following creates a synchronous 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, orderId, book
 FROM customer, bookOrder
 WHERE customer.custId=bookOrder.custId;
```

Creating an asynchronous materialized view An asynchronous materialized view is updated as specified by the refresh method and refresh interval, which are configured during the creation of the materialized view.

When you create an asynchronous materialized view, you specify the REFRESH clause with at least one of the following:

- Refresh method: For the asynchronous materialized view, specify either `FAST` or `COMPLETE` for the refresh method. `FAST` denotes an incremental refresh. `COMPLETE` indicates a full refresh. If the refresh method is omitted, then `COMPLETE` is the default refresh method. If you specify `FAST`, then you must create the asynchronous materialized view log for each detail table associated with the materialized view.

Note: Aggregate functions and outer joins are not supported in a `FAST` refresh.

- Refresh interval:
  - Manual update: If the refresh interval is not specified, the interval defaults to manual update. You can manually refresh the view by using the `REFRESH MATERIALIZED VIEW` statement, which is described at the end of this section.
  - Specify refresh after every commit: When you specify `NEXT SYSDATE` without specifying `NUMTODSINTERVL()`, the refresh is performed after every commit of any user transaction that updates the detail tables. This refresh is always performed in a separate transaction. The user transaction does not wait for the refresh to complete. The option to refresh at every commit is only supported for the FAST refresh method.
  - Specify interval: The asynchronous materialized view is updated at a specified interval when you use the `NEXT SYSDATE + NUMTODSINTERVAL(`*`IntegerLiteral`*`,`*`IntervalUnit`*`)` clause. This option is supported for both FAST and COMPLETE refresh methods.

    This clause specifies that the materialized view will be refreshed at the specified interval. *`IntegerLiteral`* must be an integer. *`IntervalUnit`* must be one of the following values: `'DAY'`, `'HOUR'`, `'MINUTE'`, `'SECOND'`.

    The last refresh time is saved in order to determine the next refresh time. Refresh is skipped if there are no changes to the any of the detail tables of the asynchronous materialized view since the last refresh. If you want to modify a configured refresh interval, you must drop and recreate the asynchronous materialized view.

If you use the `FAST` refresh method, the deferred transactions are saved in a materialized view log. Thus, before you create your asynchronous materialized view, you must create a materialized view log for each detail table included in the asynchronous materialized view that uses FAST refresh. Each detail table can have only one materialized view log even if they are used by more than one asynchronous materialized view with FAST refresh. All columns referenced in an asynchronous materialized view must be included in the corresponding asynchronous materialized view log. If there is more than one asynchronous materialized view with FAST refresh created on a detail table, make sure to include all columns that are used in the different asynchronous materialized views created for that detail table in its asynchronous materialized view log.

The following example creates an asynchronous materialized view that uses FAST refresh, where the deferred transactions are updated every hour after creation. First, create the materialized view log for each detail table, `customer` and `bookOrder`. The following statements create the materialized log views for `customer` and `bookOrder` to track the deferred transactions for the fast refresh. The materialized view log for `customer` tracks the primary key and the customer name as follows:

```
CREATE MATERIALIZED VIEW LOG ON customer WITH PRIMARY KEY (custName);
```

Note: In the CREATE MATERIALIZED VIEW LOG syntax, the primary key is included if you specify WITH PRIMARY KEY or do not mention either PRIMARY KEY or ROWID. All non-primary key columns that you want included in the materialized view log must be specified in the parenthetical column list.

The materialized view log for the `bookorder` table tracks the primary key of `orderId` and columns `custId`, and `book`.

```
CREATE MATERIALIZED VIEW LOG ON bookOrder WITH (custId, book);
```

Once you create the materialized view log for both the `customer` and `bookOrder` detail tables, you can create an asynchronous materialized view. The asynchronous materialized view must include either the ROWID or primary key columns for all the detail tables.

The following example creates an asynchronous materialized view named `SampleAMV` that generates a result set from selected columns in the `customer` and `bookOrder` detail tables. The statement specifies a fast refresh to update the deferred transactions every hour from the moment of creation.

```
CREATE MATERIALIZED VIEW SampleAMV
 REFRESH
     FAST
     NEXT SYSDATE + NUMTODSINTERVAL(1, 'HOUR')
 AS SELECT customer.custId, custName, orderId, book
 FROM customer, bookOrder
 WHERE customer.custId=bookOrder.custId;
```

If you want to manually refresh the materialized view, execute the REFRESH MATERIALIZED VIEW statement. You can manually refresh the materialized view at any time, even if a REFRESH interval is specified. For example, if there were multiple updates to the detail tables, you can manually refresh the `SampleAMV` materialized view as follows:

```
REFRESH MATERIALIZED VIEW SampleAMV;
```

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 "SQL Statements" in the *Oracle TimesTen In-Memory Database SQL Reference*, with the following restrictions:

- All columns in the GROUP BY GroupColumnList must be included in the SelectList.
- SUM and COUNT are allowed, but not expressions involving them, including AVG.
- The following cannot be used in a SELECT statement that is creating a materialized view:
  - DISTINCT
  - FIRST
  - HAVING
  - ORDER BY
  - UNION
  - UNION ALL
  - MINUS
  - INTERSECT
  - JOIN
  - User functions: USER, CURRENT_USER, SESSION_USER

  - Subqueries
  - NEXTVAL and CURRVAL
  - Derived tables and joined tables
- 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.
- Self joins are allowed. A self join is a join of a table to itself. This table appears more than once 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 are not compatible with materialized views from releases earlier than 6.0.

For synchronous materialized views or asynchronous materialized views that use COMPLETE refresh, the following is true for the SELECT statement:

- 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.
- OUTER JOINs are allowed, but the SELECT list 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. The (+) symbol must be used to specify OUTER JOINs of a materialized view.

For asynchronous materialized views that use FAST refresh, the following is true for the SELECT statement:

- Aggregate functions are not supported.
- Outer joins are not supported.
- SELECT list must include ROWID or the Primary Key columns for all the included detail tables.

Dropping a materialized view or a materialized view log

To drop any materialized view, execute the DROP VIEW statement.

The following statement drops the `sampleMV` materialized view.

```
DROP VIEW sampleMV;
```

When there are no asynchronous materialized views referencing a table, the materialized view log on that table can be dropped. For example, if you have dropped the materialized view `sampleAMV`, then the following statements drop the associated materialized view logs.

```
DROP MATERIALIZED VIEW LOG ON customer;
DROP MATERIALIZED VIEW LOG ON bookOrder;
```

The syntax for all SQL statements is provided in the "SQL Statements" chapter in the *Oracle TimesTen In-Memory Database SQL Reference*.

Identifying the table associated with a materialized view log Materialized view logs are represented in the TimesTen system tables as a table named `MVLOG$_nnnnnn`, where `nnnnn` is the table id of the table on which it was created. The table id and table name are both recorded in `SYS.TABLES`. For example, if the materialized view log filename is `MVLOG$_507244`, then you can retrieve the table name from `SYS.TABLES` where the table id is 507244 as follows:

```
select tblname from sys.tables where tblid = 507244;
< T1 >
1 row found.
```

Restrictions on materialized views and 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.

For example, any attempt to update a row in a materialized view generates the following 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.
- No referential indexes can be defined on the materialized view.
- To drop a materialized view must use the DROP VIEW statement.
- You cannot alter a materialized view. 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 are some restrictions to the SQL used to create materialized views. See "CREATE MATERIALIZED VIEW" in the *Oracle TimesTen In-Memory Database SQL Reference* for details.

Performance implications of materialized views

The following sections describes performance implications for each type of materialized view:

- Managing performance for asynchronous materialized views
- Managing performance for synchronous materialized views

Managing performance for asynchronous materialized views For managing performance, you can defer the refresh of the materialized view until an optimal time. Rows in the materialized view logs, detail table and materialized view may be locked during the refresh. If these locks interfere with the user transaction updating the detail tables, then the user can adjust the refresh interval. If performance is the highest priority and the asynchronous materialized view can be out of sync with the detail tables, set the refresh interval to execute when the system load is low.

- FAST refresh incrementally updates the materialized view based on the changes captured in the materialized view log. The time for this refresh depends on the number of modifications captured in the materialized view log and the complexities of the SELECT statement used in the CREATE MATERIALIZED VIEW statement. After every refresh, the processed rows in the materialized view log are deleted.

Update table statistics on the detail table, materialized view log tables and the materialized view at periodic intervals to improve the refresh performance. If the view involves joins, update table statistics before inserting any row in any of the detail tables. Table statistics can be updated using the `ttOptEstimateStats` built-in procedure.

- A COMPLETE refresh is similar to the initial loading of the materialized view at creation time. The time for this refresh depends on the number of rows in the detail tables.

Managing performance for synchronous materialized views The performance of UPDATE and INSERT operations may be impacted if the updated table is referenced in a materialized view. The performance impact depends on many factors, such as the following:

- Nature of the materialized view: How many detail tables, whether outer join or aggregation is used.
- Which indexes are present on the detail table and on the materialized view.
- How many materialized 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 perform "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 performs "view maintenance."

View maintenance needs appropriate indexes just like regular database operations. If they are not there, view maintenance will perform poorly.

All update, insert, or delete statements on detail tables have execution plans, as described in "The TimesTen Query Optimizer" on page 10-1. 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 fast view maintenance, you should evaluate the plans for all the operations that update the detail tables, as follows:

1. Examine all the WHERE clauses for the update or delete statements that frequently occur on the detail tables. Note any clause that uses an index key. For example, if the operations that an application performs 95 percent of the time are as follows:

   ```
   UPDATE T1 set A=A+1 WHERE K1=? AND K2=?
   DELETE FROMT2 WHERE K3=?
   ```

   Then the keys to note are (K1,K2) and K3.

2. Ensure 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 do not have to be unique. The names of the view columns can be different from the names of the table columns, though they are 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 executed in a batch, which has better performance.

The above method may not always work, however. For example, an application may have many different methods to update the detail tables. The application would have to select far too many items in the view or create too many indexes on the view, taking up more space or more performance than you might wish. An alternative method is as follows:

1. For each table in the view's FROM clause (each detail table), check which ones are frequently changed by UPDATE, INSERT and CREATE VIEW statements. For example, a view's FROM clause may have tables T1, T2, T3, T4, and 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, which speeds up the maintenance. It is generally not as fast as the first method, but it is still good.

Understanding indexes

Indexes are auxiliary data structures that greatly improve the performance of 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 "The TimesTen Query Optimizer" on page 10-1.

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 SQL Reference* for more information on NULL values.

To perform any operation that creates, drops or alters an index, the user must have the appropriate privileges, which are described along with the syntax for all SQL statements in the "SQL Statements" chapter in the *Oracle TimesTen In-Memory Database SQL Reference*.

The following sections describe how to manage your index:

- Overview of index types
- Creating an index
- Altering an index
- Dropping an index
- Estimating index size

Overview of index types

TimesTen provides three types of indexes to enable fast access to tables.

- **Hash Indexes**. Hash indexes are useful for finding rows with an exact match on one or more columns. Hash indexes are useful for doing equality searches. TimesTen currently supports a maximum of one hash index per table. A hash index is created with the UNIQUE HASH option, which is specified over the columns that make up the primary key of a table.

  The "CREATE TABLE" section of the *Oracle TimesTen In-Memory Database SQL Reference* discusses in detail the automatic creation of hash indexes. For an example of how to create a hash index when creating the table, see Example 7–2.

- **Range Indexes**. Range indexes are useful for finding rows with column values within a certain range. You can create range indexes over one or more columns of a table. Up to 32 range indexes may be created on one table.

  Range indexes and equijoins can be used for equality and range searches, such as greater than or equal to, less than or equal to, and so on. If you have a primary key on a field and want to see if `FIELD > 10`, then the primary key index will not expedite finding the answer, but a separate index will.

  The "CREATE INDEX" section of the *Oracle TimesTen In-Memory Database SQL Reference* discusses in describes how to create range indexes.

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

- **Bitmap Indexes**. Bitmap indexes are useful when searching and retrieving data from columns with low cardinality. That is, these columns can have only a few unique possible values. Bitmap indexes encode information about a unique value in a row in a bitmap. Each bit in the bitmap corresponds to a row in the table. Use a bitmap index for columns that do not have many unique values. An example of such a column is a column that records gender as one of two values.

  Bitmap indexes increase the performance of complex queries that specify multiple predicates on multiple columns connected by AND and OR operators.

  See "CREATE INDEX" in the *Oracle TimesTen In-Memory Database SQL Reference* for how to create and more information on bitmap indexes.

  Note: Alternatively, you can perform lookups by RowID for fast access to data. See the *Oracle TimesTen In-Memory Database SQL Reference* for more information on RowIDs.

Creating an index

To create an index, execute 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 7–4 Create an index

Create an index `IxID` over column `CustID` of table `NameID`.

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

You can also create a hash index by creating a primary key or using the UNIQUE HASH ON clause in the CREATE TABLE. However, TimesTen may create temporary hash and range indexes automatically during query processing to speed up query execution.

Altering an index

You can change a range index to a hash index with the USE HASH INDEX of the ALTER TABLE statement.

Dropping 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`.

Example 7–5 Drop an index

The following drops the index named `IxID`.

```
DROP INDEX IxID;
```

To drop a TimesTen index, execute the DROP INDEX SQL statement. All indexes in a table are dropped automatically when the table is dropped.

Estimating index size

Increasing the size of a TimesTen data store can be done on first connect. 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`.

"Data Types" in the *Oracle TimesTen In-Memory Database SQL Reference* contains a detailed description of these data types.

To perform any operation for inserting or deleting rows, the user must have the appropriate privileges, which are described along with the syntax for all SQL statements in the "SQL Statements" chapter in the *Oracle TimesTen In-Memory Database SQL Reference*.

The following sections describe how to manage your rows:

- Inserting rows
- Deleting rows

Inserting rows

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

Example 7–6 Insert a row in a table

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. Create Indexes after the bulk load is completed.

Deleting rows

To delete a row, execute the DELETE statement.

Example 7–7 Delete a row

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%';
```

8

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 "Data Store Attributes" in *Oracle TimesTen In-Memory Database Reference* for details on how to set isolation levels and durability options.

The main topics in this chapter are:

- TimesTen commit behavior
- Transaction semantics
- Transaction atomicity and durability
- Controlling durability and logging
- Concurrency control
- Checkpoints

TimesTen commit behavior

TimesTen closes cursors on commit and rollback. The Oracle Database supports cursors that stay open across transaction boundaries.

In TimesTen releases before 11.2.1, DDL statements were executed as part of the current transaction and were committed or rolled back along with the rest of the transaction. DDL statements include the following:

- CREATE, ALTER and DROP statements for any database object, including tables, views, users, procedures and indexes.
- TRUNCATE
- GRANT and REVOKE

In contrast, the Oracle Database issues an implicit commit before and after any DDL statement.

Beginning with release 11.2.1, TimesTen issues an implicit commit before and after any DDL statement by default. This behavior is controlled by the DDLCommitBehavior attribute, which is set to 0 (Oracle behavior) by default. See "DDLCommitBehavior" in the *Oracle TimesTen In-Memory Database Reference*.

Consequences of TimesTen default commit behavior beginning with release 11.2.1 include:

- DDL changes cannot be rolled back. Each DDL statement is committed individually.
- DDL statements delete records from global temporary tables unless the tables were created with the ON COMMIT PRESERVE ROWS clause.
- TRUNCATE statements are committed. Parent and child tables must be truncated in different transactions.
- Query results that are obtained from the CREATE TABLE ... AS SELECT statement are visible immediately.

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 or writes, depending on the transaction isolation level. See "Concurrency control" on page 8-6.
- **Logging.** Modifications to the data store are recorded at user-specified levels in a log. See "Transaction atomicity and durability" on page 8-3.
- **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. ■ Newly modified values of data are made available for other transactions to read and to modify. ■ 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. ■ Locks that were acquired on behalf of the transaction are released.
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 8-8.
Application fails	■ Transaction is rolled back.

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.

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 DELETE 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 checkpointing. See "Checkpoints" on page 8-8 and "Log files" on page 8-6 for more information.

Because transaction support adds overhead to execution time, TimesTen allows applications to choose from the following options:

- Guaranteed atomicity and durability by setting attribute DurableCommits=1.
- Guaranteed atomicity, delayed durability by setting attribute DurableCommits=0.

The following table summarizes the guarantees and limitations of the atomicity and durability options:

Attribute Setting	Non-blocking checkpoints possible?	Row-level locking possible?	Transaction rollback possible?	Recovery procedure	Committed transactions vulnerable to loss
Durable Commits = 1	Yes	Yes	Yes	Read most recent checkpoint image. Apply log.	None

Attribute Setting	Non-blocking checkpoints possible?	Row-level locking possible?	Transaction rollback possible?	Recovery procedure	Committed transactions vulnerable to loss
Durable Commits = 0	Yes	Yes	Yes	Read most recent checkpoint image. Apply log.	Transactions that committed after the last checkpoint or durable commit

The following sections describe these options in greater detail:

- Guaranteed atomicity and durability
- Guaranteed atomicity, delayed durability

Guaranteed atomicity and durability

When you set the attribute DurableCommits=1, then durable commits are implemented and the log is written to disk at transaction commit. 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 8-8 and "Log files" on page 8-6. 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 DurableCommit attributes to 1.

Guaranteed atomicity, delayed durability

When you set the attribute DurableCommits=0, then durable commits are not implemented and the log is not written to disk at transaction commit. It is possible to connect to a data store 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-durable committed transactions have much better response time than durable-committed transactions, because no I/O is required to commit the transaction. A non-durable committed transaction can be made durable either by checkpointing or by committing a subsequent transaction durably. See "Checkpoints" on page 8-8 for more information. As with guaranteed durability mode, a checkpoint makes all transactions that committed before the checkpoint durable. Committing a transaction durably commits that transaction and makes all previously committed transactions durable.

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 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.

Controlling durability and logging

Applications can control whether a transaction is durable; however, log files are always 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 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 always go to disk. Thus, the Logging attribute is always set to "1".

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 quick 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 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 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 *Oracle TimesTen In-Memory Database TimesTen to TimesTen Replication Guide*.

Log files

All log files are created in the same directory as the data store files unless the `LogDir` attribute specifies a different location. 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.

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`*`.log`*`n`*`.arch`. In this case, the application is responsible for removing these unneeded log files. See "LogPurge" in the *Oracle TimesTen In-Memory Database Reference* for details on purging log files.

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 DELETE 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, the SELECT portion of INSERT SELECT statements and MERGE 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.

Using read committed isolation level can lead to duplicates in a result set. A SELECT statement selects more or less rows than the total number of rows in the table if some rows were added or removed and committed in the range that the SELECT scan is happening. This may happen when an UPDATE, INSERT or DELETE adds or deletes a value from an index and the SELECT scan is using this index. This can also happen when an INSERT or DELETE adds or deletes rows from the table and the SELECT operation is using an all-table scan.

This happens because index values are ordered and an UPDATE of an index value may delete the old value and insert the new value into a different place. In other words it moves a row from one position in the index to another position. If an index scan sees the same row in both positions, it returns the same row twice. This does not happen with a serial scan because table pages are unordered and rows do not need to be moved around for an UPDATE. Hence once a scan passes a row, it will not see that same row again.

The only general way to avoid this problem is for the SELECT to use serializable isolation. This prevents a concurrent INSERT, DELETE or UPDATE operation. There is no reliable way to avoid this problem with INSERT or DELETE by forcing the use of an index because these operations affect all indexes. With UPDATE, this problem can be avoided by forcing the SELECT statement to use an index that is not being updated.

All data access in TimesTen uses locking or copying to provide isolation. 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 locks, table-level locks and data store-level locks:

- 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. To use row-level locking, applications must set the LockLevel connection attribute to 0 (the default value) or call the `ttLockLevel` built-in procedure with the *`lockLevel`* parameter set to `Row`.

In order to CREATE, DROP, or ALTER a user, you can only use row-level locking and thus, the Locklevel must be set to 0 before you can perform any of these operations.

- Table-level locking is useful for queries that access a significant portion of the rows of a table or when there are very few concurrent transactions that access a table. Enable the optimizer to choose table-level locking by calling the `ttOptSetFlag` built-in procedure and setting the `TbleLock` flag to 1. If both table-level and row-level locking are disabled, TimesTen chooses row-level locking. If both table-level and row-level locking are enabled, TimesTen chooses the locking scheme that is more likely to have better performance. Typically, table-level locking provides better performance than row-level locking because of reduced locking overhead. For more information, see "ttOptSetFlag" in *Oracle TimesTen In-Memory Database Reference*.
- 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 applications that never need to execute multiple concurrent transactions. With data store-level locking, every transaction effectively runs in ANSI Serializable isolation, since concurrent transactions are disallowed. To use data store-level locking, applications set the LockLevel connection attribute to 1 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" on page 9-7.

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 8-10 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, then recovery restarts, using the other checkpoint file. For example, if a system failure occurred while that checkpoint was begin written.

TimesTen also creates *dsName*.res*n* files for each data store. These files are pre-allocated space to the same size as a log file used by TimesTen for maintaining log files. The .res files allow the data store to remain operational when the log directory is full. If the log directory becomes full, an error is recorded and new transactions are prevented. Existing operations that cannot be recorded in a new log file are written to the .res files. When space becomes available in the log directory, the information in the .res files is copied to the log files and transactions can begin again.

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 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 and Fuzzy or non-blocking 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 uses the log files 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 is 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 is 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. To request a fuzzy checkpoint, an application uses the `ttCkpt` built-in procedure. This procedure issues a fuzzy checkpoint. As with all blocking checkpoints, the actual checkpoint is delayed until the requesting transaction commits or rolls back.

Setting and managing checkpoints

By default, TimesTen performs automatic non-blocking checkpoints in the background. In this case, background checkpoints are non-blocking. See "Fuzzy or non-blocking checkpoints" on page 8-9 for more information.

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, checkpoints are still taken to purge the log files. See "Transaction-consistent checkpoints" on page 8-9.

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 Guide* or "Calling TimesTen built-in procedures" in the *Oracle TimesTen In-Memory Database Java Developer's 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 Reference*.

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 Reference* 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* and *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.

9 TimesTen Database 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 to detect problems.
- How large is the potential performance impact.
- Where are the performance gains.
- What are the tradeoffs.

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
- Materialized view tuning
- Transaction tuning
- Recovery tuning
- Scaling for multiple CPUs
- XLA tuning

For information on tuning TimesTen Java applications, see "Application Tuning" in the *Oracle TimesTen In-Memory Database Java Developer's Guide*.

For information on tuning TimesTen C applications, see "Application Tuning" in the *Oracle TimesTen In-Memory Database C Developer's Guide*.

System and data store tuning

The following sections include tips for tuning your system and data stores:

- Provide enough memory

- Size your data store correctly
- Calculate shared memory size for PL/SQL runtime
- Increase LogBufMB if needed
- Use temporary data stores if appropriate
- Avoid connection overhead
- Load the data store into RAM when duplicating
- Reduce contention
- Avoid OS paging at load time
- Consider special options for maintenance
- Check your driver
- Enable tracing only as needed
- Investigate alternative JVMs
- If you are using replication, adjust log buffer size and CPU
- Increase replication throughput for active standby pairs
- Migrating data with character set conversions

Provide enough memory

Performance impact: Large

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.

Size your data store correctly

Performance impact: Variable

When you create a data store, you are required to specify a data store size. Specifically, you specify sizes for the permanent and temporary partitions of the data store. For details on how to size the data store and shared memory, see "Specifying the size of a data store" on page 1-17.

Calculate shared memory size for PL/SQL runtime

Performance impact: Variable

The PL/SQL runtime system uses an area of shared memory to hold metadata about PL/SQL objects in TimesTen and the executable code for PL/SQL program units that are currently being executed or that have recently been executed. The size of this shared memory area is controlled by the PLSQL_MEMORY_SIZE first connect attribute.

When a new PL/SQL program unit is prepared for execution, it is loaded into shared memory. If shared memory space is not available, the cached recently-executed program units are discarded from memory until sufficient shared memory space is available. If all of the PL/SQL shared memory is being used by currently executing program units, then attempts by a new connection to execute PL/SQL may result in out of space errors, such as ORA-04031. If this happens, increase the PLSQL_MEMORY_SIZE.

Even if such out of space errors do not occur, the PLSQL_MEMORY_SIZE may be too small. It is less expensive in CPU time to execute a PL/SQL procedure that is cached in shared memory than one that is not cached. In a production application, the goal should be for PLSQL_MEMORY_SIZE to be large enough so that frequently-executed PL/SQL units are always cached. The TimesTen built-in procedure `ttPLSQLMemoryStats` can be used to determine how often this occurs. The `PinHitRatio` value returned is a real number between 0 and 1.

- 1.0: A value of 1.0 means that every PL/SQL execution occurred from the cache.
- 0.0: A value of 0.0 means that every execution required that the program unit be loaded into shared memory.

The proper value of PLSQL_MEMORY_SIZE for a given application depends on the application. If only a small number of PL/SQL program units are repeatedly executed, then the size requirements can be small. If the application uses hundreds of PL/SQL program units, memory requirements increase.

Performance increases dramatically as the `PinHitRatio` goes up. In one set of experiments, an application program repeatedly executed a large number of PL/SQL stored procedures. With a larger value for PLSQL_MEMORY_SIZE, the application results in a `PinHitRatio` of around 90%, and the average execution time for a PL/SQL procedure was 0.02 seconds. With a smaller value for PLSQL_MEMORY_SIZE, there was more contention for the cache, resulting in a `PinHitRatio` of 66%. In this experiment the average execution time was 0.26 seconds.

The default value for PLSQL_MEMORY_SIZE is 32 MBytes. This should be sufficient for several hundred PL/SQL program units of reasonable complexity to execute. After running a production workload for some time, check the value of `PinHitRatio`. If it is less than 0.90, consider increasing PLSQL_MEMORY_SIZE.

Increase LogBufMB if needed

Performance impact: Large

Increasing the value of LogBufMB can have a substantial positive performance impact. If LOG_BUFFER_WAITS is increasing, then increase the value of LogBufMB.

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 LogBufMB value does not improve performance.

Use temporary data stores if appropriate

Performance impact: Variable

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. The amount of data written to the log for temporary data stores is less than that written for permanent data stores, allowing better performance for temporary data stores. 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.

Avoid connection overhead

Performance impact: Large

By default, TimesTen loads an idle data store, which is 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 to and unloaded 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.

Load the data store into RAM when duplicating

Performance impact: Large

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" on page 9-4.

Reduce contention

Data store contention can substantially impede application performance.

To reduce contention in your application:

- Choose the appropriate locking method. See "Choose the best method of locking" on page 9-7.
- Distribute data strategically in multiple tables or data stores.

If your application suffers a decrease in performance because of lock contention and a lack of concurrency, reducing contention is an important first step in improving performance.

The LOCK_GRANTS_IMMED, LOCK_GRANTS_WAIT and LOCK_DENIALS_COND columns in the SYS.MONITOR table provide some information on lock contention:

- LOCK_GRANTS_IMMED counts how often a lock was available and was immediately granted at lock request time.

- LOCK_GRANTS_WAIT counts how often a lock request was granted after the requestor had to wait for the lock to become available.
- LOCK_DENIALS_COND counts how often a lock request was not granted because the requestor did not want to wait for the lock to become available.

If limited concurrency results in a lack of throughput, or if response time is an issue, an application can serialize JDBC calls to avoid contention. This can be achieved by having a single thread issue all those calls. Using a single thread requires some queuing and scheduling on the part of the application, which has to trade off some CPU time for a decrease in contention and wait time. The result is higher performance for low-concurrency applications that spend the bulk of their time in the data store.

Avoid OS paging at load time

Performance impact: Medium

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.

Consider special options for maintenance

Performance impact: Medium

During special operations such as initial loading, you can choose different options than you would use during normal operations. In particular, consider 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.

An alternative to data store-level locking is to exploit concurrency. Multiple copies of `ttBulkCp -i` can be run using the `-notblLock` option. Optimal batching for `ttBulkCp` occurs by adding the `-xp 256` option. `ttMigrate` can be run with `-numThreads` option to load individual or multiple tables concurrently.

Check your driver

Performance impact: Large

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 1-7 and "Specify the ODBC driver" on page 1-11 for a description of the TimesTen Data Manager drivers for the different platforms.

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.

Enable tracing only as needed

Performance impact: Large

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.

Investigate alternative JVMs

Performance impact: Variable

JRockit, IBM and HP provide JVMs that may perform better than the Sun JVM.

If you are using replication, adjust log buffer size and CPU

Performance impact: Large

If you are planning a replication scheme, then ensure the following:

- The setting for LogBufMB 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.
- If the sending side and receiving side of the replication scheme are mismatched in CPU power, place the replication receiver on the faster system.

Increase replication throughput for active standby pairs

Performance impact: Medium

Use the `RecoveryThreads` first connection attribute to increase the number of threads that apply changes from the active master data store to the standby master data store from 1 to 2. If you set `RecoveryThreads` to 2 on the standby, you should also set it to 2 on the active to maintain increased throughput if there is a failover.

You can also set `RecoveryThreads` to 2 on one or more read-only subscribers in an active standby pair to increase replication throughput from the standby master data store.

Data stores must be hosted on systems that are 2-way or larger to take advantage of setting this attribute to 2.

Migrating data with character set conversions

Performance impact: Variable

If character set conversion is requested when migrating data stores, performance may be slower than if character set conversion is not requested.

Client/Server tuning

The following sections include tips for Client/Server tuning:

- Work locally when possible
- Choose a timeout interval
- Choose the best method of locking
- Use shared memory segment as IPC when client and server are on the same machine
- Enable TT_PREFETCH_CLOSE for serializable transactions
- Use a connection handle when calling SQLTransact

Work locally when possible

Performance impact: Large

Using TimesTen Client to access data stores on a remote server machine adds 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.

Choose a timeout interval

By default, connections wait 10 seconds to acquire a lock. To change the timeout interval for locks, use the `ttLockWait` built-in procedure.

Choose the best method of locking

When multiple connections access a data store simultaneously, TimesTen uses locks to ensure that the various transactions operate in apparent isolation. TimesTen supports the isolation levels described in Chapter 8, "Transaction Management and Recovery". It also supports the locking levels: data store-level locking, table-level locking and row-level locking. You can use the `LockLevel` connection attribute to indicate whether data store-level locking or row-level locking should be used. Use the `ttOptSetFlag` procedure to set optimizer hints that indicate whether table locks should be used. The default lock granularity is row locks.

Choose an appropriate lock level

If there is very little contention on the data store, use table-level locking. It provides better performance and deadlocks are less likely. There is generally little contention on the data store when transactions are short or there are few connections. In that case, transactions are not likely to overlap.

Table-level locking is also useful when a statement accesses nearly all the rows on a table. Such statements can be queries, updates, deletes or multiple inserts done in a single transaction.

TimesTen uses table locks only with serializable isolation. If your application specifies table locks with any other isolation levels, TimesTen overrides table-level locking and uses row locks. However, the optimizer plan may still display table-level locking hints.

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

Row-level locking is generally preferable when there are many concurrent transactions that are not likely to need access to the same row.

Choose an appropriate isolation level

When using row-level locking, applications can run transactions at the SERIALIZABLE or READ_COMMITTED isolation level. The default isolation level is READ_COMMITTED. You can use the `Isolation` connection attribute to specify one of these isolation levels for new connections.

When running at SERIALIZABLE transaction isolation level, TimesTen holds all locks for the duration of the transaction, so:

- Any transaction updating a row blocks writers until the transaction commits.
- Any transaction reading a row blocks out writers until the transaction commits.

When running at READ_COMMITTED transaction isolation level, TimesTen only holds update locks for the duration of the transaction, so:

- Any transaction updating a row blocks out readers and writers of that row until the transaction commits.
- Phantoms are possible. A phantom is a row that appears during one read but not during another read, or appears in modified form in two different reads, in the same transaction, due to early release of read locks during the transaction.

You can determine if there is an undue amount of contention on your system by checking for time-out and deadlock errors (errors # 6001, 6002, and 6003). Information is also available in the LOCK_TIMEOUTS and DEADLOCKS columns of the SYS.MONITOR table.

Use shared memory segment as IPC when client and server are on the same machine

Performance impact: Variable

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 Server daemon options" on page 3-8.

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 2-13.

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 2-13. 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 TT_PREFETCH_CLOSE for serializable transactions

Performance impact: Variable

Enable TT_PREFETCH_CLOSE for serializable transactions in client/server applications. In serializable Isolation mode, all transactions should be committed when executed, including read-only transactions. When TT_PREFETCH_CLOSE is enabled, the server closes the cursor and commits the transaction after the server has fetched the entire result set for a read-only query. The client should still call `SQLFreeStmt(SQL_CLOSE)` and `SQLTransact(SQL_COMMIT)`, but the calls are executed in the client and do not require a network round trip between the client and server. TT_PREFETCH_CLOSE enhances performance by decreasing the network traffic between client and server.

Do not use multiple statement handles when TT_PREFETCH_CLOSE is enabled. The server may fetch all of the result set, commit the transaction, and close the statement handle before the client is finished, resulting in the closing of all statement handles.

The following examples show how to use the TT_PREFETCH_CLOSE option with ODBC and JDBC.

This example sets TT_PREFETCH_CLOSE with the `SQLSetConnectOption` ODBC function. You can also set it with the `SQLSetStmtOption` ODBC function.

```
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);
```

This example shows how to enable the TT_PREFETCH_CLOSE option with JDBC:

```
con = DriverManager.getConnection ("jdbc:timesten:client:" + DSN);
```

```
stmt = con.createStatement();
import com.timesten.sql
...
...
con.setTtPrefetchClose(true);
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();
```

Use a connection handle when calling SQLTransact

Performance impact: Large

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).
```

If the intention of the application is simply to commit or rollback on a single connection, it should use a valid connection handle when calling `SQLTransact`.

SQL tuning

After you have determined the overall locking and I/O strategies, make sure that the individual SQL statements are executed as efficiently as possible. The following sections describe how to streamline your SQL statements:

- Tune statements and use indexes
- Select hash, range, or bitmap indexes appropriately
- Use foreign key constraint appropriately
- Computing exact or estimated statistics
- Avoid ALTER TABLE
- Avoid nested queries
- Prepare statements in advance
- Avoid unnecessary prepare operations

Tune statements and use indexes

Performance impact: Large

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 10, "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 is not being used to fetch the row, the application should set SQL_MAX_ROW_COUNT to 1. See the *Oracle TimesTen In-Memory Database Reference*.

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.

If you have implemented time-based aging for a table or cache group, create an index on the timestamp column for better performance of aging. See "Time-based aging" on page 7-8.

Select hash, range, or bitmap indexes appropriately

Performance impact: Variable

The TimesTen Data Manager supports hash, range, and bitmap indexes. Each index structure has a different strength.

Hash indexes are created when you supply the UNIQUE HASH clause for the CREATE TABLE or ALTER TABLE statements. Hash indexes require that the table have a primary key.

Range indexes are created by default with the CREATE TABLE statement or created with the CREATE INDEX statement. Range indexes can speed up exact key lookups but are more flexible and can speed up other queries as well. Select a range index if your queries include LESS THAN or GREATER THAN comparisons. Range indexes are effective for high-cardinality data: that is, data with many possible values, such as CUSTOMER_NAME or PHONE_NUMBER.

Range 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;
```

Range indexes have a dynamic structure that adjusts itself automatically to accommodate changes in table size. A range 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 range index is likely to be more compact than an equivalent hash index.

Bitmap indexes are created with the CREATE INDEX statement. Bitmap indexes are performant when searching and retrieving data from columns with low cardinality. Bitmap indexes are useful with equality queries, especially when using the AND and OR operators. These indexes increase the performance of complex queries that specify multiple predicates on multiple columns connected by AND and OR operators. Bitmap indexes are widely used in data warehousing environments. The environments typically have large amounts of data and ad hoc queries, but a low level of concurrent DML transactions. Bitmap indexes are compressed and have smaller storage requirements than other indexing techniques. For more details on when to use bitmap indexes, see "CREATE INDEX" in the *Oracle TimesTen In-Memory Database SQL Reference*.

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 range 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: Large

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. For performance reasons, TimesTen does not hold a lock on tables or rows when computing statistics.

If you have examined the plans generated for your statements and you think they may not be optimal, consider computing statistics before preparing your statements and re-examining the plans. See Chapter 10, "The TimesTen Query Optimizer" for more information.

If you have not 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`.

- The `ttOptUpdateStats` built-in procedure evaluates every row of the table(s) in question and computes exact statistics.
- The `ttOptEstimateStats` procedure evaluates only a sampling of the rows of the table(s) in question and produces estimated statistics.

Estimating statistics can be faster, although it may result in less accurate statistics. In general, if time is not an issue, it is best to call `ttOptUpdateStats`. Estimation is preferable if overall application performance may be affected. 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, you should compute statistics once after loading your data store but before preparing commands, and then periodically only if the composition of your data changes substantially. It is recommended to always update statistics after loading the data store and after a large number of inserts or deletes have occurred.

Avoid ALTER TABLE

Performance impact: Variable

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.

Avoid nested queries

Performance impact: Variable

TimesTen supports nested queries with some limitations. However, performance varies and is generally not optimal. See the *Oracle TimesTen In-Memory Database SQL Reference* for details on subqueries.

Prepare statements in advance

If you have applications that generate a statement multiple times searching for different values each time, prepare a parameterized statement to reduce compile time. For example, if your application generates statements like:

```
SELECT A FROM B WHERE C = 10
SELECT A FROM B WHERE C = 15
```

You can replace these statements with the single statement:

```
SELECT A FROM B WHERE C = ?
```

TimesTen shares prepared statements automatically after they have been committed. As a result, an application request to prepare a statement for execution may be completed very quickly if a prepared version of the statement already exists in the system. Also, repeated requests to execute the same statement can avoid the prepare overhead by sharing a previously prepared version of the statement.

Even though TimesTen allows prepared statements to be shared, it is still a good practice for performance reasons to use parameterized statements. Using parameterized statements can further reduce prepare overhead, in addition to any savings from sharing statements.

Avoid unnecessary prepare operations

Because preparing SQL statements is an expensive operation, your application should minimize the number of calls to the prepare API. Most applications prepare a set of statements at the beginning of a connection and use that set for the duration of the connection. This is a good strategy when connections are long, consisting of hundreds or thousands of transactions. But if connections are relatively short, a better strategy is to establish a long-duration connection that prepares the statements and executes them on behalf of all threads or processes. The trade-off here is between communication overhead and prepare overhead, and can be examined for each application. Prepared statements are invalidated when a connection is closed.

See "`ttSQLCmdCacheInfoGet`" in the *Oracle TimesTen In-Memory Database Reference* for related information.

Materialized view tuning

The following sections include tips for improving performance of materialized views:

- Limit number of join rows
- Use indexes on join columns
- Avoid unnecessary updates
- Avoid changes to the inner table of an outer join
- Limit number of columns in a view table

Limit number of join rows

Performance impact: Variable

Larger numbers of join rows decrease performance. You can limit the number of join rows and the number of tables joined 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.

Use indexes on join columns

Performance impact: Variable

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.

Avoid unnecessary updates

Performance impact: Variable

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.

Avoid changes to the inner table of an outer join

Performance impact: Variable

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.

Limit number of columns in a view table

Performance impact: Variable

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 range indexes and temporary hash scans.

Transaction tuning

The following sections describe how to increase performance when using transactions:

- Size transactions appropriately
- Use durable commits appropriately
- Avoid frequent checkpoints
- Turn off autocommit mode
- Avoid transaction rollback

Size transactions appropriately

Each transaction, when it generates transaction log records (for example, a transaction that does an INSERT, DELETE or UPDATE), incurs a disk write when the transaction commits. Disk I/O affects response time and may affect throughput, depending on how effective group commit is.

Performance-sensitive applications should avoid unnecessary disk writes at commit. Use a performance analysis tool to measure the amount of time your application spends in disk writes (versus CPU time). If there seems to be an excessive amount of I/O, there are two steps you can take to avoid writes at commit:

- Adjust the transaction size.
- Adjust whether disk writes are performed at transaction commit. See "Use durable commits appropriately".

Long transactions perform fewer disk writes per unit of time than short transactions. However, long transactions also can reduce concurrency, as discussed in Chapter 8, "Transaction Management and Recovery".

- If only one connection is active on a data store (for example, if it is an exclusive connection), longer transactions could improve performance. However, long transactions may have some disadvantages, such as longer rollbacks.
- If there are multiple connections, there is a trade-off between transaction log I/O delays and locking delays. In this case transactions are best kept to their natural length, as determined by their requirements for atomicity and durability.

Use durable commits appropriately

By default, each TimesTen transaction results in a disk write at commit time. This practice ensures that no committed transactions are lost because of system or application failures. Applications can avoid some or all of these disk writes by performing nondurable commits. Nondurable commits do everything that a durable commit does except write the transaction log to disk. Locks are released and cursors are closed, but no disk write is performed.

Note: Some controllers or drivers only write data into cache memory in the controller or write to disk some time after the operating system is told that the write is completed. In these cases, a power failure may cause some information that you thought was durably committed to be lost. To avoid this loss of data, configure your disk to write to the recording media before reporting media before reporting completion or use an uninterruptible power supply.

The advantage of nondurable commits is a potential reduction in response time and increase in throughput. The disadvantage is that some transactions may be lost in the event of system failure. An application can force the transaction log to disk by performing an occasional durable commit or checkpoint, thereby decreasing the amount of potentially lost data. In addition, TimesTen itself periodically flushes the transaction log to disk when internal buffers fill up, limiting the amount of data that will be lost.

Transactions can be made durable or can be made to have delayed durability on a connection-by-connection basis. Applications can force a durable commit of a specific transaction by calling the `ttDurableCommit` procedure.

Applications that do not use nondurable commits can benefit from using synchronous writes in place of write and flush. To turn on synchronous writes set the first connection attribute `LogFlushMethod=2`.

The XACT_D_COMMITS column of the SYS.MONITOR table indicates the number of transactions that were durably committed.

Avoid frequent checkpoints

Applications that are connected to a data store for a long period of time occasionally need to call the `ttCkpt` built-in procedure to checkpoint the data store so that transaction log files do not fill up the disk. Transaction-consistent checkpoints can have a significant performance impact because they require exclusive access to the data store.

It is generally better to call `ttCkpt` to perform a non-blocking (or "fuzzy") checkpoint than to call `ttCkptBlocking` to perform a blocking checkpoint. Non-blocking checkpoints may take longer, but they permit other transactions to operate against the data store at the same time and thus impose less overall overhead. You can increase the interval between successive checkpoints by increasing the amount of disk space available for accumulating transaction log files.

As the transaction log increases in size (if the interval between checkpoints is large), recovery time increases accordingly. If reducing recovery time after a system crash or application failure is important, frequent checkpoints may be preferable. The DS_CHECKPOINTS column of the SYS.MONITOR table indicates how often checkpoints have successfully completed.

Turn off autocommit mode

AUTOCOMMIT mode forces a commit after each statement, and is enabled by default. Committing each statement after execution, however, can significantly degrade performance. For this reason, it is generally advisable to disable AUTOCOMMIT, using the appropriate API for your programming environment.

The XACT_COMMITS column of the SYS.MONITOR table indicates the number of transaction commits.

Note: If you do not include any explicit commits in your application, the application can use up important resources unnecessarily, including memory and locks. All applications should do periodic commits.

Avoid transaction rollback

When transactions fail due to erroneous data or application failure, they are rolled back by TimesTen automatically. In addition, applications often explicitly rollback transactions to recover from deadlock or timeout conditions. This is not desirable from a performance point of view, as a rollback consumes resources and the entire transaction is wasted.

Applications should avoid unnecessary rollbacks. This may mean designing the application to avoid contention and checking application or input data for potential errors in advance, if possible. The XACT_ROLLBACKS column of the SYS.MONITOR table indicates the number of transactions that were rolled back.

Recovery tuning

The following sections include tips for improving performance of data store recovery after data store shutdown or system failure:

- Set RecoveryThreads
- Discovered direct I/O on HP-UX

Set RecoveryThreads

Performance impact: Large

Set the RecoveryThreads attribute to the number of indexes or CPUs to improve recovery performance.

Discovered direct I/O on HP-UX

Performance impact: Medium

Setting `discovered_direct_iosz` for the Veritas file system on HP-UX improves recovery performance. Writes that are larger than `discovered_direct_iosz` bypass the file system buffer cache and go directly to disk. Set `discovered_direct_iosz` to at least one megabyte. See Veritas documentation for more information.

Scaling for multiple CPUs

The following sections include tips for improving performance for multiple CPUs:

- Run the demo applications as a prototype
- Limit database-intensive connections per CPU
- Use read operations when available
- Limit prepares, re-prepares and connects
- Limit replication transmitters and receivers and XLA readers
- Allow indexes to be rebuilt in parallel during recovery

- Use private commands

Run the demo applications as a prototype

Performance impact: Variable

One way to determine the approximate scaling you can expect from TimesTen is to run one of the scalable demo applications, such as `tptbm`, on your system.

The `tptbm` application implements a multi-user throughput benchmark. It allows you to control how it executes, including options to vary the number of processes that execute TimesTen operations and the transaction mix of SELECTs, UPDATEs, and INSERTs, for example. Run `tptbm -help` to see the full list of options.

By default the demo executes one operation per transaction. You can specify more operations per 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 comment out the TimesTen calls and find they still have bad scaling due to issues in their application.

You may also find, for example, that some simulated application data is not being generated properly, so that 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.

See the Quick Start home page at *`install_dir`*`/quickstart.html` for additional information about `tptbm` and other demo applications. Go to the ODBC link under "Sample Programs".

Limit database-intensive connections per CPU

Performance impact: Variable

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.

Use read operations when available

Performance impact: Variable

Read operations scale better than write operations. Make sure that the read and write balance reflects the real-life workload of your application.

Limit prepares, re-prepares and connects

Performance impact: Variable

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.

Limit replication transmitters and receivers and XLA readers

Performance impact: Variable

Replication and XLA operations have significant logging overhead. Replication scales best when there are a limited number of transmitters or receivers. Check your replication topology and see if you can simplify it. Generally, 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.

Allow indexes to be rebuilt in parallel during recovery

Performance impact: Variable

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 user 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.

Use private commands

Performance impact: Variable

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 tuning

The following sections include tips for improving XLA performance:

- Increase transaction log buffer size when using XLA
- Prefetch multiple update records
- Acknowledge XLA updates

Increase transaction log buffer size when using XLA

A larger transaction log buffer size is appropriate when using XLA. When XLA is enabled, additional transaction log records are generated to store additional information for XLA. To ensure the transaction log buffer is properly sized, one can watch for changes in the SYS.MONITOR table entries LOG_FS_READS and LOG_BUFFER_WAITS. For optimal performance, both of these values should remain 0. Increasing the transaction log buffer size may be necessary to ensure the values remain 0.

Prefetch multiple update records

Performance impact: Medium

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.

Acknowledge XLA updates

Performance impact: Medium

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.

10 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 MATERIALIZED VIEW statement is prepared through an ODBC `SQLPrepare` or `SQLExecDirect`function or any of the JDBC execute methods. 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:

Action	**Description**
`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.
`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:

Action	**Description**
`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.
. . .	. . .
`ttOptUpdateStats` on T1	Command is invalidated if the invalidate flag is passed to the `ttOptUpdateStats` procedure.

Action	Description
. . .	. . .
`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, these hints persist until the command is deleted. See "Modifying plan generation" on page 10-11" for more information. 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 SQL commands stored in the SQL Command Cache

All commands executed—SQL statements, built-in procedures, and so on—are stored in the SQL Command Cache, which uses temporary memory. The commands are stored up until the limit of the SQL Command Cache is reached, then the new commands are stored after the last used commands are removed. You can retrieve one or more of these commands that are stored in the SQL Command Cache.

Note: This section describes viewing the commands stored in the SQL Command Cache. For details on how to view the query plans associated with these commands, see "Viewing query plans associated with commands stored in the SQL Command Cache" on page 10-8.

The following sections describe how to view commands cached in the SQL Command Cache:

- Managing performance and troubleshooting commands
- Displaying commands stored in the SQL Command Cache

Managing performance and troubleshooting commands

You can view all one or more of the SQL commands or details of their query plans with the `ttSqlCmdCacheInfo` and `ttSqlCmdQueryPlan` built-in procedures. Use the query plan information to monitor and troubleshoot your queries.

Viewing the SQL commands and query plans can help you perform the following:

- Detect updates or deletes that are not using an index scan.
- Monitor query plans of executing queries to ensure all plans are optimized.
- Detect applications that do not prepare SQL statements or that re-prepare the same statement multiple times.
- Discover the percentage of space used in the command cache for performance evaluation.

Displaying commands stored in the SQL Command Cache

The commands executed against the TimesTen database are cached in the SQL command cache. The `ttSqlCmdCacheInfo` built-in procedure displays a specific or all cached commands in the TimesTen SQL command cache. By default, all commands are displayed; if you specify a command id, then only this command is retrieved for display.

The command data is saved in the following format:

- Command identifier, which is used to retrieve a specific command or its associated query plan.
- Private connection identifier.
- Counter for the number of executions.
- Counter for the number of times the user prepares this statement.
- Counter for the number of times the user re-prepares this statement.
- Freeable status, where if the value is one, then the subdaemon can free the space with the garbage collector. A value of zero determines that the space is not able to be freed.
- Total size in bytes allocated for this command in the cache.
- User who created the command.
- Query text up to 1024 characters.

At the end of the list of all SQL commands, a status is printed of how many commands were in the cache.

The following examples show how to display all or a single SQL command from the SQL Command Cache using the `ttSqlCmdCacheInfo` built-in utility:

- Displaying all SQL commands in the SQL Command Cache
- Displaying a single SQL command

Example 10–1 Displaying all SQL commands in the SQL Command Cache

This example executes within `ttIsql` the `ttSqlCmdCacheInfo` built-in procedure without arguments to show all cached SQL commands. The SQL commands are displayed in terse format. To display the information where each column is prepended with the column name, execute `vertical on` before executing the `ttsqlCmdCacheInfo` procedure.

```
Command> call ttsqlCmdCacheInfo;
< 528079360, 2048, 0, 1, 0, 1, 2168, PAT                            , select * from t7
where x7 is not null or exists (select 1 from t2,t3 where not 'tuf' like 'abc') >
< 527609108, 2048, 0, 1, 0, 1, 2960, PAT                            , select * from t1
where x1 = (select x2 from t2 where z2 in (1,3) and y1=y2) order by 1, 2, 3 >
```

```
< 528054656, 2048, 0, 1, 0, 1, 1216, PAT                        , create table
t2(x2 int,y2 int, z2 int) >
< 528066648, 2048, 0, 1, 0, 1, 1176, PAT                        , insert into t2
select * from t1 >
< 528013192, 2048, 0, 1, 0, 1, 1848, PAT                        , select * from t1
where exists (select * from t2 where x1=x2) or y1=1 >
< 527582620, 2048, 0, 1, 0, 1, 1240, PAT                        , insert into t2
select * from t1 >
< 527614292, 2048, 0, 1, 0, 1, 2248, PAT                        , select * from t1
where exists (select x2 from t2 where x1=x2) order by 1, 2, 3 >
< 528061248, 2048, 0, 1, 0, 1, 696, PAT                        , create index i1
on t3(y3) >
< 528070368, 2048, 0, 1, 0, 1, 824, PAT                        , call
ttOptSetOrder('t3 t4 t2 t1') >
< 528018856, 2048, 0, 1, 0, 1, 984, PAT                        , insert into t2
select * from t1 >
< 527606460, 2048, 0, 1, 0, 1, 2624, PAT                        , select * from t1
where x1 = (select x2 from t2 where y1=y2) order by 1, 2, 3 >
< 528123000, 2048, 0, 1, 0, 1, 3616, PAT                        , select * from t1
where x1 = 1 or x1 = (select x2 from t2,t3 where z2=t3.x3) >
< 528074624, 2048, 0, 1, 0, 1, 856, PAT                        , call
ttOptSetOrder('t4 t2 t3 t1') >
< 527973892, 2048, 0, 1, 0, 1, 2872, PAT                        , select * from t1
where x1 in (select x2 from t2) or x1 in (select x3 from t3) order by 1, 2, 3 >
< 527953876, 2048, 0, 1, 0, 1, 3000, PAT                        , select * from t1
where x1 = (select x2 from t2) order by 1, 2, 3 >
< 527603900, 2048, 0, 1, 0, 1, 2440, PAT                        , select * from t1
where x1 in (select x2 from t2 where y1=y2) order by 1, 2, 3 >
< 528093308, 2048, 0, 1, 0, 1, 3608, PAT                        , select * from t1
where x1 = 1 or x1 = (select x2 from t2,t3 where z2=t3.x3 and t3.z3=1) >
< 528060608, 2048, 0, 1, 0, 1, 696, PAT                        , create index i1
on t2 (y2) >
```

Example 10–2 Displaying a single SQL command

If you provide a command id as the input for the `ttSqlCmdCacheInfo`, the single SQL command is displayed from within the SQL Command Cache. You can discover the command id from executing this built-in without input. The command id is the first column displayed.

The following example displays the SQL command identified by command id of 527973892. It is displayed in terse format; to view with the column headings prepended, execute `vertical on` before executing the `ttSqlCmdCacheInfo` built-in.

```
Command> call ttsqlCmdCacheInfo(527973892);
< 527973892, 2048, 0, 1, 0, 1, 2872, PAT                        , select * from t1
where x1 in (select x2 from t2) or x1 in (select x3 from t3) order by 1, 2, 3 >
1 row found.
```

Viewing SQL query plans

You can view the query plan for a SQL command in one of two ways: storing the latest query plan into the system PLAN table or viewing all cached SQL commands and their query plans in the SQL command cache. Both methods are described in the following sections:

- Viewing a query plan from the system PLAN table

- Viewing query plans associated with commands stored in the SQL Command Cache

Viewing a query plan from the system PLAN table

The optimizer prepares the query plans. For the last SQL command to be executed—such as a prepared SELECT, UPDATE, DELETE, INSERT SELECT, CREATE TABLE, CREATE MATERIALIZED VIEW and so on—you can instruct that the plan be stored in the system PLAN table:

1. Instruct TimesTen to generate the plan and store it in the system PLAN table.
2. Prepare the statement means calling the ODBC `SQLPrepare` function or JDBC `Connection.prepareStatement` method on the statement. TimesTen stores the plan into the PLAN table.
3. Read the generated plan within the SYS.PLAN table.

The stored plan is updated automatically whenever the command is reprepared. Re-preparation occurs automatically if one ore more of the following occurs:

- A table in the statement is altered.
- If indexes are created or dropped.
- The application invalidates commands when statistics are updated with the invalidate option in the `ttOptUpdateStat` built-in procedure.

For these cases, read the PLAN table to view how the plan has been modified.

Instruct TimesTen to store the plan in the system PLAN table

Before you can view the plan in the system PLAN table, call the built-in `ttOptSetFlag` procedure 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 completes after the processing of the command and prepares in the next transaction are not 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 is prepared with the `genPlan` flag set, it is recompiled with this flag. Thus, the plan is generated even though the plan for another query was found in the SYS.PLAN table.

For example, try the query and optimizer hints with the `ttIsql` utility. To display optimizer plans, issue the following commands:

```
autocommit 0;
showplan 1;
```

Reading query plan from 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 "System and Replication Tables" in the *Oracle TimesTen In-Memory Database SQL Reference*.

Example 10–3 Generating a query plan

This example uses 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 generates 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	Operation	TblNames	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

For details about each column in the SYS.PLAN table, see "Describing the PLAN table columns" on page 10-7.

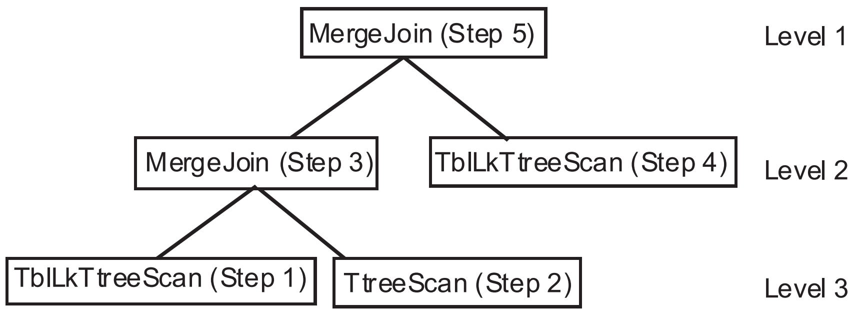
Describing the PLAN table columns

The SYS.PLAN table has seven columns.

Column 1 (Step) Indicates the order of operation, which always starts with one. Example 10–3 uses a table lock range scan in the following order:

1. Table locking range scan of IX1 on table T1.
2. Table locking range scan of IX2 on T2.
3. Merge join of T1 and T2 and so forth.

Column 2 (Level) Indicates the position of the operation in the join-tree diagram that describes the execution. For Example 10–3, the join tree is as follows:



Column 3 (Operation) Indicates the type of operation being executed. For a description of the potential values in this field and the type of table scan each represents, see SYS.PLAN in "System and Replication Tables" in the *Oracle TimesTen In-Memory Database SQL Reference*.

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. 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—such as a hash or range scan. In all other cases, this column is NULL. Names of range 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. The predicate character string is limited to 1,024 characters.

This column may be NULL—indicating no predicate—for a range scan. The optimizer may choose a range scan over a table scan because, in addition to filtering, it has two useful properties:

- Rows are returned in sorted order, on index key.
- Rows may be returned faster, especially if the table is sparse.

In Example 10–3, the range scans are used for their sorting capability; none of them evaluates a predicate.

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 two of the plan generated for Example 10–3, a range 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`.

Viewing query plans associated with commands stored in the SQL Command Cache

Use the query plan information to monitor and troubleshoot your queries.

Note: For more reasons why to use the `ttSqlCmdQueryPlan` built-in procedure, see "Managing performance and troubleshooting commands" on page 10-3.

The `ttSqlCmdQueryPlan` built-in procedure displays the query plan of a specific statement or all statements in the command cache. It displays the detailed run-time query plans for the cached SQL queries. By default, all query plans are displayed; if you specify the command id taken from the SQL command output, only the query plan for the specified SQL command is displayed.

Note: If you want to display a query plan for a specific command, you must provide the command identifier, which is displayed with the `ttSqlCmdCacheInfo` built-in procedure. See "Displaying commands stored in the SQL Command Cache" on page 10-4 for full details.

The plan data displayed when you invoke this built-in procedure is as follows:

- Command identifier
- Query text up to 1024 characters
- Step number of the current operation in the run-time query plan
- Level number of the current operation in the query plan tree
- Operation name of current step
- Name of table used
- Owner of the table
- Name of index used
- If used and available, the index predicate
- If used and available, the non-indexed predicate

Note: For more information on how to view this information, see "Reading query plan from the PLAN table" on page 10-6. The source of the data may be different, but the mapping and understanding of the material is the same as the query plan in the system PLAN table.

The following examples show how to display all or a single SQL query plan from the SQL Command Cache using the `ttSqlCmdQueryPlan` built-in procedure:

- Displaying all SQL query plans
- Displaying a single SQL query plan

Example 10–4 Displaying all SQL query plans

You can display all SQL query plans associated with commands stored in the command cache with the `ttSqlCmdQuery` plan built-in procedure within the `ttIsql` utility.

The following example shows the output when executing the `ttSqlCmdQueryPlan` built-in procedure without arguments, which displays detailed run-time query plans for all valid queries. For invalid queries, there is no query plan; instead, the query text is displayed.

The query plans are displayed in terse format; to view with the column headings prepended, execute `vertical on` before executing the `ttSqlCmdQueryPlan` built-in procedure.

Note: For complex expressions, there may be some difficulties in printing out the original expressions.

```
Command> call ttSqlCmdQueryPlan();

< 528079360, select * from t7 where x7 is not null or exists (select 1 from t2,t3
where not 'tuf' like 'abc'), <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>,
<NULL>, <NULL> >
< 528079360, <NULL>, 0, 2, RowLkSerialScan              , T7
, PAT                        ,                                , ,  >
< 528079360, <NULL>, 1, 3, RowLkTtreeScan               , T2
, PAT                        , I2                             , , NOT(LIKE( tuf
,abc ,NULL ))  >
< 528079360, <NULL>, 2, 3, RowLkTtreeScan               , T3
, PAT                        , I2                             , ,  >
< 528079360, <NULL>, 3, 2, NestedLoop                   ,
,                                ,                                , ,  >
< 528079360, <NULL>, 4, 1, NestedLoop(Left OuterJoin)    ,
,                                ,                                , ,  >
< 528079360, <NULL>, 5, 0, Filter                       ,
,                                ,                                , , X7 >
< 527576540, call ttSqlCmdQueryPlan(527973892), <NULL>, <NULL>, <NULL>, <NULL>,
<NULL>, <NULL>, <NULL>, <NULL> >
< 527576540, <NULL>, 0, 0, Procedure Call               ,
,                                ,                                , ,  >
< 528054656, create table t2(x2 int,y2 int, z2 int), <NULL>, <NULL>, <NULL>,
<NULL>, <NULL>, <NULL>, <NULL>, <NULL> >
< 528066648, insert into t2 select * from t1, <NULL>, <NULL>, <NULL>, <NULL>,
<NULL>, <NULL>, <NULL>, <NULL> >
< 528066648, <NULL>, 0, 0, Insert                       , T2
, PAT                        ,                                , ,  >
< 528013192, select * from t1 where exists (select * from t2 where x1=x2) or y1=1,
<NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL> >
< 528061248, create index i1 on t3(y3), <NULL>, <NULL>, <NULL>, <NULL>, <NULL>,
<NULL>, <NULL>, <NULL> >
< 528070368, call ttOptSetOrder('t3 t4 t2 t1'), <NULL>, <NULL>, <NULL>, <NULL>,
<NULL>, <NULL>, <NULL>, <NULL> >
< 528070368, <NULL>, 0, 0, Procedure Call               ,
,                                ,                                , ,  >
< 528018856, insert into t2 select * from t1, <NULL>, <NULL>, <NULL>, <NULL>,
<NULL>, <NULL>, <NULL>, <NULL> >
< 527573452, call ttsqlCmdCacheInfo(527973892), <NULL>, <NULL>, <NULL>, <NULL>,
<NULL>, <NULL>, <NULL>, <NULL> >
< 527573452, <NULL>, 0, 0, Procedure Call               ,
,                                ,                                , ,  >
< 528123000, select * from t1 where x1 = 1 or x1 = (select x2 from t2,t3 where
z2=t3.x3), <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL> >
< 528123000, <NULL>, 0, 2, RowLkSerialScan              , T1
, PAT                        ,                                , ,  >
< 528123000, <NULL>, 1, 6, RowLkTtreeScan               , T2
, PAT                        , I2                             , ,  >
< 528123000, <NULL>, 2, 6, RowLkTtreeScan               , T3
, PAT                        , I2                             , ,  Z2 = X3; >
< 528123000, <NULL>, 3, 5, NestedLoop                   ,
,                                ,                                , ,  >
< 528123000, <NULL>, 4, 4, Materialized View            ,
,                                ,                                , ,  >
< 528123000, <NULL>, 5, 3, GroupBy                      ,
,                                ,                                , ,  >
< 528123000, <NULL>, 6, 2, Filter                       ,
,                                ,                                , ,  X1 = colum_
name; >
< 528123000, <NULL>, 7, 1, NestedLoop(Left OuterJoin)    ,
,                                ,                                , ,  >
< 528123000, <NULL>, 8, 0, Filter                       ,
,                                ,                                , ,  X1 = 1; >
```

Example 10–5 Displaying a single SQL query plan

You can display any query plan associated with a SQL command by providing the command id of the SQL command as the input for the `ttSqlCmdQueryPlan` built-in procedure. The single query plan is displayed from within the SQL Command Cache. You can discover the command id from executing this `ttSqlCmdCacheInfo` built-in without input. The command id is the first column displayed.

The following example displays the query plan of the SQL command identified by command id of 528078576. It is displayed in terse format; to view with the column headings prepended, execute `vertical on` before executing the `ttSqlCmdQueryPlan` built-in procedure.

Note: for complex expressions, there are some difficulties to print original expressions.

```
Command> call ttSqlCmdQueryPlan( 528078576);
< 528078576, select * from t1 where 1=2 or (x1 in (select x2 from t2, t5 where y2
in (select y3 from t3)) and y1 in (select x4 from t4)), <NULL>, <NULL>, <NULL>,
<NULL>, <NULL>, <NULL>, <NULL>, <NULL> >
< 528078576, <NULL>, 0, 4, RowLkSerialScan             , T1
, PAT                        ,                                , ,  >
< 528078576, <NULL>, 1, 7, RowLkTtreeScan              , T2
, PAT                        , I2                             , ,  >
< 528078576, <NULL>, 2, 7, RowLkTtreeScan              , T5
, PAT                        , I2                             , ,  >
< 528078576, <NULL>, 3, 6, NestedLoop                  ,
,                                ,                                , ,  >
< 528078576, <NULL>, 4, 6, RowLkTtreeScan              , T3
, PAT                        , I1                             ,  ( (Y3=Y2; ) ) ,
>
< 528078576, <NULL>, 5, 5, NestedLoop                  ,
,                                ,                                , ,  >
< 528078576, <NULL>, 6, 4, Filter                     ,
,                                ,                                , ,  X1 = X2; >
< 528078576, <NULL>, 7, 3, NestedLoop(Left OuterJoin)    ,
,                                ,                                , ,  >
< 528078576, <NULL>, 8, 2, Filter                     ,
,                                ,                                , ,  >
< 528078576, <NULL>, 9, 2, RowLkTtreeScan              , T4
, PAT                        , I2                             , ,  Y1 = X4; >
< 528078576, <NULL>, 10, 1, NestedLoop(Left OuterJoin)    ,
,                                ,                                , ,  >
< 528078576, <NULL>, 11, 0, Filter                     ,
,                                ,                                , ,  >
13 rows found.
Command>
```

Modifying plan generation

If you decide that you want to modify a query plan, you can only modify the query plan that exists in the system PLAN table, as described in "Viewing a query plan from the system PLAN table" on page 10-6. Once you do modify the query plan, it does not replace the query plan, but creates a new query plan with your changes.

The following sections describe why you may want to modify execution plans and then how to modify them:

- Why modify an execution plan?
- When to modify an execution plan
- How to modify execution plan generation

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 performant.** The query optimizer chooses the plan that it estimates will execute the 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 providing hints to the optimizer. Hints are specified by calls to one of the TimesTen optimizer built-in procedures and are in effect for all calls to the ODBC `SQLPrepare` function or JDBC PreparedStatement objects in the transaction. For more information on how to provide these hints, see "How to modify execution plan generation" on page 10-14.

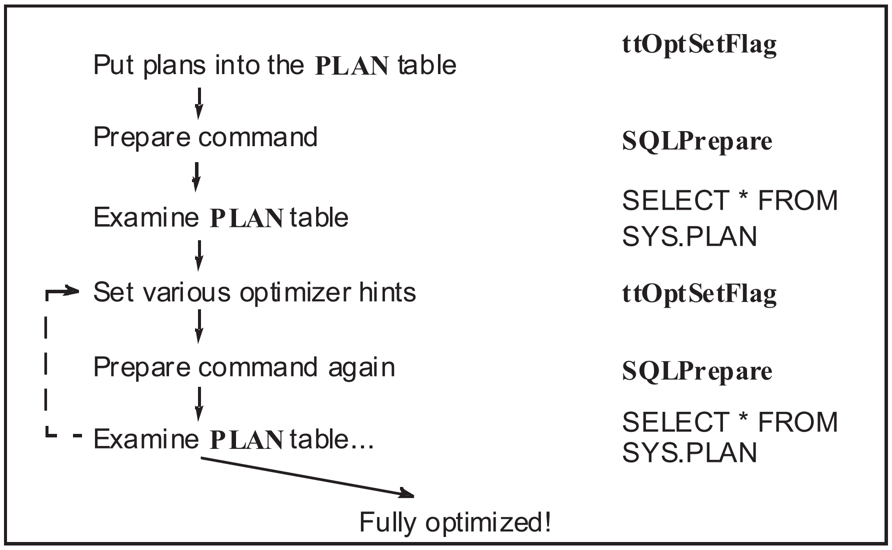
Note: Make sure AUTOCOMMIT is not set. If it is, the current transaction completes after processing the `ttOptSetFlag` procedure and prepares in the next transaction are not affected.

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 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 10-1.

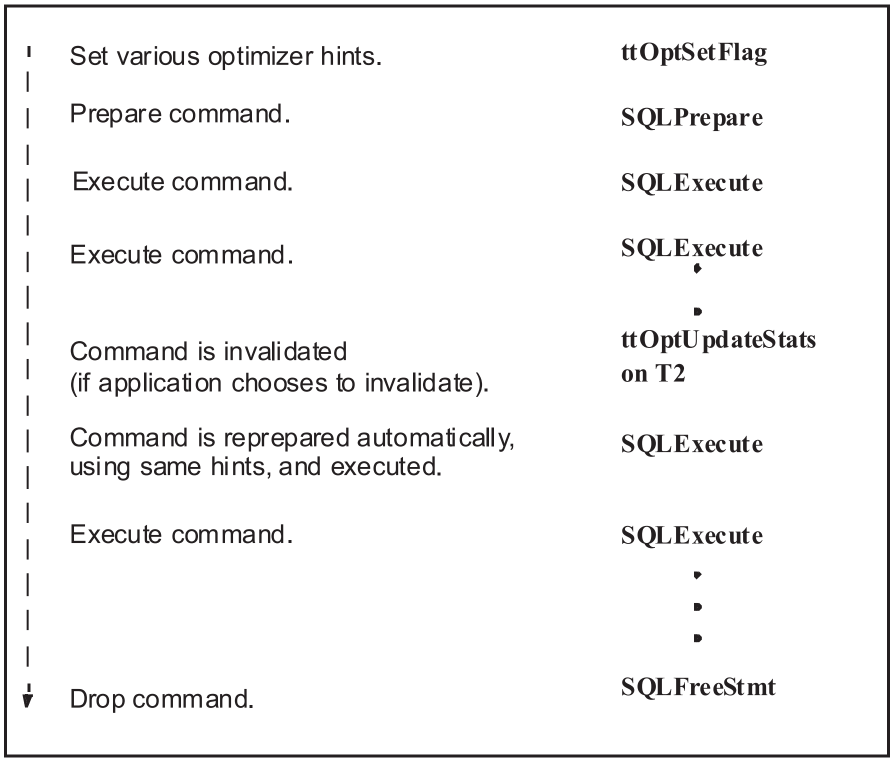
If a command is prepared without hints, subsequent hints will not affect the command if it is reprepared automatically. An application must call the ODBC `SQLPrepare` function or JDBC `Connection.prepareStatement` method a second time so that hints have an effect.

Example 10–6   Tuning a join when using ODBC

When using ODBC, a developer tuning a join on T1 and T2 might go through the steps shown in the following figure.

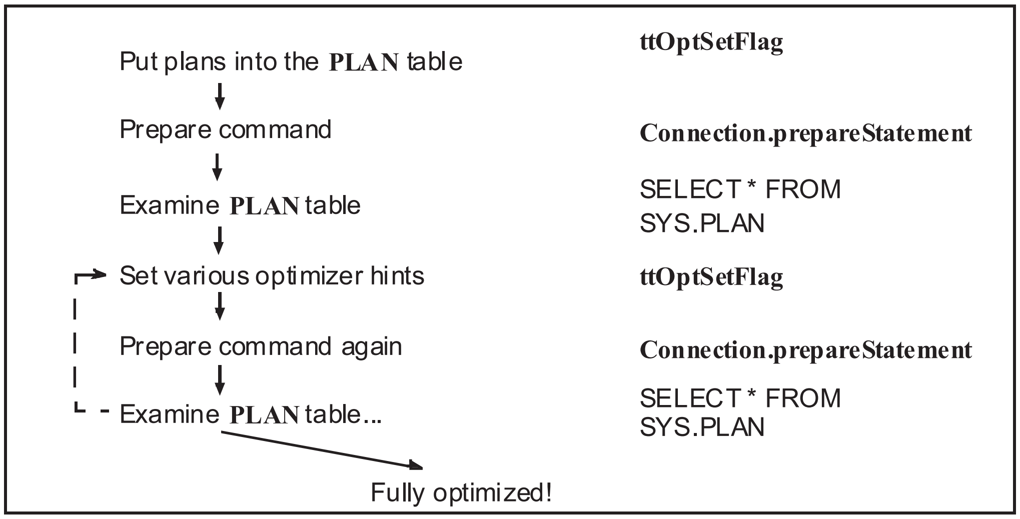


During execution, the application may then go through the steps shown in the following figure.

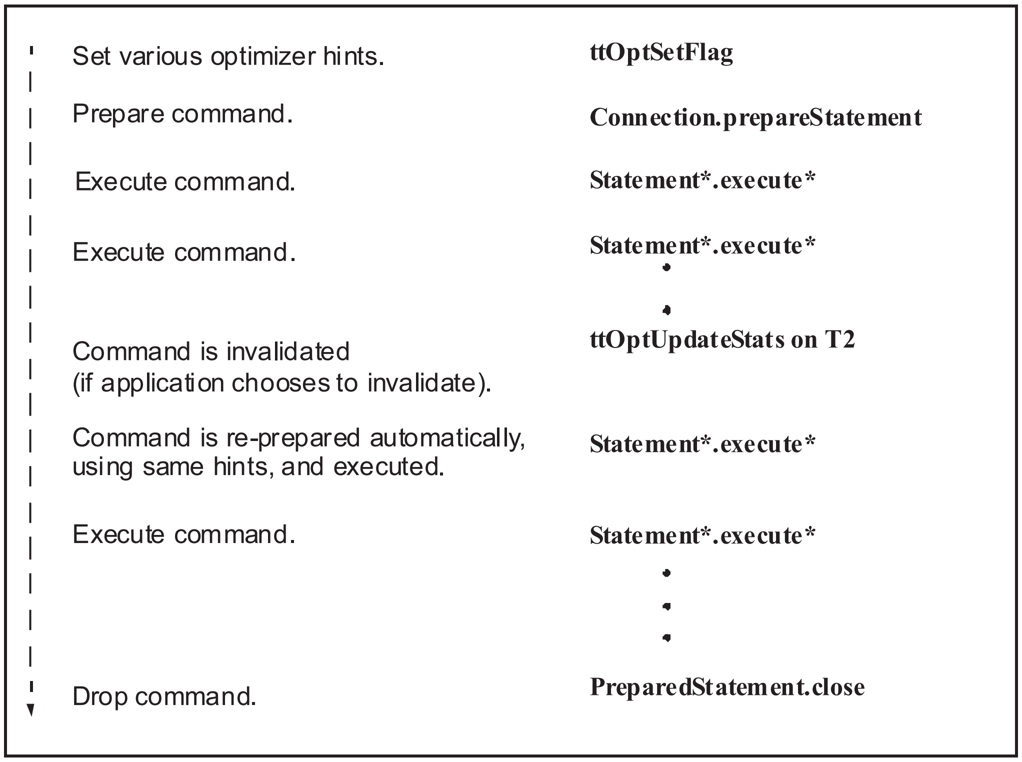


Example 10–7 Tuning a join when using JDBC

When using JDBC, a developer tuning a join on T1 and T2 might go through the steps shown in the following figure.



During execution, the application may then go through the steps shown in the following figure.



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:

- `ttOptSetFlag`—Sets certain optimizer parameters.
- `ttOptSetOrder`—Allows an application to specify the table join order.
- `ttOptUseIndex`—Allows an application to specify that an index be used or to disable the use of certain indexes.

- ttOptClearStats, ttOptEstimateStats, ttOptSetColIntvlStats, ttOptSetTblStats, ttOptUpdateStats—Manipulate statistics that the TimesTen Data Manager maintains on the application's data that are used by the query optimizer to estimate costs of various operations.

Some of these built-in procedures require that the user have privileges to the objects on which the utility executes. For full details on these built-in procedures and any privileges required, see "Built-In Procedures" in the *Oracle TimesTen In-Memory Database Reference*.

The following examples provide an ODBC and JDBC method on how to use the ttOptSetFlag built-in procedure:

> **Note:** You can also experiment with optimizer settings using the ttIsql utility. The commands that start with try control the optimizer hints. To view your current optimizer hint settings, use the optprofile command.

- Example 10–8—JDBC example.
- Example 10–9—ODBC example.

Example 10–8 Using ttOptSetFlag in ODBC

This JDBC example 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 10–9 Using ttOptSetFlag in ODBC

This ODBC example 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 */
...
```

Glossary

.odbc.ini file

See "ODBC initialization file (ODBC INI)".

ACID transaction semantics

An acronym referring to the four fundamental properties of a transaction: atomicity, consistency, isolation and durability.

atomicity

A property of a transaction whereby either all or none of the operations of a transaction are applied to the data store.

backup instance

A set of files containing backup information for a given data store, residing at a given backup path. See also "backup path", "full backup" and "incremental backup".

backup path

The location of a data store, specified by a directory name and an optional basename.

backup point

The time at which a backup begins. See also "backup path", "full backup" and "incremental backup".

bitmap index

Indexes are used to speed up queries on a table. Bitmap indexes are useful when searching and retrieving data from columns with low cardinality. That is, these columns can have only a few unique possible values.

cache group

A set of cached tables related through foreign keys.

cache instance

A set of rows related through foreign keys. Each cache instance contains exactly one row from the root table of a cache group and zero or more rows from the other tables in the cache group.

client/server

An approach to application design and development in which application processing is divided between components running on an end user's machine, such as the client, and a network server. Generally, user interface elements are implemented in the client component, while the server controls database access.

client data source name

See "data source name, client".

concurrency

The ability to have multiple transactions access and manipulate the data store at the same time.

connection

A data path between an application and a particular ODBC data source.

connection attribute

A character string that defines a connection parameter to be used when connecting to an ODBC data source. Connection attributes have the form *name=value*, where *name* is the name of the parameter and *value* is the parameter value. See also connection string.

connection request

A message sent by an application through an ODBC driver to an ODBC data source to request a connection to that data source.

connection string

A character string that defines the connection parameters to be used when connecting to an ODBC data source. A connection string is expressed as one or more connection attributes separated by semicolons.

consistency

A property of transactions whereby each transaction transforms the database from one consistent state to another.

cursor

A control structure used by an application to iterate through the results of an SQL query.

data source definition

A named collection of connection attributes that defines the connection parameters to be used when connecting to an ODBC data source. See also "data source name".

data source name

A logical name by which an end user or application refers to an ODBC data source definition. Sometimes incorrectly used to mean "data source definition". See also "data source definition", ODBC.INI file.

data source name, client

A data source name defined on a TimesTen client machine that refers to a TimesTen server data source name on a server machine.

data source name, server

A system data source name (system DSN) defined on a server machine. Server Data Source Names become available to all TimesTen clients on a network when the TimesTen Server is running.

data source name, system

A data source name that is accessible by all users of a particular machine.

data source name, user

A data source name that is accessible only by the user who created the data source name.

driver

See "ODBC driver".

DSN

See "data source name".

DSN, client

See "data source name, client".

DSN, server

See "data source name, server".

DSN, system

See "data source name, system".

DSN, user

See "data source name, user".

durability

A property of transactions whereby the effects of a committed transaction survive system failures.

environment variable

A *name*, *value* pair maintained by the operating system that can be used to pass configuration parameters to an application.

event

An activity or occurrence that can be tracked by a logging mechanism in an application, service or operating system. See also "logging", "protocol message logging" and "event viewer".

event viewer

On Windows, a utility program used to view the contents of the operating system event log.

full backup

A data store backup procedure in which a complete copy of a data store is created. Typically, the first backup of a data store must be a full backup. See also "incremental backup".

hash index

Indexes are used to speed up queries on a table. Hash indexes are useful for finding rows with an exact match on one or more columns.

host

A computer. Typically used to refer to a computer on a network that provides services to other computers on the network.

host name

A character string name that uniquely identifies a particular computer on a network. Examples: `athena`, `thames.mycompany.com`. See also "host".

in-line column

A column whose values are physically stored together with the other column values of a row.

incremental backup

A data store backup procedure in which an existing backup is augmented with all the log records created since its last full or incremental backup. See also "backup instance" and "full backup".

initialization file

See ODBC.INI file.

IP address

A numeric address that uniquely identifies a computer on a network and consists of four numbers separated by dots. Abbreviation for Internet Protocol address. Example: `123.61.129.91`.

IPC

Inter Process Communication

isolation

A property of transactions whereby each transaction runs as if it were the only transaction in the system.

listener thread

A thread that runs on the TimesTen Server that receives and processes connection requests from TimesTen Clients.

logging

The process by which an application, service or operating system records specific events that occur during processing.

multithreading

A programming paradigm in which a process contains multiple threads of control.

network address

A host name, or IP address that uniquely identifies a particular computer on a network. Examples: `123.61.129.91`, `athena`, `thams.mycompany.com`.

ODBC

See "Open Database Connectivity (ODBC)".

ODBC Administrator

A utility program used on Windows to create, configure and delete data source definitions.

ODBC data source

See "data source name" (DSN).

ODBC data source name

See "data source name" (DSN).

ODBC driver

A library that implements the function calls defined in the ODBC API and enables applications to interact with ODBC data sources.

ODBC Driver Manager

A library that acts as an intermediary between an ODBC application and one or more ODBC drivers.

ODBC initialization file (ODBC INI)

The ODBC.INI file contains a list of Data Sources and any properties for each. Each Data Source name must have a driver property defined. This enables the Driver Manager to load the driver when a connect call is made.

Open Database Connectivity (ODBC)

A database-independent application programming interface that enables applications to access data stored in heterogeneous relational and non-relational databases. Based on the Call-Level Interface (CLI) specification developed by X/Open's SQL Access Group and first popularized by Microsoft on the Windows platform.

Open database connectivity (ODBC), is a database access protocol that lets you connect to a database and then prepare and run SQL statements against the database. In conjunction with an ODBC driver, an application can access any data source including data stored in spreadsheets, like Excel. Because ODBC is a widely accepted standard API, applications can be written to comply to the ODBC standard. The ODBC driver performs all mappings between the ODBC standard and the particular database the application is accessing. Using a data source-specific driver, an ODBC compliant program can access any data source without any more development effort.

TimesTen provides the ODBC interface so that applications of any type that are ODBC compliant can access TimesTen using the ODBC driver provided by TimesTen.

out-of-line column

A column whose values are physically stored separately from the other column values of a row.

phantom

A row that appears during one read but not during another read within the same transaction, due to the actions of other concurrently executing transactions.

ping

A utility that tests the connection between two computers on a network by sending a message from one computer to the other and measuring how long it takes for the receiving system to confirm that the message was received. Typically packaged with network software.

port number

See "TCP/IP port number".

procedure

See "stored procedure".

process

An instance of a program in execution.

propagate

When using Cache Connect to Oracle (IMDB Cache) to send table or row modifications from an IMDB Cache to an Oracle database. Compare with "replicate".

protocol message logging

The process that the TimesTen Server uses to record each message it receives through the TimesTen network protocol.

range index

Indexes are used to speed up queries on a table. A range index is similar in functionality to a B+-tree index and is best used for retrieving rows with column values within a certain range.

replicate

The sending of table or row modifications from one data store to another. Compare with "propagate".

result set

A collection of zero or more rows of data that represent the result of an SQL query.

rollback

To undo the actions of a transaction, thereby returning all items modified by the transaction to their original state.

row buffering

A performance enhancement used by the TimesTen Client in which the client receives multiple result rows of an SQL query in each message from the TimesTen Server to reduce network communication.

RPC

Remote Procedure Call.

scalability

The degree to which a system or application can handle increasing demands on system resources without significant performance degradation.

schema

A schema is automatically created for a user upon user creation. A schema is the namespace for a given user, where all objects owned by this user belong and all objects are identified by schema qualified names. For example, user PAT belongs to the PAT schema. In addition, the object EMPLOYEES owned by PAT is identified as PAT.EMPLOYEES.

If a user refers to an object without the schema name, TimesTen first tries to resolve the name to the user’s schema. If this object does not exist, TimesTen tries to resolve the name to SYS.EMPLOYEES.

A user always has all privileges to all objects in their own schema. These privileges can never be revoked.

server data source name

See "data source name, server".

server DSN

See "data source name, server".

system DSN

See "data source name, system".

shorthand name

A logical name used to refer to a particular TimesTen Server. Shorthand names relieve the end user of having to enter a host name and port number to connect to a TimesTen Server.

SMP

Symmetric multi-processing. A hardware configuration in which two or more similar processors are connected via a high-bandwidth link and managed by one operating system, where each processor has equal access to I/O devices.

SNMP

Simple Network Management Protocol. Used to manage nodes on a network.

SQL

Structured Query Language.

stack overflow condition

An error condition in which the stack usage of a thread or process exceeds the amount of space allocated for the stack.

stored procedure

An executable object or named entity stored in a data store that can be invoked with input and output parameters and which can return result sets similar to those returned by an SQL query.

system account

A special account on Windows used by the operating system and certain operating system services. The TimesTen Service and the TimesTen Server run under the system account.

system DSN

See "data source name, system".

TCP/IP

The communications protocol used by computers on the Internet. Abbreviation for Transport Control Protocol/Internet Protocol.

TCP/IP port number

A number used by TCP/IP that identifies the end point for a connection to a host that supports multiple simultaneous connections.

telnet

A utility program and protocol that enables a user on one computer to open a virtual terminal, log in to a remote host and interact as a terminal user of that host.

thread

An independent sequence of execution of program code inside a process. See also "process".

thread-safe ODBC driver

An ODBC driver that supports multithreaded servers and clients. The TimesTen data manager driver and the TimesTen Client driver are thread-safe.

timeout error

An error condition indicating that the requested operation did not complete within the given amount of time. See also "timeout interval".

timeout interval

A configuration parameter that specifies the maximum amount of time that an operation should take to complete. See also "timeout error".

TimesTen Client

(1) An ODBC driver that enables end users to access data sources through a TimesTen Server. (2) A computer on which the TimesTen Client software has been installed. Using the TimesTen Client driver, an end user or application can access any data source managed by an available TimesTen Server.

TimesTen Client/Server network protocol

The protocol used by TimesTen Clients and TimesTen Servers to exchange data over a standard TCP/IP network connection.

TimesTen Data Server

(1) An application program that makes TimesTen data sources available to the TimesTen Clients on a network. (2) A computer on which the TimesTen Data Server software is running.

TimesTen Server address

The host name or IP address used during installation of the TimesTen Server to identify the computer on which the software is being installed.

transaction

An operation or set of operations performed against data in a data store. The operations defined in a transaction must be completed as a whole; if any part of the transaction fails, the entire transaction fails. See also "ACID transaction semantics".

UCS-4

A fixed-width, 32-bit Unicode character set. Each character occupies 32 bits of storage. The UCS-2 characters are the first 65,536 code points in this standard, so it can be viewed as a 32-bit extension of UCS-2.

UTF-16

An encoding scheme defined by the ISO/IEC 10646 standard in which each Unicode character is represented by either a two-byte integer or a pair of two-byte integers. Characters from European scripts and most Asian scripts are represented in two bytes. Surrogate pairs are represented in four bytes. Surrogate pairs represent characters such as infrequently used Asian characters that were not included in the original range of two-byte characters.

user account

The combination of a user name, password and access permissions that gives an individual user access to an operating system.

user data source name

See"data source name, user".

user DSN

See "data source name, user".

User Manager

A Windows utility program used to create user accounts and assign access rights and group membership.

Windows sockets (Winsock)

An API that defines a standard binary interface for TCP/IP transports on Windows platforms. This API adds Windows-specific extensions to the Berkeley Sockets interface originally defined in Berkeley UNIX.

Index