Oracle® TimesTen In-Memory Database

Operations Guide

11*g* Release 2 (11.2.2)

E21633-12

October 2014

ORACLE®

Oracle TimesTen In-Memory Database Operations Guide, 11*g* Release 2 (11.2.2)

E21633-12

Contents

Preface

Oracle TimesTen In-Memory Database (TimesTen) is a relational database that is memory-optimized for fast response and throughput. The database resides entirely in memory at runtime and is persisted to disk storage for the ability to recover and restart. Replication features allow high availability. TimesTen supports standard application interfaces JDBC, ODBC, and ODP.NET, in addition to Oracle interfaces PL/SQL, OCI, and Pro*C/C++. TimesTen is available separately or as a cache for Oracle Database.

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/technetwork/database/database-technologies/timesten/documentation/index.html

Oracle Database documentation is also available on the Oracle Technology network. This may be especially useful for Oracle Database features that TimesTen supports, but does not document.

http://www.oracle.com/pls/db112/homepage/

Conventions

TimesTen supports multiple platforms. Unless otherwise indicated, the information in this guide applies to all supported platforms. The term Windows refers to all supported Windows platforms. The term UNIX applies to all supported UNIX and Linux platforms. See "Platforms" in *Oracle TimesTen In-Memory Database Release Notes* for specific platform versions supported by TimesTen.

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

This document uses the following text conventions:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text.
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.
`italic monospace`	Italic monospace type indicates a variable in a code example that you must replace. For example: `Driver=`*`TimesTen_install_dir`*`/lib/libtten.so` Replace *`TimesTen_install_dir`* with the path of your TimesTen installation directory.
[ ]	Square brackets indicate that an item in a command line is optional.
{ }	Curly braces indicate that you must choose one of the items separated by a vertical bar (|) in a command line.
|	A vertical bar 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 prompt for the UNIX root user.

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

Convention	Meaning
`TimesTen_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, 32 or 64, that represent either the 32-bit or 64-bit version of the operating system.
`release` or *`rr`*	The first three parts in a release number, with or without dots. The first three parts of a release number represent a major TimesTen release. For example, 1122 or 11.2.2 represents TimesTen 11*g* Release 2 (11.2.2).
`jdk_version`	One or two digits that represent the major version number of the Java Development Kit (JDK) release. For example, 5 represents JDK 5.
`DSN`	The data source name.

Documentation Accessibility

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

Access to Oracle Support

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

What's New

This section summarizes the new features and functionality of Oracle TimesTen In-Memory Database Release 11.2.2 that are documented in this guide, providing links into the guide for more information.

- New features in release 11.2.2.8.0
- New features in release 11.2.2.7.0
- New features in release 11.2.2.6.0
- New features in release 11.2.2.5.0
- New features in release 11.2.2.4.0
- New features in release 11.2.2.2.0
- New features in release 11.2.2.1.0
- New features in release 11.2.2.0.0

New features in release 11.2.2.8.0

- You can use the `CkptReadThreads` connection attribute to set the number of threads that TimesTen uses to read the checkpoint files when loading the database into memory. For more information, see "Setting the number of checkpoint file read threads" on page 7-14 and "Set CkptReadThreads" on page 10-30.

New features in release 11.2.2.7.0

- You can use the `ttIsql edit` command to edit a file or edit `ttIsql` commands in a text editor. The `ttIsql edit` command starts a text editor such as `emacs`, `gedit`, or `vi`. For more information, see "Using the ttIsql edit command" on page 6-8.
- The default size of the user log file and support log file has been increased. The new default size of the user log file is 10 MB and the support log file is 100 MB. For more information, see "Modifying informational messages" on page 3-6.

New features in release 11.2.2.6.0

- A parallel load operation may take a long time to execute. You can cancel and cleanly stop all threads that are performing a parallel load operation with either the `SQLCancel(`*`hstmt`*`)` ODBC function or by pressing Ctrl-C in the ttIsql utility. For more information, see "Cancel a parallel load operation" on page 6-31.
- TimesTen concurrently processes read and write queries optimally. Your read queries can be optimized for read-only concurrency when you use transaction

level optimizer hints such as `ttOptSetFlag ('tblLock',1)` or statement level optimizer hints such as `/*+ tt_tbllock(1) tt_rowlock(0) */`. Write queries that operate concurrently with read optimized queries may result in contention. You can control read optimization during periods of concurrent write operations with the `ttDbWriteConcurrencyModeSet` built-in procedure. For more information, see "Control read optimization during concurrent write operations" on page 10-23.

New features in release 11.2.2.5.0

- By default, if an automatic recovery of the database is unsuccessful after a fatal error, TimesTen changes the `always` and `manual` RAM policies to `InUse` to prevent reoccurring failures. You can now modify the default recovery process including preventing the RAM policy from changing with the `-enablePolicyInactive` option in the `ttendaemon.options` file. This option also stops the database from being reloaded. For details, see "Changes to RAM policy after automatic recovery fails" on page 1-26.
- When recovering a database, TimesTen reloads the database into memory. If the invalidated database still exists in memory, reloading the database can fill up available RAM. See "Preventing an automatic reload of the database after failure" on page 1-27 on how to stop automatic reloading of the database with the `ttAdmin -noautoreload` command.
- You can hold off command invalidation for commands in the SQL command cache until all statistics are generated or until after major changes to tables or indexes are complete. Once complete, you can manually invalidate the commands with the `ttOptCmdCacheInvalidate` built-in procedure. For more information, see "Control the invalidation of commands in the SQL command cache" on page 10-20.
- You can use the `CREATE [UNIQUE] HASH INDEX` statement to create a unique or nonunique hash index. For more information on creating hash indexes, see "Understanding indexes" on page 8-21.
- Historically, during transaction reclaim operations (the portion of a transaction where resources are cleaned up), a small number of the transaction log records have been cached to reduce the need to access the transaction log on disk. This helps only with relatively small transactions, however. To improve performance for large transactions as well, TimesTen has now added the `CommitBufferSizeMax` connection attribute, which allows users to configure the size of the cache. See "Transaction reclaim operations" on page 7-20 for details.
- TimesTen provides built-in procedures that measure and display the execution time of SQL operations to determine the performance of SQL statements. Instead of tracing, the built-in procedures sample the execution time of SQL statements as they are executed. For full details, see "Collect and evaluate sampling of execution times for SQL statements" on page 10-13.
- You can use the `ttStats` utility and `TT_STATS` PL/SQL package to collect and display database metrics. You can evaluate the collected metrics to evaluate the performance of your TimesTen database. See "Use metrics to evaluate performance" on page 10-7.
- You can apply hints to pass instructions to the TimesTen query optimizer. The optimizer considers these hints when choosing the best execution plan for your query. In this release, you can apply a statement level optimizer hint for a particular SQL statement. For details, see "Apply statement level optimizer hints for a SQL statement" on page 9-15.

- You can configure a range for all shared memory keys used by TimesTen with the `-shmkeyrange` daemon option. For details, see "Configuring a range for shared memory keys" on page 3-8.

New features in release 11.2.2.4.0

- A new tool, the Index Advisor, can be used to recommend a set of indexes that could improve the performance of a specific SQL workload. For more details, see "Using the Index Advisor to recommend indexes" on page 8-24.
- You can generate a SQL script with the `ttOptStatsExport` built-in procedure from which you can restore the table statistics to the current state. For more information, see "Create script to regenerate current table statistics" on page 10-19.
- New tools have been added that enable you to load the results of a SQL query from a back-end Oracle database into a single table on TimesTen without creating a cache grid, cache group, and cache table to contain the results. TimesTen provides the tools that execute a user provided `SELECT` statement on the Oracle database and load the result set into a table on TimesTen. For more information, see "Loading data from an Oracle database into a TimesTen table" on page 6-25.
- You can defragment a TimesTen database with the `ttMigrate` utility. For full details, see "Defragmenting TimesTen databases" on page 1-32.

  To defragment multiple databases with minimal overall service downtime, use a combination of the `ttMigrate` and `ttRepAdmin -duplicate` utilities to defragment TimesTen databases that are involved in an active standby pair replication scheme. For details, see "Online defragmentation of TimesTen databases" on page 1-33.

New features in release 11.2.2.2.0

- You can now declare, use, and set bind variables within `ttIsql` with the `variable` and `setvariable` commands. In addition, automatic bind variables can also be created. See "Using, declaring, and setting variables" on page 6-19 for more details.
- You can cancel an ODBC function with Ctrl-C. See "Canceling ODBC functions" on page 6-38 for more details.
- Use the `WHENEVER SQLERROR` command to control error recovery within `ttIsql`. For more details, see "Error recovery with WHENEVER SQLERROR" on page 6-39.
- Use the `IF-THEN-ELSE` command construct to implement conditional branching logic in a `ttIsql` session. See "Conditional control with the IF-THEN-ELSE command construct" on page 6-24 for more details.

New features in release 11.2.2.1.0

- It is important to verify at frequent intervals that there are no transaction log holds that could result in an excessive accumulation of transaction log files. If too many transaction log files accumulate and fill up available disk space, new transactions in the TimesTen database cannot begin until the transaction log hold is advanced and transaction log files are purged by the next checkpoint operation. See "Monitoring accumulation of transaction log files" on page 7-15 for more details.
- The LOB data type is now supported. See "SQL string and character functions" on page 5-7 for more details.

New features in release 11.2.2.0.0

- Range indexes used to be referred to as T-tree indexes. Now all output and commands use range as the identifying terminology. In "Viewing and changing query optimizer plans" on page 6-32, the `showplan` command now takes `tryrange` and `trytmprange` as options and the output for the query plan shows range indexes, such as `TmpRangeScan`.
- There is a new `tablesize` command within `ttIsql` that shows the actual sizes of tables within the TimesTen database. For full details, see "Using the ttIsql tablesize command" on page 6-11.

1

Managing TimesTen Databases

A TimesTen database is a collection of elements such as tables, views, and sequences. You can access and manipulate the TimesTen database through SQL. If your database does not exist, TimesTen creates the database with the specified attributes when the instance administrator connects to the database. You can free the database shared memory segment by disconnecting all existing connections to the TimesTen database.

Thus, this chapter describes first how to configure for a connection to the TimesTen database, because the configuration and management for your TimesTen database is contained in attributes within the connection definition.

Once you have created a database, you can perform the following:

- Use the `ttIsql` utility to connect to the database 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 database.

The main topics are as follows:

- Connecting to TimesTen with ODBC and JDBC drivers
- Specifying Data Source Names to identify TimesTen databases
- Defining a Data Manager DSN
- Defining client and server DSNs
- Resolution path for a DSN
- DSN examples
- odbc.ini file entry descriptions
- Specifying a RAM policy
- Loading and unloading the database from memory
- Specifying the size of a database
- Manage existing tables in the database
- Thread programming with TimesTen
- Defragmenting TimesTen databases

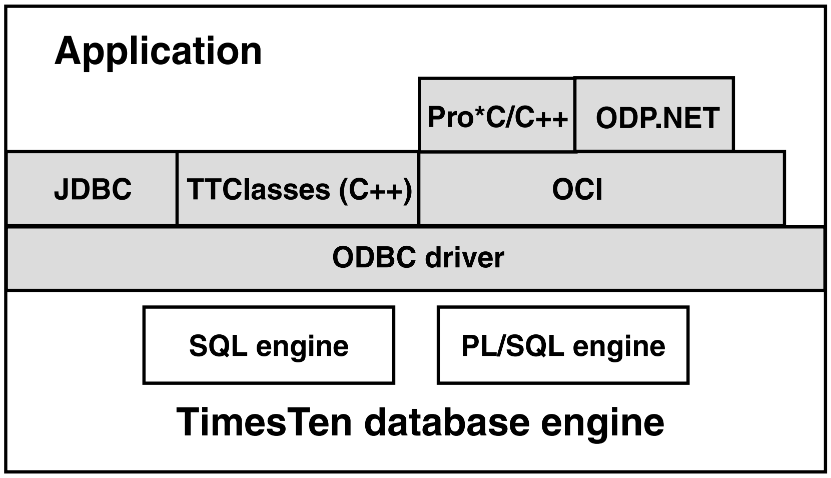
Connecting to TimesTen with ODBC and JDBC drivers

As described in "TimesTen connection options" in the *Oracle TimesTen Application-Tier Database Cache Introduction*, applications use the TimesTen ODBC driver to access a

TimesTen database. The application can use the ODBC direct driver, the Windows ODBC driver manager, the ODBC client driver or the ODBC driver indirectly through a provided interface to access the TimesTen database.

Figure 1–1 shows how the application can use different drivers and interfaces to access the TimesTen database.

Figure 1–1 Application access to TimesTen database diagram



- C applications interact with TimesTen by linking directly with the TimesTen ODBC driver, by linking with the Windows ODBC driver manager, or by using the OCI or Pro*C/C++ interfaces that access the ODBC driver.
- Java applications interact with TimesTen by loading the JDBC library.
- C++ applications interact with TimesTen through a TimesTen-provided set of classes called TTClasses or by using the OCI or Pro*C/C++ interfaces that access the ODBC driver.
- C# applications interact with TimesTen through Oracle Data Provider for .NET support for the TimesTen database.

Consider the following points:

- An application that links directly with an ODBC driver, whether it is linked with the direct driver or client driver, is limited to using only the driver with which it is linked. An application linked directly to either of the TimesTen drivers can connect to multiple databases at the same time.
  - The TimesTen direct driver supports multiple connections to multiple TimesTen databases, each of which are all the same TimesTen version.
  - The TimesTen client driver, used to facilitate a client/server connection, supports multiple connections to multiple TimesTen databases, which can be different TimesTen versions.

  This option offers less flexibility but better performance than linking with a driver manager.
- An application can link with more than one ODBC driver within the same application, even if the drivers are for different databases. If the application loads more than one ODBC driver, the application must use a driver manager, such as the Windows ODBC driver manager.

An application might need multiple drivers if it needs to use both the TimesTen direct driver and the TimesTen client driver.

The Windows ODBC driver manager dynamically loads an ODBC driver at runtime. However, carefully evaluate the benefits of using the ODBC driver manager, because it may affect the performance of your application with its additional runtime overhead.

Note: An application that is using an ODBC driver manager cannot use XLA.

For more information on how to compile an application that uses the TimesTen driver manager, see "Compiling and linking applications" in the *Oracle TimesTen In-Memory Database C Developer's Guide*, "Compiling Java applications" in the *Oracle TimesTen In-Memory Database Java Developer's Guide*, and "Compiling and linking applications" in the *Oracle TimesTen In-Memory Database TTClasses Guide*.

The following sections describe how to define TimesTen databases:

- Connecting using TimesTen ODBC drivers
- Connecting using the TimesTen JDBC driver and driver manager

Connecting using TimesTen ODBC drivers

TimesTen includes the following ODBC drivers:

- *TimesTen Data Manager driver*: A TimesTen ODBC driver for use with direct connect applications.
- *TimesTen Client driver*: A TimesTen Client ODBC driver for use with client/server applications.

TimesTen includes the following two versions of the Data Manager ODBC driver:

- *Production*: Use the *production* version of the TimesTen Data Manager driver for most application development and for all deployment.
- *Debug*: 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.

WINDOWS

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.

Table 1–1 lists the ODBC drivers for Windows:

Table 1–1 ODBC drivers provided for Windows platforms

Platform	Version	Name
Windows	Production	TimesTen Data Manager 11.2.2 Driver.
Windows	Debug	TimesTen Data Manager 11.2.2 Debug Driver.
Windows	Client	TimesTen Client 11.2.2 Driver

On UNIX, depending on the options selected at install time, TimesTen may install the Client driver and both the production version and the debug version of the TimesTen Data Manager ODBC driver.

Table 1–2 lists the TimesTen ODBC drivers for UNIX platforms.

Table 1–2 ODBC drivers provided for UNIX platforms

Platform	Version	Location and name
Solaris Linux	Production	`install_dir/lib/libtten.so` TimesTen Data Manager 11.2.2 Driver.
Solaris Linux	Debug	`install_dir/lib/libttenD.so` TimesTen Data Manager 11.2.2 Debug Driver.
Solaris Linux	Client	`install_dir/lib/libttclient.so` TimesTen Client 11.2.2 Driver.
AIX	Production	`install_dir/lib/libtten.a` TimesTen Data Manager 11.2.2 Driver.
AIX	Debug	`install_dir/lib/libttenD.a` TimesTen Data Manager 11.2.2 Debug Driver.
AIX	Client	`install_dir/lib/libttclient.a` TimesTen Client 11.2.2 Driver.

Connecting using the TimesTen JDBC driver and driver manager

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

As shown in Figure 1–1, the TimesTen JDBC driver uses the ODBC driver to access TimesTen databases. 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 database operations, the first step in using JDBC is to define a TimesTen database and the ODBC driver that accesses it on behalf of JDBC.

The TimesTen JDBC API is implemented using native methods to bridge to the TimesTen native API and provides a driver manager that can support multiple drivers connecting to separate databases. The JDBC driver manager in the `DriverManager` class keeps track of all 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 Client JDBC driver and the TimesTen direct driver can be loaded by an application. Then, Java applications can access databases either on the local system or a remote system.

For a list of the Java functions supported by TimesTen, see "Support for interfaces in the java.sql package" in the *Oracle TimesTen In-Memory Database Java Developer's Guide*.

Specifying Data Source Names to identify TimesTen databases

When you connect from an application, you use a Data Source Name (DSN) to uniquely identify the particular TimesTen database to which you want to connect. Specifically, a DSN is a character-string name that identifies a TimesTen database and a collection of connection attributes that are to be used when connecting to the database. On Windows, the DSN also specifies the ODBC driver to be used to access the database.

You can also define a default DSN that can be used when a user or an application either does not specify a DSN or specifies a DSN that is not defined in the `odbc.ini` file at connect time. For details, see "Setting up a default DSN" on page 1-16.

Note: 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 "Connection Attributes" in the *Oracle TimesTen In-Memory Database Reference.*

Even though the DSN uniquely identifies a TimesTen database, a database can be referenced by multiple DSNs. The difference between each of these unique DSNs is in the specification of the connection attributes to the database. This provides convenient names to different connection configurations for a single database.

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 case insensitive.
- It is composed of ASCII characters except for the following: ( ) [ ] { } , ; ? * = ! @ \ /
- TimesTen does not recommend the use of spaces as part of the DSN. If a DSN contains a space, some TimesTen utilities truncate the DSN at the point where they encounter the space. In addition, a DSN cannot start or end with a space, or consist solely of spaces.

The following sections describe how to configure and manage your DSNs:

- Overview of user and system DSNs
- Defining DSNs for direct or client/server connections
- Connection attributes for Data Manager DSNs or server DSNs

Overview of user and system DSNs

DSNs are resolved using a two-tiered naming system, where TimesTen first tries to resolve the DSN within the defined user DSNs and secondly within the defined 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, define user DSNs in the user `odbc.ini` file. TimesTen locates this file by first finding if a file is specified by the `ODBCINI` environment variable. If not, TimesTen locates the *`$HOME`*`/.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 database can be referenced by other users' user DSNs or by system DSNs.

TimesTen supports data sources for both the TimesTen Data Manager and the TimesTen Client in the `.odbc.ini` file.

- A **system** DSN can be used by any user on the system on which the system DSN is defined to connect to the TimesTen database.
  - 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 `sys.odbc.ini` file, which is referred to as the system `odbc.ini` file. TimesTen locates the system DSN file in the following order:
    - The file is located if it is specified by the `SYSODBCINI` environment variable.
    - In a non-root installation, the file is located in *`install_dir`*`/info/sys.odbc.ini`. In a root installation, the file is located in `/var/TimesTen/InstanceName/sys.odbc.ini`.
    - If not found in either the root or non-root locations, TimesTen looks for the `/var/TimesTen/sys.odbc.ini` file.
    - If not found in any of these locations, TimesTen looks on the system for the `/etc/odbc.ini` file.

Defining DSNs for direct or client/server connections

DSNs are created to uniquely identify a database, whether local or remote. The following explains the type of DSN to use for either a direct or client/server connection:

- *Data Manager DSN*: A DSN that specifies a local database uses the TimesTen Data Manager ODBC driver, which is the direct driver. You can use either the production version or the debug version of the TimesTen Data Manager driver.

  A Data Manager DSN refers to a database using a path name and a file name prefix. The database path name specifies the directory location of the database and the prefix for the database, such as `C:\data\chns\AdminDS` or `/home/chns/AdminDS`.

  Note: This path name and prefix does not define a file name, but the name of the directory where the database is located and the prefix for all database files. The actual files used by the database append file suffixes, such as `C:\data\chns\AdminDS.ds0` or `/home/chns/AdminDS.log2`.

  A Data Manager DSN that refers to a given TimesTen database must be defined on the same system on which the database resides. TimesTen creates *`dsName`*`.resn` files for each database. These files are used internally by TimesTen for maintaining logs.

  If multiple Data Manager DSNs refer to the same database, they must all use exactly the same database path name, even if some other path name identifies the same location. For example, you cannot use a symbolic link to refer to the database in one DSN and the actual path name in another DSN. On Windows, you cannot use a mapped drive letter in the database path name.

- *Client DSN*: A client DSN specifies a remote database and uses the TimesTen client. A client DSN refers to a TimesTen database indirectly by specifying a `hostname`, `DSN` pair, where the hostname represents the server system on which TimesTen Server is running and the DSN refers to a server DSN that specifies the TimesTen database on the server host.
- *Server DSN*: A server DSN is always defined as a system DSN and is defined on the server system for each database on that server that can be accessed by client applications. The format and attributes of a server DSN are very similar to those of a Data Manager DSN.

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 system where the database resides.
Client DSN	Can be a user or system DSN	Located on any local or remote system.
Server DSN	Must be a system DSN	Located on the system where the database resides.

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 or server DSNs

There are four types of TimesTen Data Manager DSN or server DSN attributes:

Note: For a complete description of all attributes, see "Connection Attributes" in the *Oracle TimesTen In-Memory Database Reference*.

- **Data Store attributes** are associated with a database when it is created and cannot be modified by subsequent connections. They can only be changed by destroying and re-creating the database.

  The following are the most commonly used data store attributes:

  - `DataStore`: Directory name and file name prefix of the database.
  - `LogDir`: Directory name of the database transaction log files. By default, the transaction log files reside in the checkpoint files directory. Placing the transaction log files and checkpoint files on different disks can improve system throughput.
  - `DatabaseCharacterSet`: Required character set specification that defines the storage encoding.

- **First connection attributes** are used 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 database is closed. First connection attributes can be modified only when the TimesTen database is unloaded and then the instance administrator reconnects with different values for the first connection attributes.

  The following are the most commonly used first connection attributes:

  – `PermSize`: Configures the allocated size of the database's permanent memory region. The permanent memory region contains persistent database elements. TimesTen only writes the permanent memory region to disk during a checkpoint operation.

  – `TempSize`: Configures the allocated size of the database's temporary memory region. The temporary memory region contains transient data generated when executing statements.

  Note: Your system must have sufficient main memory to accommodate the entire database. For more details on setting region sizes, see "Specifying the size of a database" on page 1-29.

- **General connection attributes** are set by each connection and persist for the duration of the connection. Each concurrent connection can have different values.

  The following are the most commonly used general connection attributes:

  – `UID`: Specifies the user name to be used for the connection to the database, whether using a direct or client/server connection. To connect as the instance administrator or as an external user, you do not need to specify an user name. When you do not specify an user name, TimesTen assumes that the `UID` is the user name identified by the operating system.

  – `PWD`: Specifies the password that corresponds with the specified `UID`. For internal users, if you do not set the `PWD` attribute in the `odbc.ini` file for the specified DSN or in the connection string, TimesTen prompts for the password. For external users, you do not provide the password as it is verified by the operating system.

    When you initiate a client/server connection, the password sent for the connection is encrypted by the client/server protocol.

  Note: For more information on the `UID` and `PWD` general connection attributes, see "UID and PWD" in the *Oracle TimesTen In-Memory Database Reference*. See "Managing users to control authentication" on page 4-1 for details on the instance administrator, external users, or internal users.

  – `PWDCrypt`: Specifies the encrypted password that corresponds with the specified `UID`.

Note: If you provide connection attributes in the connection string, this overrides the connection attributes set in the DSN. See "Connecting to a database using a connection string" on page 1-24 for details.

- **TimesTen Cache attributes** enable you to enter the Oracle Service Identifier for the Oracle database instance from which data is 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.

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

On UNIX, you specify attributes in the `odbc.ini` file. Attributes that do not appear in the `odbc.ini` file assume their default value.

Defining a Data Manager DSN

The following sections describe how to create a Data Manager DSN on either platform:

- Creating a Data Manager DSN on Windows
- Creating a Data Manager DSN on UNIX

Creating a Data Manager DSN on Windows

The following sections describe how to create a DSN on Windows:

- Specify the ODBC driver
- Specify the Data Manager DSN
- Specify the connection attributes

Note: For additional examples of setting up a Data Manager DSN, see "DSN examples" on page 1-16.

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 "Connecting using the TimesTen JDBC driver and driver manager" on page 1-4.

---

1. On the Windows Desktop from the **Start** menu, select **Settings**, **Control Panel**, **Administrative Tools**, and then select **Data Sources (ODBC)**. This opens the ODBC Data Source Administrator.

Note: If you are using a 32-bit TimesTen installation on a 64-bit Windows system, start the 32-bit ODBC Data Source Administrator with the following executable:

```
C:\WINDOWS\SysWOW64\odbcad32.exe
```

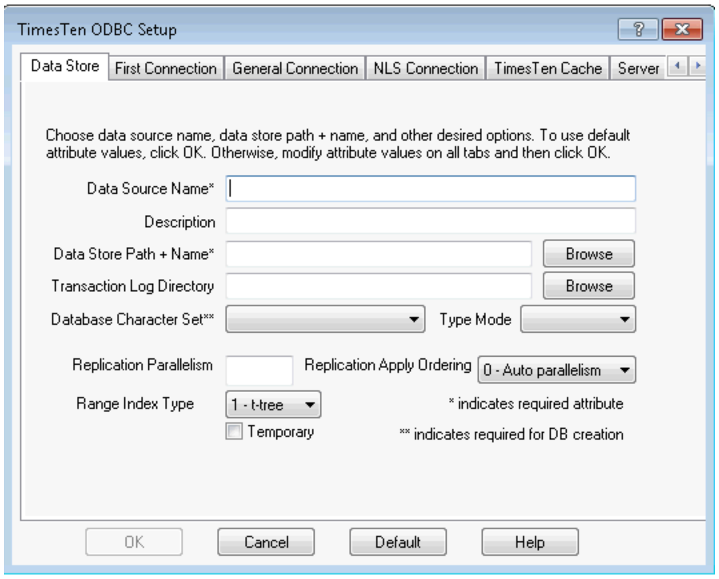
2. Choose whether you want to create a **User DSN** or **System DSN**. For a description of user and system DSNs, see "Overview of user and system DSNs" on page 1-5.
3. Perform one of the following:
   - Select an existing TimesTen data source and click **Configure**.
   - Click **Add**. Then, select the appropriate TimesTen driver from the list. Click **Finish**. This displays the TimesTen ODBC Setup dialog.

Note: For a list of TimesTen ODBC drivers, see "Connecting using TimesTen ODBC drivers" on page 1-3.

Specify the Data Manager DSN

On the Data Store tab of the TimesTen ODBC Setup dialog, specify a data source name (DSN), a database directory path and prefix, and a database character set. The database directory path cannot reference a mapped drive. See Figure 1–2.

Figure 1–2 Data Store tab



TimesTen ODBC Setup

Data Store | First Connection | General Connection | NLS Connection | TimesTen 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*

Description

Data Store Path + Name*

Transaction Log Directory

Database Character Set**

Type Mode

Replication Parallelism

Replication Apply Ordering: 0 - Auto parallelism

Range Index Type: 1 - t-tree

Temporary

* indicates required attribute

** indicates required for DB creation

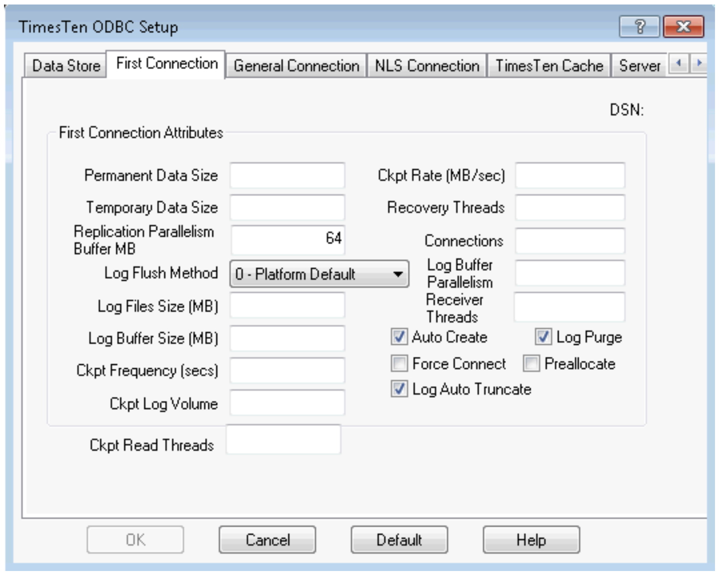
For an explanation of the DSN, database path and prefix, see "Specifying Data Source Names to identify TimesTen databases" on page 1-4. For an explanation of database character sets, see "Choosing a database character set" on page 5-2. The description field is optional.

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 as shown in Figure 1–3, Figure 1–4,and Figure 1–5. In addition, if you are using TimesTen Cache for an Oracle database, specify the connection attributes shown in Figure 1–6. If you are using a multi-threaded client/server configuration, specify the connection attributes shown in Figure 1–7.

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

Figure 1–3 First Connection Attributes


TimesTen ODBC Setup

Tabs: Data Store | First Connection | General Connection | NLS Connection | TimesTen Cache | Server

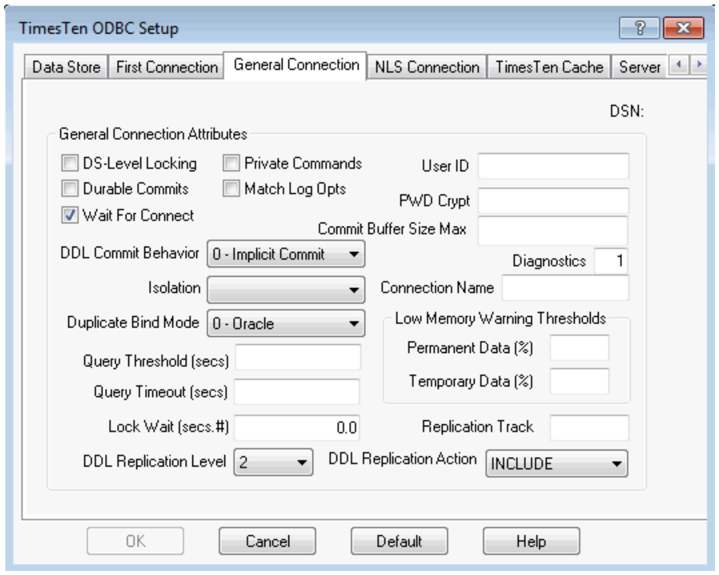
DSN:

First Connection Attributes

- Permanent Data Size:
- Temporary Data Size:
- Replication Parallelism Buffer MB: 64
- 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
- Ckpt Read Threads:

Buttons: OK | Cancel | Default | Help

Figure 1–4 General Connection Attributes


TimesTen ODBC Setup

Tabs: Data Store | First Connection | General Connection | NLS Connection | TimesTen Cache | Server

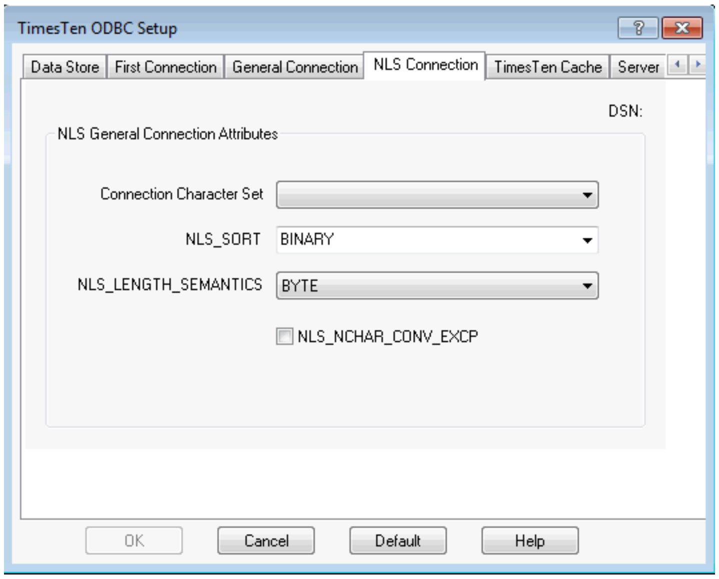
DSN:

General Connection Attributes

- [ ] DS-Level Locking
- [ ] Private Commands
- [ ] Durable Commits
- [ ] Match Log Opts
- [✓] Wait For Connect
- User ID:
- PWD Crypt:
- Commit Buffer Size Max:
- DDL Commit Behavior: 0 - Implicit Commit
- Diagnostics: 1
- Isolation:
- Connection Name:
- Duplicate Bind Mode: 0 - Oracle
- Low Memory Warning Thresholds
  - Permanent Data (%):
  - Temporary Data (%):
- Query Threshold (secs):
- Query Timeout (secs):
- Lock Wait (secs.#): 0.0
- Replication Track:
- DDL Replication Level: 2
- DDL Replication Action: INCLUDE

Buttons: OK | Cancel | Default | Help

Figure 1–5 NLS Connection Attributes



DSN:

NLS General Connection Attributes

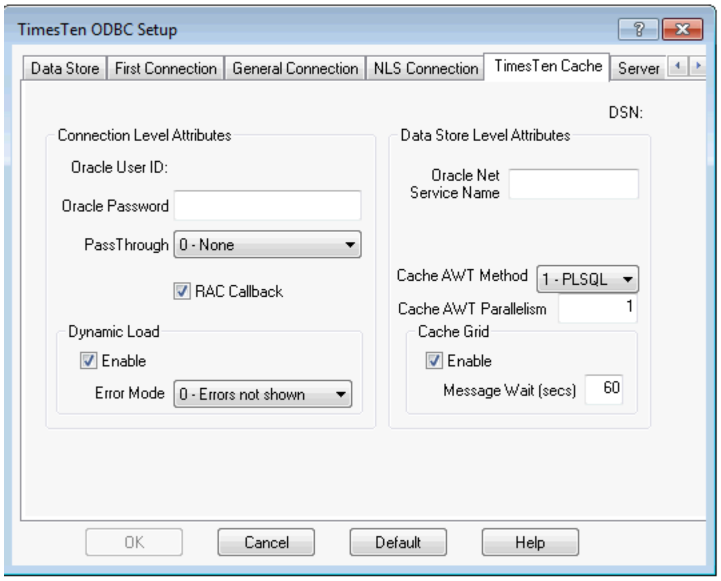
Connection Character Set

NLS_SORT BINARY

NLS_LENGTH_SEMANTICS BYTE

☐ NLS_NCHAR_CONV_EXCP

Figure 1–6 TimesTen Cache Attributes



DSN:

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 AWT Method 1 - PLSQL

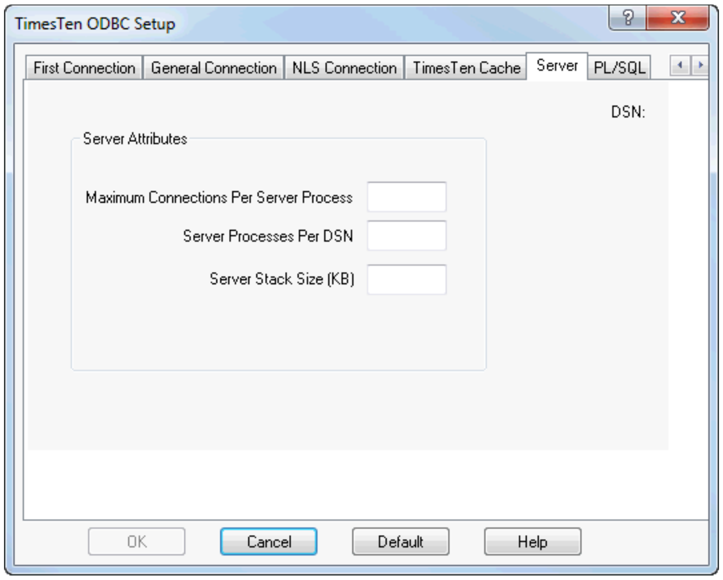
Cache AWT Parallelism 1

Cache Grid

☑ Enable

Message Wait (secs) 60

Figure 1–7 Server Attributes



TimesTen ODBC Setup

First Connection | General Connection | NLS Connection | TimesTen Cache | Server | PL/SQL

DSN:

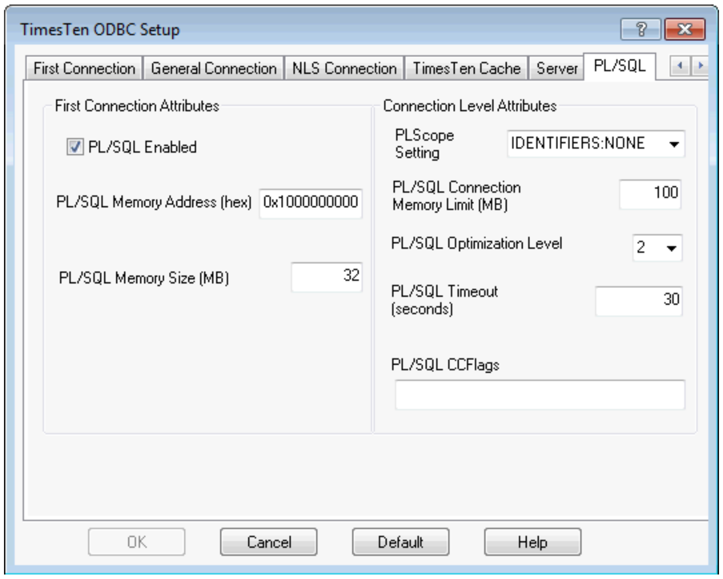
Server Attributes

Maximum Connections Per Server Process

Server Processes Per DSN

Server Stack Size (KB)

Figure 1–8 PL/SQL Attributes



TimesTen ODBC Setup

First Connection | General Connection | NLS Connection | TimesTen Cache | Server | PL/SQL

First Connection Attributes

[✓] PL/SQL Enabled

PL/SQL Memory Address (hex) 0x1000000000

PL/SQL Memory Size (MB) 32

Connection Level Attributes

PLScope Setting IDENTIFIERS:NONE

PL/SQL Connection Memory Limit (MB) 100

PL/SQL Optimization Level 2

PL/SQL Timeout (seconds) 30

PL/SQL CCFlags

Click **OK** when finished.

Creating a Data Manager DSN on UNIX

This section includes the following topics:

- Create a user or system odbc.ini file
- Using environment variables in database path names

Note: For examples on defining a DSN, see "DSN examples" on page 1-16.

Create a user or system 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 system `odbc.ini` file, which is located in *`install_dir`*`/info/sys.odbc.ini`.

The syntax for user and system `odbc.ini` files are the same. The syntax is described in "odbc.ini file entry descriptions" on page 1-22. The system `odbc.ini` file is created when TimesTen is installed on the system. Users must create their own user `odbc.ini` file.

Perform the following to create the DSN:

1. Specify the DSN 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]
```

2. Specify the ODBC driver.

> **Note:** JDBC users need to specify the ODBC driver to be used by the JDBC driver, as described in "Connecting using the TimesTen JDBC driver and driver manager" on page 1-4.

To set the TimesTen driver, specify the `DRIVER` attribute in the `odbc.ini` file. The following example provides the TimesTen ODBC driver that this DSN is configured to use:

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

> **Note:** For a list of TimesTen ODBC drivers that you can use, see Table 1–2.

3. Specify the database directory path and prefix in the `odbc.ini` file. The following example defines `/users/robin` as the database directory path and `FixedDs` as the prefix for the database files:

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

> **Note:** For more information, see "Specifying Data Source Names to identify TimesTen databases" on page 1-4.

The database directory path can use environment variables, as discussed in "Using environment variables in database path names" on page 1-15.

4. Choose a database character set. The following example defines the database character set in the `odbc.ini` file as `US7ASCII`:

```
DatabaseCharacterSet=US7ASCII
```

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

5. Set connection attributes in your `odbc.ini` file. Attributes that do not appear in the `odbc.ini` file assume their default value.

Note: See "Connection Attributes" in *Oracle TimesTen In-Memory Database Reference*. For examples, see "DSN examples" on page 1-16.

Using environment variables in database path names

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

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

Note: Environment variable expansion uses the environment of the process connecting to the database. Different processes may have different values for the same environment variables and may therefore expand the database 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.

Defining client and server DSNs

For directions on how to define client or server DSNs for each platform, see "Defining server DSNs on a TimesTen Server system" on page 2-7 and "Creating client DSNs on a TimesTen Client system" on page 2-12.

Resolution path for a DSN

When resolving a specific DSN, TimesTen performs the following:

Note:

- If a user DSN and a system DSN with the same name exist, TimesTen retrieves the user DSN.
- On UNIX, if there are multiple DSNs with the same name in the same `odbc.ini` file, TimesTen retrieves the first one specified in the file.

---

1. Searches for a user DSN with the specified name in the following files:
   a. The file referenced by the `ODBCINI` environment variable, if it is set.
   b. The `.odbc.ini` file in the user's home directory, if the `ODBCINI` environment variable is not set.
2. If no matching user DSN is found, TimesTen looks for a system DSN with the specified name.
   a. The file referenced by the `SYSODBCINI` environment variable, if it is set.
   b. The `sys.odbc.ini` file in the daemon home directory, if the `SYSODBCINI` environment variable is not set.

c. On UNIX, for a non-root installation, the file is located in `install_dir/info/sys.odbc.ini`. Or for a root installation, the file is located at `/var/TimesTen/InstanceName/sys.odbc.ini` or `/var/TimesTen/sys.odbc.ini`.

DSN examples

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

- Setting up a default DSN
- Setting up a temporary database
- Specifying PL/SQL connection attributes in a DSN
- Creating multiple DSNs to a single database

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

Setting up a default DSN

Optionally, you can add a default data source definition. At connect time, if an application specifies a DSN that is not in the `odbc.ini` file or if the application does not specify a DSN, the default DSN is used to configure the connection to the TimesTen database.

The default data source must be named `default` when defined. The default DSN can be defined with the same attributes as any other data source, which are described in "DSN specification" on page 1-22.

Note: On Windows platforms, when using the "ODBC Data Source Administrator" to create the default DSN, you can specify only one default DSN in each of the User DSN, System DSN, and File DSN tabs. The default DSN must be associated with an ODBC driver.

When connecting, TimesTen uses the default DSN in any of the following scenarios:

- When you specify the `DSN=default` keyword-value pair in the connection string.
- When you specify an undefined value for the `DSN` connection attribute in the connection string.
- When you do not specify any value for the `DSN` connection attribute in the connection string.

However, in general, it is best to connect with a specific data source.

When using a default DSN, provide `default` as the DSN name when performing TimesTen utility operations that require a DSN name, such as destroying the database with the `ttDestroy` utility.

The following example shows the user invoking `ttIsql` to connect using an undefined DSN. Since there is no definition for `doesNotExist` in the `odbc.ini` file, TimesTen uses the `default` DSN to create the database and initiate a connection to it. It also demonstrates the user invoking both the `ttStatus` and `ttDestroy` utilities with `default` specified as the DSN. In addition, it shows the error thrown if the user provides `doesNotExist` as the DSN, instead of `default`.

```
$ ttIsql doesNotExist;
```

```
Copyright (c) 1996-2011, Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

connect "DSN=doesNotExist";
Connection successful:
Command> exit
Disconnecting...
Done.
]$ ttStatus default
TimesTen status report as of Mon Oct 22 12:27:52 2012

Daemon pid 13623 port 16138 instance myhost
TimesTen server pid 13632 started on port 16140
------------------------------------------------------------------------
Data store /timesten/install/info/default
There are no connections to the data store
Replication policy  : Manual
Cache Agent policy  : Manual
PL/SQL enabled.
------------------------------------------------------------------------
Accessible by group xyz
End of report
$ ttDestroy doesNotExist;
Failed to destroy data store: Specified DSN is not found in user and system
odbc.ini files (or registry)
$ ttDestroy default;
```

The following example shows how to configure connection attributes for a default DSN. While not necessary, you can configure connection attributes for a default DSN as you would configure any other DSN. Notice that it is not specified in the *ODBC Data Sources* section.

```
[ODBC Data Sources]
sampledb_1122=TimesTen 11.2.2 Driver
...

[default]
Driver=install_dir/lib/libtten.so
DataStore=install_dir/info/DemoDataStore/default
PermSize=40
TempSize=64
DatabaseCharacterSet=US7ASCII

[sampledb_1122]
Driver=install_dir/lib/libtten.so
DataStore=install_dir/info/DemoDataStore/sampledb_1122
PermSize=40
TempSize=32
PLSQL=1
DatabaseCharacterSet=US7ASCII
```

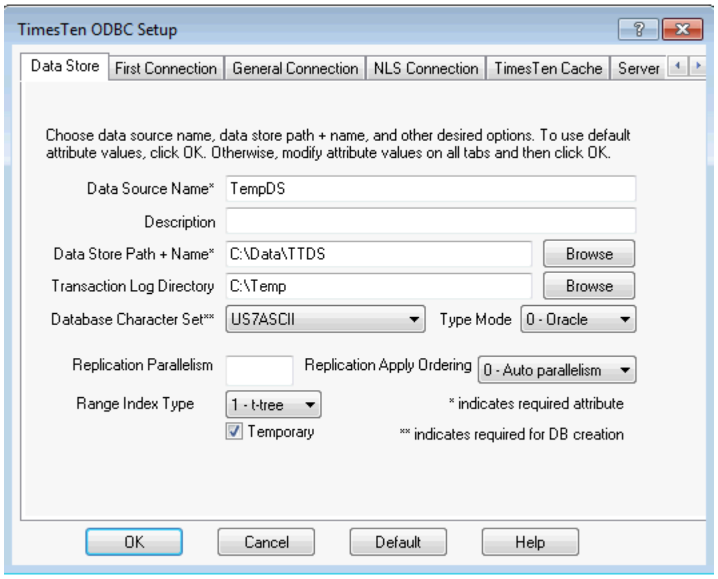
Setting up a temporary database

This example illustrates how to set up a temporary database. For information on temporary databases, see "Database overview" on page 8-1.

WINDOWS

On Windows, you can use the settings in the TimesTen ODBC Setup dialog to set up a temporary database. See Figure 1–9 and Figure 1–10.

Figure 1–9 Data Store tab

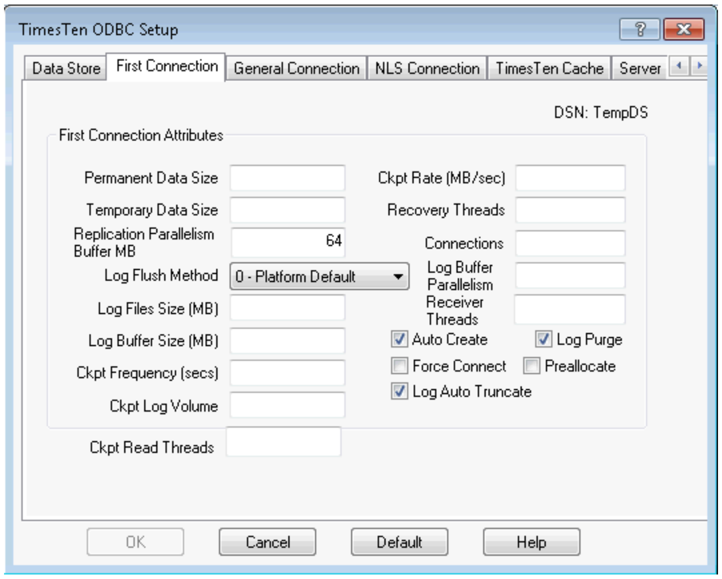


TimesTen ODBC Setup

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:\Data\TTDS
Transaction Log Directory C:\Temp
Database Character Set** US7ASCII
Type Mode 0 - Oracle
Replication Parallelism
Replication Apply Ordering 0 - Auto parallelism
Range Index Type 1 - t-tree
[✓] Temporary
* indicates required attribute
** indicates required for DB creation

Figure 1–10 First Connection Attributes



TimesTen ODBC Setup

DSN: TempDS

First Connection Attributes

Permanent Data Size
Temporary Data Size
Replication Parallelism Buffer MB 64
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
Ckpt Read Threads

UNIX

To set up a temporary database on UNIX, create the following entries in your `odbc.ini` file. For a list of drivers for all UNIX platforms, see the table in "Connecting using TimesTen ODBC drivers" on page 1-3.

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 database
Temporary=1
#create database if it is not found
AutoCreate=1
#log database updates to disk
LogPurge=1
DatabaseCharacterSet=US7ASCII
```

Specifying PL/SQL connection attributes in a DSN

You can specify values for PL/SQL general connection attributes.

Note: For a complete list of PL/SQL connection attributes, see "Connection Attributes" in the *Oracle TimesTen In-Memory Database Reference*.

The following are some 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 by setting PLSQL to "1" and sets the PL/SQL shared memory segment size to 32 MB.

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

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 database

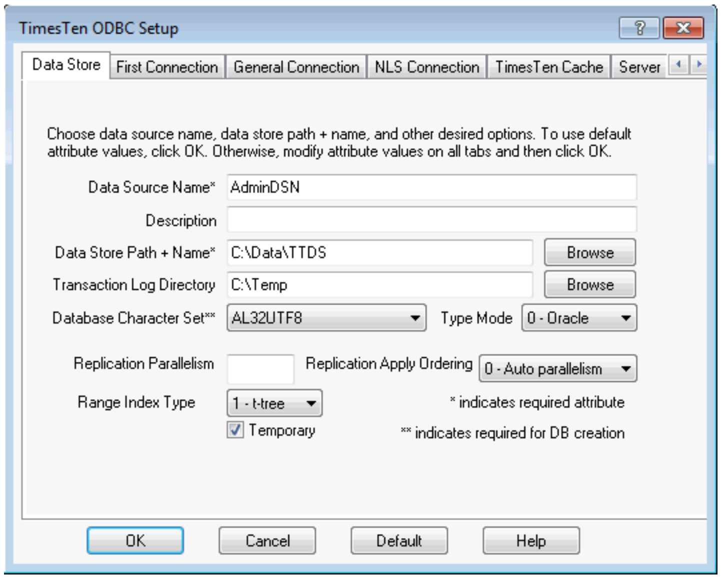
You can create two or more DSNs that refer to the same database but have different connection attributes.

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` database by using `AdminDSN`. Applications that use multibyte characters can connect to the `TTDS` database by using `GlobalDSN`.

WINDOWS

For Windows, use the ODBC Data Source Administrator to define AdminDSN as shown in Figure 1–11. `AdminDSN` is created with the `AL32UTF8` database character set. Figure 1–12 shows that `US7ASCII` is the connection character set for `AdminDSN`.

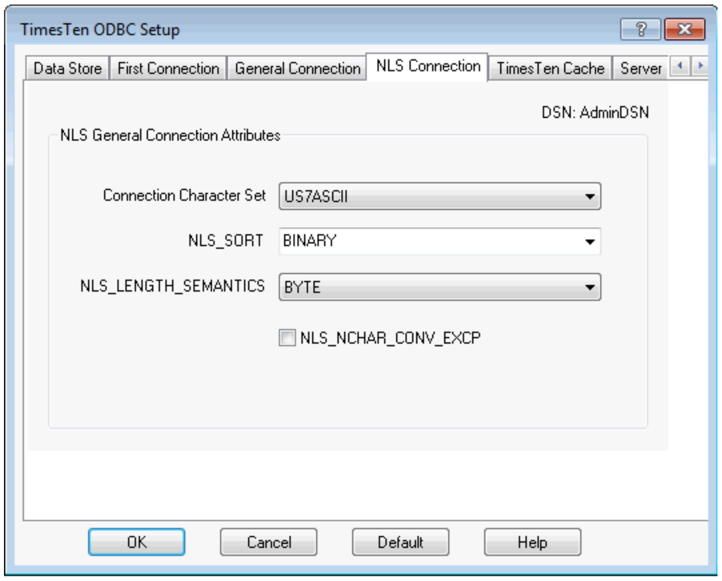
Figure 1–11 Creating AdminDSN using TTDS database



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
Replication Parallelism
Replication Apply Ordering 0 - Auto parallelism
Range Index Type 1 - t-tree
[✓] Temporary
* indicates required attribute
** indicates required for DB creation

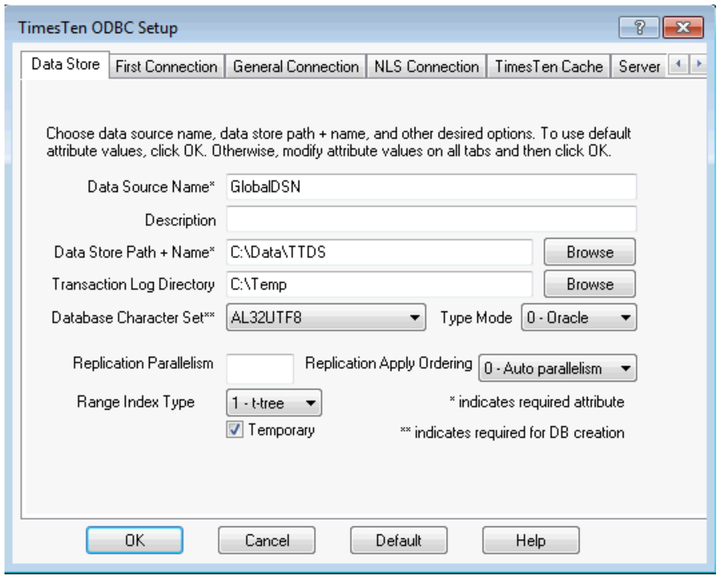
Figure 1–12 Setting the connection character set for AdminDSN



DSN: AdminDSN
NLS General Connection Attributes
Connection Character Set US7ASCII
NLS_SORT BINARY
NLS_LENGTH_SEMANTICS BYTE
[ ] NLS_NCHAR_CONV_EXCP

`GlobalDSN` is also created with the `AL32UTF8` database character set, as shown in Figure 1–13. Figure 1–14 shows that the connection character set for `GlobalDSN` is `AL32UTF8`.

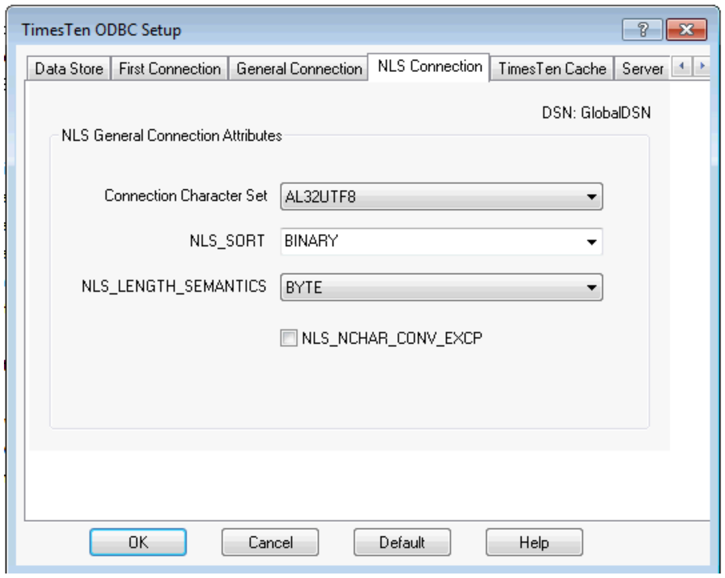
Figure 1–13 Creating GlobalDSN using TTDS database



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
Replication Parallelism
Replication Apply Ordering 0 - Auto parallelism
Range Index Type 1 - t-tree
[✓] Temporary
* indicates required attribute
** indicates required for DB creation

Figure 1–14 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
```

odbc.ini file entry descriptions

The following sections describe the entries in the `odbc.ini` file:

- ODBC Data Sources
- DSN specification
- odbc.ini file example

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]
DSN=driver-description
```

- The *`DSN`* is required and 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.

The optional Data Sources section, when present in the system DSN file on the TimesTen Server, is used during the setup of client DSNs. All system DSNs are made available to the client DSN setup for the ODBC Data Source Administrator on the client, which displays all available DSNs on the TimesTen Server. The user can always add a new system DSN in the ODBC Data Source Administrator. When adding DSNs to the system DSN file, you should only include those DSNs that can be advertised to clients. All system DSNs are potentially accessible through the client/server configuration, even if they are not advertised.

DSN specification

Each DSN listed in the ODBC Data Sources section has its own DSN specification. The DSN specification for Data Manager DSN has the format shown in Table 1–3.

Table 1–3 Data Source specification format

Component	Description
`[`*`DSN`*`]`	The *`DSN`* is required. It is the name of the DSN, 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-prefix`*	The directory path and prefix of the database to access. This is required.
`DatabaseCharacterSet=`*`Database-character-set`*	The database character set determines the character set in which data is stored. The database character set is required and cannot be altered after the database has been created. For more information, see "Choosing a database character set" on page 5-2.

Table 1–3 (Cont.) Data Source specification format

Component	Description
Optional attributes	See "Connection Attributes" in *Oracle TimesTen In-Memory Database Reference* for information about attributes.

For example, the `sampledb_1122` DSN could have a specification that includes the following:

```
[sampledb_1122]
Driver=install_dir/lib/libtten.so
DataStore=install_dir/info/DemoDataStore/sampledb_1122
...
```

The database specification for TimesTen client DSN has the format shown in Table 1–4.

Note: While the syntax for the TimesTen client DSN is listed here, full directions for setting the client DSN and server DSN are located in "Defining server DSNs on a TimesTen Server system" on page 2-7 and "Creating client DSNs on a TimesTen Client system" on page 2-12.

Table 1–4 Database specification for TimesTen Client configurations

Component	Description
`[DSN]`	The *DSN* is required. It is the same DSN specified in the ODBC Data Sources section of the `.odbc.ini` file.
`TTC_Server=server-name`	The *server-name* is required. It is the DNS name, host name, IP address or logical server 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 databases.

For example, the client/server data source `sampledbCS_1122` that connects to `sampledb_1122` on the TimesTen Server `ttserver` could have a data source specification that includes the following:

```
[sampledbCS_1122]
TTC_Server=ttserver
TTC_SERVER_DSN=sampledb_1122
TTC_Timeout=30
```

odbc.ini file example

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

```
...
[ODBC Data Sources]
sampledb_1122=TimesTen 11.2.2 Driver
...
```

```
[sampledb_1122]
Driver=install_dir/lib/libtten.so
DataStore=install_dir/info/DemoDataStore/sampledb_1122
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 11.2.2 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_1122=TimesTen 11.2.2 Client Driver
...

[sampledbCS_1122]
TTC_SERVER=localhost
TTC_SERVER_DSN=sampledb_1122
...
```

Connecting to a database using a connection string

TimesTen applications require a DSN or a connection string be specified to connect to a database. For modularity and maintainability, it is better to set attributes in a DSN rather than in a connection string within the application, unless a particular connection requires that specific attribute settings override the settings in the DSN or the default settings.

The syntax for a connection string contains connection attribute definitions, where each attribute is separated by a semicolon.

These precedence rules are used to determine the settings of DSN attributes:

1. Attribute settings specified in a connection string have the highest precedence.If an attribute appears more than once in a connection string, the first specification is used.
2. If an attribute is not specified in the connection string, the attribute settings that are specified in the DSN are used.
3. Default attribute settings have the lowest precedence.

You can connect to a TimesTen database without a predefined DSN with any ODBC application or the `ttIsql` utility if the connection string contains the `Driver`, `DataStore`, and `DatabaseCharacterSet` attributes. Define the connection string as follows:

- The name or path name of the ODBC driver using the `Driver` attribute.
  - On Windows, the value of the `Driver` attribute should be the name of the TimesTen ODBC Driver. For example, the value can be TimesTen Data Manager 11.2.2.
  - 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.

- The database path and file name prefix using the `DataStore` attribute.
- The character set for the database using the `DatabaseCharacterSet` attribute.

The following example shows how you can connect providing the `Driver`, `DataStore`, and `DatabaseCharacterSet` attributes using a connection string in the `ttIsql` utility:

```
C:\ ttIsql
Copyright (c) 1996-2011, Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.
Command> connect "Driver=TimesTen Data Manager
11.2.2;DataStore=C:\sales\admin;DatabaseCharacterSet=US7ASCII";
```

Specifying a RAM policy

TimesTen allows you to specify a RAM policy that determines when databases are loaded and unloaded from main memory. For each database, you can have a different RAM policy.

The RAM policy options are as follows:

- *inUse*: The database is loaded into memory when the first connection to the database is opened, and it remains in memory as long as it has at least one active connection. When the last connection to the database is closed, the database is unloaded from memory. This is the default policy.
- *inUse with RamGrace*: The database is loaded into memory when the first connection to the database is opened, and it remains in memory as long as it has at least one active connection. When the last connection to the database is closed, the database remains in memory for a "grace period." The database is unloaded from memory only if no processes have connected to the database for the duration of the grace period. The grace period can be set or reset at any time. It stays in effect until the next time the grace period is changed.
- *always*: The database always stays in memory. If the TimesTen daemon is restarted, it automatically reloads the database. The database is always automatically reloaded unless an unrecoverable error condition occurs. For more details on database error recovery, see "Changes to RAM policy after automatic recovery fails" on page 1-26.
- *manual*: The database is manually loaded and unloaded by system administrators. Once loaded, TimesTen ensures that the database stays loaded until the administrator unloads the database or unless an unrecoverable error condition occurs. For more details on database error recovery, see "Changes to RAM policy after automatic recovery fails" on page 1-26.

A system administrator can set the RAM policy or manually load or unload the database with either the `ttAdmin` utility or the C API RAM policy utilities. For more details, see "ttAdmin" in the *Oracle TimesTen In-Memory Database Reference* or the "TimesTen Utility API" chapter in the *Oracle TimesTen In-Memory Database C Developer's Guide*.

Note: By default, if an automatic recovery of the database is unsuccessful after a fatal error, TimesTen changes the `always` and `manual` RAM policies to `InUse` to prevent reoccurring failures. For more information on how to prevent the RAM policy from changing, see "Changes to RAM policy after automatic recovery fails" on page 1-26.

The following example sets the RAM policy to `always` for the database identified by the `ttdata` DSN:

Note: The first line shows the RAM residence policy set to `always`. The rest of the output details other policies you can set with the `ttAdmin` utility. See "ttAdmin" in the *Oracle TimesTen In-Memory Database Reference* for more information.

```
% ttadmin -rampolicy always ttdata
RAM Residence Policy            : always
Replication Agent Policy        : manual
Replication Manually Started    : False
Cache Agent Policy              : manual
Cache Agent Manually Started    : False
```

Changes to RAM policy after automatic recovery fails

If a fatal error invalidates the database and the automatic database recovery performed by TimesTen is unsuccessful, the following occurs by default:

- TimesTen changes the `always` and `manual` RAM policies to `InUse` to prevent reoccurring errors. For both the `always` and `manual` RAM policies, TimesTen automatically reloads the database after each failure, even if the same failure occurs multiple times. However, for the `InUse` RAM policy, TimesTen does not automatically reload the database until a process tries to connect.
- If a fatal error causes the database to be invalidated, user processes connected to this database may not know that the database has been invalidated. In this case, the invalidated database exists in memory until all user processes close their connections. Thus, the invalidated database could coexist in memory with a newly reloaded database.

Note: Reloading a large database into memory when an invalidated database still exists in memory can fill up available RAM. See "Preventing an automatic reload of the database after failure" on page 1-27 on how to stop automatic reloading of the database.

You can modify the default database recovery behavior by setting the `-enablePolicyInactive` option in the `ttendaemon.options` file before you start the TimesTen daemon. Once set, the following is the behavior when automatic recovery fails:

- The RAM policies of `manual` and `always` remain unchanged.
- The replication and cache agents are not restarted.
- After several failed attempts to reload the database, TimesTen sets the `policyInactive` mode, which prevents any more attempts at loading the database.

Any one of the following clears the `policyInactive` mode, so that normal behavior can resume:

- The TimesTen daemon restarts.
- A process connects successfully.
- The administrator executes a `ttAdmin` command for the database that changes the RAM policy, performs a RAM load, or starts either the cache or replication agents.

You can see if `policyInactive` mode is active when one of the following messages is displayed in the `ttStatus` output:

- `"Automatic reloads disabled."`
- `"Data store should be manually loaded, but inactive due to failures."` This is shown when the RAM policy is `manual`.
- `"Data store should be loaded, but inactive due to failures."` This is shown if the RAM policy is `always`.

Preventing an automatic reload of the database after failure

After a fatal error that causes the database to be invalidated, TimesTen attempts to reload and recover the database, as long as it is consistent with the settings for the RAM policy, cache agent policy, and replication agent policy. However, user processes could still be connected to the invalidated database if they do not know that the original database has been invalidated. In this case, the invalidated database exists in memory until all user processes close their connections. Thus, the invalidated database could coexist in memory with the newly reloaded database. This can be an issue if the database is large.

> **Note:** Not only does the RAM policy determines whether the database is reloaded and recovered, but the cache agent and replication agent policies also factor into whether the database is reloaded after invalidation. If the cache agent and replication agent policies are set so that the daemon automatically restarts the agent after a failure, the agent initiates a connection to the database. If this is the first connection, the daemon reloads the database and performs a recovery.
>
> For more information on cache agent and replication agent policies, see "Starting and stopping the replication agents" in the *Oracle TimesTen In-Memory Database Replication Guide*, "Set a cache agent start policy" in the *Oracle TimesTen Application-Tier Database Cache User's Guide*, and "ttAdmin" in the *Oracle TimesTen In-Memory Database Reference*.

You can prevent the database from being automatically reloaded after an invalidation using the `ttAdmin -noautoreload` command. You can reset to the default automatic database reload behavior with the `ttAdmin -autoreload` command. See "ttAdmin" in the *Oracle TimesTen In-Memory Database Reference* for more details.

> **Note:** The `ttRamPolicyAutoReloadSet` built-in procedure performs the same actions as `ttAdmin -noautoreload` and `ttAdmin -autoreload`. See "ttRamPolicyAutoReloadSet" in the *Oracle TimesTen In-Memory Database Reference* for more details.

Any one of the following initiates a reload and recovery of the database so that normal behavior can resume:

- The TimesTen daemon restarts.
- A process connects successfully.

- The administrator executes a `ttAdmin` command for the database that changes the RAM policy, performs a RAM load, or starts either the cache or replication agents.

If you set the behavior to prevent automatic reloads of the database, you may receive the following error when connecting to a database that was not reloaded.

```
Error 707, "Attempt to connect to a data store that has been manually unloaded
from RAM"
```

Loading and unloading the database from memory

Loading and unloading the database from memory is described in these sections:

- Loading the database into memory
- Unloading the database from memory

Loading the database into memory

Set the RAM policy to `manual` or `inUse` before loading the database into memory. If the RAM policy is set to `always`, then the database is already loaded since this policy states to keep the database loaded at all times. Also, ensure that the TimesTen daemon is running. The default RAM policy for a TimesTen database is `inUse`. For more information on specifying a RAM policy, see "Specifying a RAM policy" on page 1-25.

Before you try to load the database into memory, confirm that the TimesTen daemon is running:

```
ttDaemonAdmin -start
```

To load the database into memory, run the `ttAdmin` utility:

- The following example sets the RAM policy of a TimesTen database to `manual`:

  ```
  ttAdmin -ramPolicy manual sampledb_1122

  RAM Residence Policy            : manual
  Manually Loaded In RAM          : False
  Replication Agent Policy        : manual
  Replication Manually Started    : False
  Cache Agent Policy              : manual
  Cache Agent Manually Started    : False
  ```

  Then, load the TimesTen database into memory with the `ttAdmin -ramload` utility. The `-ramLoad` option of the `ttAdmin` utility can only be used with the manual RAM policy:

  ```
  ttAdmin -ramLoad sampledb_1122

  RAM Residence Policy            : manual
  Manually Loaded In RAM          : True
  Replication Agent Policy        : manual
  Replication Manually Started    : False
  Cache Agent Policy              : manual
  Cache Agent Manually Started    : False
  ```

- The following example defines the database RAM policy as `inUse` with a grace period of 200 seconds:

  ```
  ttAdmin -ramPolicy inUse -ramGrace 200 sampledb_1122

  RAM Residence Policy            : inUse plus grace period
  ```

```
    RAM Residence Grace (Secs)      : 200
    Replication Agent Policy        : manual
    Replication Manually Started    : False
    Cache Agent Policy              : manual
    Cache Agent Manually Started    : False
```

If your database is configured replication or cache for your database, run the `ttAdmin` utility to start the replication and cache agents.

```
ttAdmin -repStart sampledb_1122

ttAdmin -cacheStart sampledb_1122
```

For more information on these utilities, see "ttAdmin" and "ttDaemonAdmin" in the *Oracle TimesTen In-Memory Database Reference*.

Unloading the database from memory

Before unloading the database from memory, you must first close all active connections to the database and set the RAM policy of the database to `manual` or `inUse`.

- To close all active connections to the database, run the `ttStatus` utility to find processes connected to the database and stop them. For more information on `ttStatus`, see "ttStatus" in the *Oracle TimesTen In-Memory Database Reference*.

  If you configured cache or replication for your database, stop the cache and replication agents with the `-cacheStop` and `-repStop` options of the `ttAdmin` utility.
- To set the RAM policy to `manual` or `inUse`, see "Specifying a RAM policy" on page 1-25 for information.
- To unload a database with a `manual` RAM policy from memory, run the `ttAdmin -ramUnload` utility. TimesTen unloads a database with an `inUse` RAM policy from memory once you close all active connections. For more information on unloading a database with a `manual` RAM policy from memory, see "Unloading a database from memory" in the *Oracle TimesTen In-Memory Database Installation Guide*.

For more information on the `ttAdmin` utility, see "ttAdmin" in the *Oracle TimesTen In-Memory Database Reference*.

Specifying the size of a database

TimesTen manages database space using two separate memory regions within a single contiguous memory space. One region contains permanent data and the other contains temporary data.

- Permanent data includes the tables and indexes that make up a TimesTen database. When a database is loaded into memory, the contents of the permanent memory region are read from files stored on disk. The permanent memory region 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 memory region is created when a database is loaded into memory and is destroyed when it is unloaded.

The connection attributes that control the size of the database when it is in memory are `PermSize` and `TempSize`. The `PermSize` attribute specifies the size of the permanent memory region and the `TempSize` attribute specifies the size of the temporary memory region.

Note: See "Connection Attributes" in the *Oracle TimesTen In-Memory Database Reference* for further description of these attributes.

The sizes of the permanent and temporary memory regions are set when a database is loaded into memory and cannot be changed while the database is in memory. To change the size of either region, you must unload the database from memory and then reconnect using different values for the `PermSize` or `TempSize` attributes. For more information on unloading the database from memory, see "Unloading the database from memory" on page 1-29.

Managing the database size is described in these sections:

- Estimating and modifying the memory region sizes for the database
- Monitoring PermSize and TempSize attributes
- Receiving out-of-memory warnings

Estimating and modifying the memory region sizes for the database

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

- `PermSize` connection attribute: You can increase but not decrease the permanent memory region.
- `TempSize` connection attribute: You can increase or decrease the size of the temporary memory region for databases 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 database. In general, the minimum size of this shared memory segment should be:

```
PermSize + TempSize + LogBufMB + 64 MB 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 memory region. The amount of permanent memory region required by a stored PL/SQL unit depends on the size and complexity of the unit. Small procedures can take less than 3 KBs, while larger ones can take considerably more. On average, reasonably complex units could be expected to use about 20 KBs of permanent memory region space.

If the database is configured for replication, reconfigure the database sizes for all replicas of the database. Once you have made the change in database size, load the database into memory and restart the cache and replication agents.

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*.

Ensuring sufficient disk space

TimesTen saves a copy of the database in two checkpoint files, each of which is stored in the directory that is specified with the `DataStore` attribute. As each checkpoint file grows on disk, it never decreases in size. This can result in the size of each checkpoint file being equal to the maximum size that the database has ever reached in the permanent memory region. The maximum size for each checkpoint file is `PermSize` + the database header. For each permanent database, you must have enough disk space for both checkpoint files and all transaction log files. For more information about the `DataStore` attribute, see "DataStore" in the *Oracle TimesTen In-Memory Database Reference*.

You can set the `Preallocate` connection attribute to `1` to have TimesTen reserve disk space at connect time for checkpoint files. This is useful for big databases, to ensure that the disk always has room for the checkpoint files as data is added to the database. For more information about the `Preallocate` connection attribute, see "Preallocate" in the *Oracle TimesTen In-Memory Database Reference*.

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`, `TEMP_IN_USE_SIZE`, and `TEMP_IN_USE_HIGH_WATER`. Each of these columns show in KB units the currently allocated size of the database and the in-use size of the database. 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 database by calling the `ttBlockInfo` built-in procedure. For more details, see "ttBlockInfo" in the *Oracle TimesTen In-Memory Database Reference*.

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*.

Manage existing tables in the database

The following utilities enable you to manage certain aspects of existing tables in the database:

- 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 database.

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 database and that the file found is compatible with `ttBulkCp`.

- Rename the owner of tables in a database. Use the `ttMigrate` utility. When restoring tables, you can use the `-rename` option to rename the owner of tables.

Thread programming with TimesTen

TimesTen supports multithreaded application access to databases. When a connection is made to a database, 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 databases.

Defragmenting TimesTen databases

Under some circumstances, a TimesTen database may develop memory fragmentation such that significant amounts of free memory are allocated to partially filled pages of existing tables. This can result in an inability to allocate memory for other uses (such as new pages for other tables) due to a lack of free memory. In these circumstances, it is necessary to defragment the database in order to make this memory available for other uses.

A secondary table partition is created after a table has been altered with the `ALTER TABLE ADD` SQL statement. Defragmentation enables you to remove the secondary table partitions and create a single table partition that contains all of the table columns. When secondary table partitions have been created, it is recommended to periodically defragment the database in order to improve space utilization and performance.

The following procedures address both types of database fragmentation:

- Offline defragmentation of TimesTen databases
- Online defragmentation of TimesTen databases

Offline defragmentation of TimesTen databases

To defragment a database, use the `ttMigrate` utility as follows:

1. Stop all connections to the database.
2. Save a copy of the database using `ttMigrate`.

```
ttMigrate -c ttdb ttdb.dat
```

3. As the administration user, rebuild the `ttdb` database:

```
ttMigrate -r -relaxedUpgrade -connstr "dsn=ttdb" ttdb.dat
```

Note: If you do not want to condense table partitions, remove the `-relaxedUpgrade` option when executing the `ttMigrate -r` command.

At this time:

- All the users, cache groups, and the active standby pair have been restored to `ttdb`.
- The cache groups are in `AUTOREFRESH STATE = OFF`.
- The cache agent and replication agent are not running.

Table partitions can be added when columns are added to tables with the `ALTER TABLE ADD` SQL statement. See the notes on "ALTER TABLE" in the *Oracle TimesTen In-Memory Database SQL Reference* for more information.

For more information on `ttMigrate`, see "ttMigrate" in the *Oracle TimesTen In-Memory Database Reference*.

Online defragmentation of TimesTen databases

Use a combination of the `ttMigrate -relaxedUpgrade` and `ttRepAdmin -duplicate` utilities to defragment TimesTen databases (with minimal overall service downtime) that are involved in a replication scheme where `TABLE DEFINITION CHECKING` is set to `RELAXED`. In addition, the `ttMigrate -relaxedUpgrade` option condenses partitions.

Note: You can only defragment TimesTen databases that are involved in an active standby pair replication scheme if the replication scheme either does not contain any cache groups or contains only `READONLY` cache groups.

The following sections describe how to defragment TimesTen databases that are involved in a replication scheme:

Note: The examples provided in each section assume that you are familiar with the configuration and management of replication schemes. For more information, see "Getting Started" in the *Oracle TimesTen In-Memory Database Replication Guide*.

- Online defragmentation of databases in an active standby pair replication scheme
- Online defragmentation of databases in a non active standby pair replication scheme

Online defragmentation of databases in an active standby pair replication scheme

The following sections describe how to defragment TimesTen databases that are involved in an active standby pair replication scheme:

- Migrate and rebuild the standby database
- Reverse the active and standby roles

- Destroy and re-create the new standby

The example in this section shows how to perform an online defragmentation with an active standby pair replication scheme where the active database is `ttdb1` and the standby database is `ttdb2`.

Migrate and rebuild the standby database

The following shows how to stop replication to the standby TimesTen database, save a copy of the standby database, and then defragment the standby database.

Note: While the standby database is defragmented, application processing can continue on the active database.

Perform the following to save a copy of the standby database:

1. Stop the replication agent on the standby database (`ttdb2`):

   ```
   ttAdmin -repStop ttdb2
   ```

2. If there any subscribers, execute `ttRepStateSave` on the active database to set the status of the standby to failed. As long as the standby database is unavailable, updates to the active database are replicated directly to the subscriber databases.

   ```
   ttRepStateSave('FAILED', 'ttdb2', 'ttsrv2');
   ```

3. Save a copy of the standby database using `ttMigrate`.

   ```
   ttMigrate -c -relaxedUpgrade ttdb2 ttdb2.dat
   ```

4. Stop the cache agent, drop any cache groups, and destroy the standby.

   ```
   ttAdmin -cacheStop ttdb2
   ```

   While connected as cache manager user, drop all cache groups:

   ```
   Command> DROP CACHE GROUP t_cg;
   ```

   Destroy the standby database:

   ```
   ttDestroy ttdb2
   ```

5. Rebuild the standby database. Execute the following on the standby as the instance administrator:

   ```
   ttIsql ttdb2
   ```

6. Create the cache manager user and grant the user `ADMIN` privileges.

   ```
   Command> CREATE USER cacheadmin IDENTIFIED BY cadminpwd;
   Command> GRANT CREATE SESSION, CACHE_MANAGER, CREATE ANY TABLE,
    DROP ANY TABLE TO cacheadmin;
   Command> GRANT ADMIN TO cacheadmin;
   ```

Note: The cache manager user requires `ADMIN` privileges in order to execute `ttMigrate -r`. Once migration is completed, you can revoke the `ADMIN` privilege from this user if desired.

For more information on `ttMigrate`, see "ttMigrate" in the *Oracle TimesTen In-Memory Database Reference*.

---

7. As the cache manager user, rebuild the `ttdb2` database:

   ```
   ttMigrate -r -relaxedUpgrade -cacheuid cacheadmin -cachepwd cadminpwd -connstr
    "dsn=ttdb2;uid=cacheadmin;pwd=cadminpwd;oraclepwd=oraclepwd" ttdb2.dat
   ```

   At this time:

   - All the users, cache groups, and the active standby pair have been restored to `ttdb2`.
   - The cache groups are in `AUTOREFRESH STATE = OFF`.
   - The cache agent and replication agent are not running.

8. As the cache manager user, start the cache agent on the standby:

   ```
   ttAdmin -cacheStart ttdb2
   ```

9. Load any cache groups.

   ```
   Command> ALTER CACHE GROUP t_cg SET AUTOREFRESH STATE PAUSED;
   Command> LOAD CACHE GROUP t_cg COMMIT EVERY 256 ROWS PARALLEL <nThreads>;
   ```

   Note:

   - Choose *`nThreads`* based on how many CPU cores you use to insert the data into TimesTen for this load operation.
   - If there are several read-only cache groups it is recommended that you run several `LOAD` operations in separate sessions in parallel, if the TimesTen and Oracle Database resources are available.

10. After completion, verify the cache group state.

    ```
    Command> cacheGroups;
    Cache Group CACHEADMIN.T_CG:
      Cache Group Type: Read Only
      Autorefresh: Yes
      Autorefresh Mode: Incremental
      Autorefresh State: On
      Autorefresh Interval: 10 Seconds

      Autorefresh Status: ok
      Aging: No aging defined

      Root Table: ORATT.T
      Table Type: Read Only

    1 cache group found.
    ```

11. Start the replication agent on the standby database:

    ```
    ttAdmin -repStart ttdb2
    ```

12. Check the replication state on the standby:

    ```
    ttIsql ttdb2
    Command> call ttRepStateGet;
    < STANDBY, NO GRID >
    1 row found.
    ```

The standby database (`ttdb2`) has been defragmented and both the active and standby databases are functional.

Reverse the active and standby roles

In order to perform the database defragmentation on the active database, switch the roles of the active and standby database. The active (`ttdb1`) becomes the standby database. The original standby (`ttdb2`) becomes the active database.

1. Stop all application processing and disconnect all application connections. Any query only processing can be moved to work at the `ttdb2` TimesTen database.
2. Call the `ttRepSubscriberWait` built-in procedure at the current active database (`ttdb1`), with the database name and host of the current standby database (`ttdb2`) as input parameters. This ensures that all queued updates have been transmitted to the current standby database.

   Note: If you set the `waitTime` to `-1`, the call waits until all transactions that committed before the call have been transmitted to the subscriber.

   However, if you set the `waitTime` to any value (this value cannot be `NULL`), ensure that the return `timeOut` parameter value is `0x00` before continuing. If the returned value `0x01`, call the `ttRepSubscriberWait` built-in procedure until all transactions that committed before the call have been transmitted to the subscriber.

   For more information about the `ttRepSubscriberWait` built-in procedure, see "ttRepSubscriberWait" in the *Oracle TimesTen In-Memory Database Reference*.

   ```
   Command> call ttRepSubscriberWait(NULL,NULL,'ttdb2','ttsrv2', 100);
   ```

3. Stop the replication agent on the current active database.

   ```
   Command> call ttRepStop;
   ```

4. Call the `ttRepDeactivate` built-in procedure on the current active database. This puts the database in the `IDLE` state.

   ```
   Command> call ttRepDeactivate;
   Command> call ttRepStateGet;
   < IDLE, NO GRID >
   1 row found.
   ```

5. Promote the standby to active by calling the `ttRepStateSet('ACTIVE')` built-in procedure on the old standby database. This database now becomes the active database in the active standby pair. Use the `ttRepStateGet` built-on to verify that the database has become active.

   ```
   Command> call ttRepStateSet('ACTIVE');
   Command> call ttRepStateGet;
   < ACTIVE, NO GRID >
   1 row found.
   ```

6. Stop the replication agent on the database that used to be the active database.

   ```
   ttAdmin -repStop ttdb1
   ```

7. Execute `ttRepStateSave` on the new active database to set the status of the old active database to failed. As long as the standby database is unavailable, updates to the active database are replicated directly to the subscriber databases.

   ```
   Command> call ttRepStateSave('FAILED', 'ttdb1', 'ttsrv1');
   ```

8. Restart the full application workload on the new active database (`ttdb2`).

This database now acts as the standby database in the active standby pair.

Destroy and re-create the new standby

Destroy and recreate the new standby using `ttRepAdmin -duplicate` from the new active. During these steps, application processing can continue at the active database.

1. Stop the cache agent on the new standby database:

   ```
   ttAdmin -cacheStop ttdb1
   ```

2. As the cache manager user, drop all cache groups:

   ```
   Command> DROP CACHE GROUP t_cg;
   ```

3. Destroy the database:

   ```
   ttDestroy ttdb1
   ```

4. Re-create the new standby database by duplicating the new active.

   ```
   ttRepAdmin -duplicate -from ttdb2 -host ttsrv2 -setMasterRepStart
    -UID ttadmin -PWD ttadminpwd -keepCG -cacheUID cacheadmin
    -cachePWD cadminpwd ttdb1
   ```

5. Start cache and replication agents on the new standby database:

   ```
   ttAdmin -cacheStart ttdb1
   ttAdmin -repStart ttdb1
   ```

This process defragments both the active and standby databases with only a few seconds of service interruption.

Online defragmentation of databases in a non active standby pair replication scheme

The following sections describe how to defragment TimesTen databases that are involved in a non active standby pair replication scheme:

- Migrate and rebuild the standby database
- Alter the replication scheme
- Destroy and re-create a database

Note: These sections discuss how to defragment databases that are involved in a bidirectional replication scheme. In bidirectional replication schemes, each database is both a master and subscriber.

The examples in this section show how to perform an online defragmentation with bidirectional and unidirectional replication schemes with two TimesTen databases named `ttdb1` and `ttdb2`. For the unidirectional replication example, `ttdb1` represents the master and `ttdb2` represents the subscriber.

Migrate and rebuild a database

The first step in the procedure is to stop replication on one of the TimesTen databases and then defragment this database.

Note: While one of the databases is defragmented, application processing can continue on the other database.

Perform the following to save a copy of the TimesTen database:

1. Stop the replication agents on one of the databases.

   On the `ttdb2` database:

   ```
   ttAdmin -repStop ttdb2
   ```

2. Save a copy of the `ttdb1` database using `ttMigrate`.

   ```
   ttMigrate -c -relaxedUpgrade ttdb2 ttdb2.dat
   ```

3. Destroy the database:

   ```
   ttDestroy ttdb2
   ```

4. As a TimesTen user with `ADMIN` privileges, rebuild the `ttdb2` database:

   ```
   ttMigrate -r -relaxedUpgrade -connstr "dsn=ttdb2;uid=ttadmin;pwd=ttadminpwd"
   ttdb2.dat
   ```

   At this time:

   - All the users have been restored to `ttdb2`.
   - The replication agent is not running.

5. Restart the replication agent on `ttdb2`:

   ```
   ttAdmin -repStart ttdb2
   ```

The `ttdb2` TimesTen database has been defragmented.

Alter the replication scheme

In order to perform the database defragmentation on the `ttdb1` database, perform the following:

1. Stop all application processing and disconnect all application connections. Any processing can be moved to work at the `ttdb2` TimesTen database.
2. Call the `ttRepSubscriberWait` built-in procedure at the TimesTen database that has not been defragmented (`ttdb1`), with the database name and host of the defragmented database (`ttdb2`) as input parameters. This ensures that all queued updates have been transmitted to both databases.

Note: If you set the `waitTime` to `-1`, the call waits until all transactions that committed before the call have been transmitted to the subscriber.

However, if you set the `waitTime` to any value (this value may not be `NULL`), ensure that the return `timeOut` parameter value is `0x00` before continuing. If the returned value `0x01`, call the `ttRepSubscriberWait` built-in procedure until all transactions that committed before the call have been transmitted to the subscriber.

For more information about the `ttRepSubscriberWait` built-in procedure, see "ttRepSubscriberWait" in the *Oracle TimesTen In-Memory Database Reference*.

On ttdb1:

```
ttIsql ttdb1

Command> call ttRepSubscriberWait(NULL,NULL,'ttdb2','ttsrv2', 100);
```

If you are using a bidirectional replication scheme, skip steps 3-4 and move to step 5.

3. For a unidirectional replication scheme, where `ttdb1` is the master and `ttdb2` is the subscriber, drop the replication scheme on both TimesTen databases:

   On `ttdb1`:

   ```
   ttIsql ttdb1

   Command> DROP REPLICATION r1;
   ```

   On `ttdb2`:

   ```
   ttIsql ttdb2

   Command> DROP REPLICATION r1;
   ```

4. For a unidirectional replication scheme, drop the replication scheme on the master (`ttdb1`) and subscriber (`ttdb2`):

   On `ttdb1`:

   ```
   ttIsql ttdb1

   Command> DROP REPLICATION r1;
   ```

   On `ttdb2`:

   ```
   ttIsql ttdb2

   Command> DROP REPLICATION r1;
   ```

5. Start the replication agent on `ttdb2`:

   ```
   ttAdmin -repStart ttdb2
   ```

6. Stop the replication agent on `ttdb1`.

   ```
   ttAdmin -repStop ttdb1
   ```

If you modified a unidirectional replication scheme, the `ttdb2` database now acts as the master database in the unidirectional scheme; the `ttdb1` database acts as the subscriber database in the unidirectional replication scheme.

Destroy and re-create a database

Destroy and recreate the TimesTen database in the replication scheme that has not yet been defragmented using `ttRepAdmin -duplicate`. During these steps, application processing can continue on the defragmented database.

1. Destroy the database:

   ```
   ttDestroy ttdb1
   ```

2. Recreate the new TimesTen database (`ttdb1`) by duplicating the previously defragmented TimesTen database (`ttdb2`) involved in the replication scheme.

   ```
   ttRepAdmin -duplicate -from ttdb2 -host ttsrv2 -setMasterRepStart
    -UID ttadmin -PWD ttadminpwd ttdb1
   ```

3. Start the replication agent on the new standby database:

   ```
   ttAdmin -repStart ttdb1
   ```

This process defragments both the TimesTen databases involves in either a unidirectional or bidirectional replication scheme with only a few seconds of service interruption.

2 Working with the TimesTen Client and Server

You can open client/server connections across a network to the TimesTen database with the TimesTen Client and TimesTen Server.

The following sections describe the TimesTen Client/Server and how to connect using them:

- Overview of the TimesTen Client/Server
- Configuring TimesTen Client and Server
- Running the TimesTen Server
- Accessing a remote database on UNIX

Overview of the TimesTen Client/Server

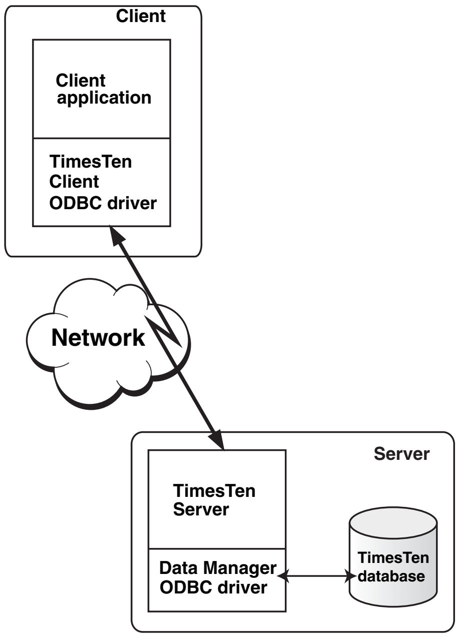
The TimesTen Server is a process that runs on a server system that takes network requests from TimesTen Clients and translates them into operations on databases on the server system. This enables clients to connect to databases that are located on different systems, potentially running a different platform and operating system bit level.

You can install the TimesTen Client on a separate or the same system as the TimesTen Server. If you install the TimesTen Client on the same system as the TimesTen Server, you can use it to access TimesTen databases on the local system. For example, this is useful when you want a 32-bit application to access a 64-bit database on the same system, 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.

Figure 2–1 demonstrates how the TimesTen Client and TimesTen Server communicate using their respective drivers.

Figure 2–1 Diagram of TimesTen Client and TimesTen Server



- TimesTen Client: To access TimesTen databases on remote systems, link your application with the TimesTen Client ODBC driver. The application then communicates with the TimesTen Server process. Using the TimesTen Client driver, applications can connect transparently to TimesTen databases on a remote or local system that has the TimesTen Server and Data Manager installed.

  You can link a client application directly with the TimesTen Client driver or with the Windows ODBC driver manager to access the TimesTen database. TimesTen supplies a driver manager for UNIX and Windows with the Quick Start sample applications. Note that there are performance considerations in using a driver manager.

  Also, you can link a client application through a provided interface, like JDBC, OCI or Pro*C/C++, to access the TimesTen database.

  Note: For details on how the application can use different drivers and interfaces to access a TimesTen database, see "Connecting to TimesTen with ODBC and JDBC drivers" on page 1-1

- TimesTen Server: On the server system, the TimesTen Server uses the TimesTen Data Manager ODBC driver. The server's responsibility is to listen for requests from a client application, process the request through the Data Manager ODBC driver, and then send the results and any error information back to the client application.

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

The following sections describe the restrictions and the communication protocols for client/server communication:

- Restrictions on client/server communication
- Communication protocols for Client/Server communication

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/server 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 TimesTen Client driver and can be used to connect to client DSNs when accessing a database over a client/server connection. 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 procedure cannot be used over a client/server 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/server 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 fails. 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 4-3 and "Granting privileges to connect to the database" on page 4-11.
- On UNIX, TimesTen does not allow a child process to use a connection opened by its parent. Any attempt from a child process using `fork()` to use a connection opened by the parent process returns an error.

Communication protocols for 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 system, you can pre-spawn a reserve pool of server processes. See "Prespawning TimesTen Server processes" on page 3-9 for details.

The following sections describe the communication protocols that the TimesTen Client can use 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 using TCP/IP sockets. This is the only form of communication available when the TimesTen Client and Server are installed on different systems.

Shared memory communication

If both the TimesTen Client and Server are installed on the same system, applications using the TimesTen Client ODBC driver may use a shared memory segment for inter-process communication (IPC). Using a shared memory segment provides better performance than TCP/IP communication. To use a shared memory segment as communication, you must:

1. Configure the server options to use shared memory communication in the `ttendaemon.options` file. See "Using shared memory for Client/Server IPC" on page 3-11.
2. Define the Network Address as `ttShmHost` in the logical server name. See "Defining a logical server name" on page 2-9 for details.

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 system, you can use UNIX domain sockets for communication. Using a shared memory segment allows for the best performance but slightly 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 "Defining a logical server name" on page 2-9 for more information.

Configuring TimesTen Client and Server

Note: Before configuring the TimesTen Client and Server, read "Connecting to TimesTen with ODBC and JDBC drivers" on page 1-1 and "Specifying Data Source Names to identify TimesTen databases" on page 1-4.

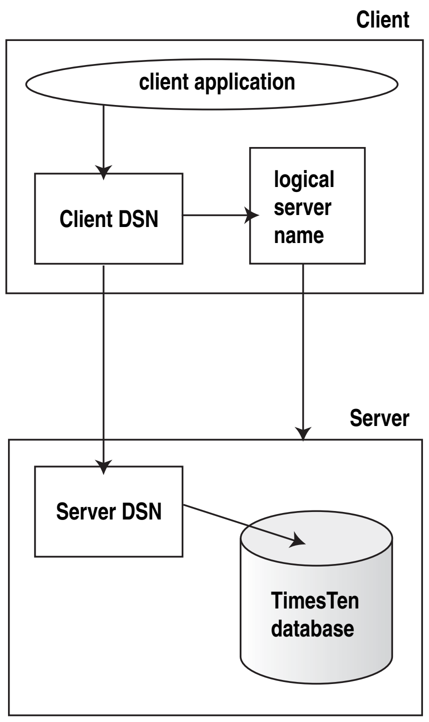
The following sections describe how to connect an application to a TimesTen database using TimesTen Client and Server:

- Overview of TimesTen Client/Server configuration
- Installing and configuring for client/server connections
- Defining server DSNs on a TimesTen Server system
- Defining a logical server name
- Creating client DSNs on a TimesTen Client system
- Using automatic client failover

Overview of TimesTen Client/Server configuration

Before the client application can connect to a TimesTen database, the user must configure, as shown in Figure 2–2, the client DSN, optionally a logical server name, and a server DSN to uniquely identify the desired TimesTen database.

Figure 2–2 Configuring for a client/server connection



The client application refers to the client DSN when initiating a connection. With the following details, the connection request is resolved to be able to connect to the intended TimesTen database:

- The client DSN is configured in either the user or system `odbc.ini` file with the server host name, either the logical server name or the actual server system name, and the server DSN that identifies the TimesTen database on the server.
- The logical server name is an optional configuration on the client. When used, it specifies the server host name where the TimesTen database is installed. This is used when you want to hide or simplify the server host name. You must use the logical server name when using shared memory IPC or UNIX domain sockets.
- The server DSN is configured in the system `odbc.ini` file with the TimesTen database name and its connection attributes. The connection attributes specify how the TimesTen database is loaded and configured, and how the connections to it are to be controlled or managed.

Thus, when these are configured correctly, the client application can use the client DSN to locate and connect to the TimesTen database. The client DSN defines the server system and the server DSN. The server DSN, in turn, specifies the TimesTen database on that server, how the database is to be loaded, and how connections are to be managed.

Installing and configuring for client/server connections

The following sections describe what you must install on which node for client/server connections:

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

Configuring Client/Server of the same TimesTen release

The following sections describe how to install and configure TimesTen when the Client and Server are of the same TimesTen release:

- Install and configure the TimesTen Server
- Install and configure the TimesTen Client

Install and configure the TimesTen Server Perform the following tasks on the system on which the TimesTen database resides. This system is called the server system.

1. Install the TimesTen Server. For information on how to install the TimesTen Server, see the *Oracle TimesTen In-Memory Database Installation Guide*.
2. Create and configure a server DSN corresponding to the TimesTen database. See "Defining server DSNs on a TimesTen Server system" on page 2-7. Set TimesTen connection attributes in the server DSN. See "Connection Attributes" in the *Oracle TimesTen In-Memory Database Reference*.

Install and configure the TimesTen Client Perform the following tasks on the system where the client application resides. This system is called the client system.

1. Install the TimesTen Client. For information on how to install the TimesTen Client, see the *Oracle TimesTen In-Memory Database Installation Guide*.
2. If you are using JDBC to connect to the database, install the Java Developer's Kit (JDK) and set up the environment variables, such as `CLASSPATH` and the shared library search path. See "Setting the environment for Java development" in the *Oracle TimesTen In-Memory Database Java Developer's Guide* for details.
3. Create and configure a client DSN corresponding to the server DSN. See "Creating and configuring client DSNs on UNIX" on page 2-17 and "Creating and configuring client DSNs on Windows" on page 2-13.
4. For OCI and Pro*C/C++ client/server connections, configure the application to use either `tnsnames.ora` or easy connect as described in "Connecting to a TimesTen database from OCI" in the *Oracle TimesTen In-Memory Database C Developer's Guide*.
5. 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/C++ applications in the same manner as any OCI or Pro*C/C++ direct connect applications, which is described in "TimesTen Support for OCI" and "TimesTen Support for Pro*C/C++" in the *Oracle TimesTen In-Memory Database C Developer's Guide*.

Configuring cross-release TimesTen Client/Server

A TimesTen Client can connect to a TimesTen Server from a different release and of a different bit level. A TimesTen Client 7.0 or later release may connect to a TimesTen

Server 7.0 or later release of any bit level. If you are configuring for a cross-release TimesTen Client/Server connection, install and configure as directed in "Configuring Client/Server of the same TimesTen release" on page 2-6.

The TimesTen Server loads a driver and a TimesTen database of its own release and bit level when a TimesTen Client application connects to the TimesTen database. The TimesTen Data Manager is automatically installed on the Server system.

- 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.

Defining server DSNs on a TimesTen Server system

Server DSNs identify TimesTen databases that are accessed by a client/server connection. A server DSN must be defined as a system DSN and has the same configuration format and attributes as a TimesTen Data Manager DSN. For a description of DSNs and instructions on creating them, see "Creating a Data Manager DSN on Windows" on page 1-9 or "Creating a Data Manager DSN on UNIX" on page 1-13.

Note: You can add or configure a server DSN while the TimesTen Server is running.

Because a server DSN identifies databases that are accessed by a TimesTen Server, a server DSN can be configured using the same connection attributes as a Data Manager DSN. In addition, there are connection attributes that are only allowed within the server DSN specification. These attributes enable you to specify multiple client/server connections to a single server.

Note: Some connection attributes, including the ones described in the following sections, can be configured in the TimesTen daemon options file (`ttendaemon.options`). If you have set the same connection attributes in both the server DSN and the daemon options file, the value of the connection attributes in the server DSN takes precedence.

For a description of the TimesTen daemon options see "Managing TimesTen Client/Server options" on page 3-9.

The following sections describe the server DSN attributes in the context of the `odbc.ini` file or the ODBC Data Source Administrator:

Note: For a complete description of the TimesTen Server connection attributes, see "Connection Attributes" in the *Oracle TimesTen In-Memory Database Reference*.

- Server DSN connection attributes defined in odbc.ini file

- Server DSN connection attributes defined in ODBC Data Source Administrator

Server DSN connection attributes defined in odbc.ini file

By default, TimesTen creates only one connection to a server per child process. However, the following Server connection attributes enable you to specify multiple client/server connections to a single TimesTen Server:

> **Note:** These attributes are read at first connection. Changes to TimesTen Server settings do not occur until the TimesTen server is restarted. To restart the Server, use the following command:
>
> ```
> ttDaemonAdmin -restartserver
> ```

- `MaxConnsPerServer`: Set the maximum number of client connections that are handled by a single server process. The server is referenced by the server DSN. The server may have multiple server processes, where each process can only have the maximum number of connections as specified by this attribute.
- `ServersPerDSN`: You can have multiple server processes serving multiple incoming connections on the server. The `ServersPerDSN` attribute specifies the number of server processes that are initially spawned on the server. Each new incoming connection spawns a new server process up to the `ServersPerDSN`. When `ServersPerDSN` is reached, the existing server processes handle multiple connections up to the number specified in `MaxConnsPerServer`.
- `ServerStackSize`: Set the size of the stack on the Server for each connection.

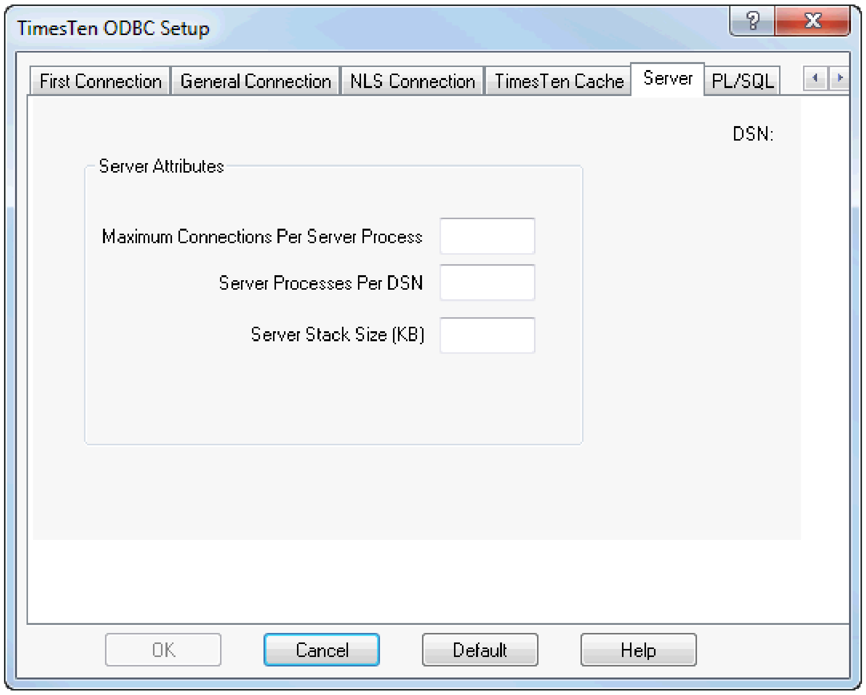
The `MaxConnsPerServer` and `ServersPerDSN` attributes are related. Neither of these attributes limits the number of client connections to given DSN. Instead, they control how connections are distributed over server processes. For example, if `MaxConnsPerServer` is set to 2 and `ServersPerDSN` is set to 5, then the following occurs:

- Connection 1 arrives at the server, the first server process is started for this connection. Connections 2 through 5 arrive at the server, server processes 2 through 5 are initiated where each server process services a connection.
- Connection 6 arrives at the server. Since `ServersPerDSN` is reached, and `MaxConnsPerServer` is not, connection 6 is given to the first server process. Incoming connections 7 through 10 are given respectively as the second connection to server processes 2 through 5.
- Connection 11 arrives at the server. Both `ServersPerDSN` and `MaxConnsPerServer` are reached, so server process 6 is started to handle connection 11.

Server DSN connection attributes defined in ODBC Data Source Administrator

If you anticipate having more than one connection using the server DSN, specify appropriate values for the server DSN attributes as needed. On Windows in the ODBC Data Source Administrator, these are specified on the **Server** tab.

Figure 2–3 Server tab in TimesTen ODBC Administrator



Defining a logical server name

A logical server name is a definition for a server system on the TimesTen Client. In some cases, such as when using a communication protocol other than TCP/IP for local client/server or the TimesTen Server process is not listening on the default TCP/IP port, you must define a logical server name on the client system. In these cases, the client DSN must refer to the logical server name. However, in most cases when the communication protocol used is TCP/IP, the client DSN can refer directly to the server host name without having to define a logical server name.

The following sections demonstrate how to define a logical server name on Windows or UNIX platforms:

- Creating and configuring a logical server name on Windows
- Creating and configuring a logical server name on UNIX

Creating and configuring a logical server name on Windows

To create and configure a logical server name:

1. On the Windows Desktop from the **Start** menu, select **Settings**, **Control Panel**, **Administrative Tools**, and finally **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 11.2.2** 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.

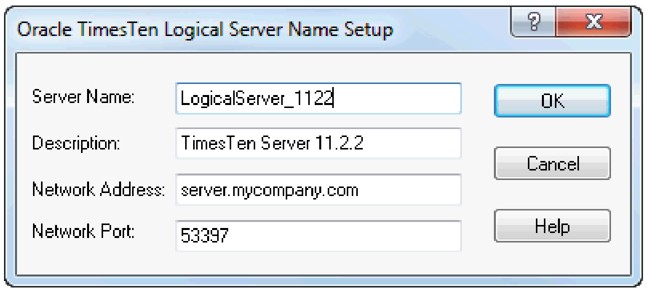
6. In the **Server Name** field, enter a logical server name.

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 system. The Network Address must be one of:

Type of connection	Network Address
Remote client/server connection	The name of the system where the TimesTen Server is running. For example, `server.mycompany.com`
Local client/server connection that uses shared memory for inter-process communication (IPC)	`ttShmHost` In order to use shared memory as IPC, verify that you have configured your system correctly. See "Shared memory communication" on page 2-4.

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:



Server Name: LogicalServer_1122
Description: TimesTen Server 11.2.2
Network Address: server.mycompany.com
Network Port: 53397

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

To delete a server name:

1. On the Windows Desktop on the client system from the **Start** menu, select **Settings**, and then select **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**.

Creating and configuring a logical server name on UNIX

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.

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 *install_dir*/info/sys.ttconnect.ini file

Working with the ttconnect.ini file TimesTen uses the `ttconnect.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 the system where the TimesTen Client is installed. By default, the `ttconnect.ini` file is *install_dir*/info/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.

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*]	Logical server name of the TimesTen Server system
`Description=`*description*	Description of the TimesTen Server
`Network_Address=`*network-address*	The DNS name, host name or IP address of the system 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 11.2.2 is 53393 for 32-bit platforms and 53397 for 64-bit platforms.

The Network Address must be one of the following:

Type of connection	Network address
Remote client/server connection	The name of the system where the TimesTen Server is running. For example, `server.mycompany.com`
Local client/server connection that uses UNIX domain sockets	`ttLocalHost`
Local client/server connection that uses shared memory for inter-process communication	`ttShmHost`

Example 2–1 Defining a logical server name

This example from a `ttconnect.ini` file defines a logical server name, `LogicalServer_1122`, for a TimesTen Server running on the system `server.mycompany.com` and listening on port `53397`. The instance name of the TimesTen installation is `tt1122_64`.

```
[LogicalServer_1122]
Description=TimesTen Server 11.2.2
Network_Address=server.mycompany.com
TCP_Port=53397
```

Example 2–2 Using UNIX domain sockets for communication

If both the client and server are on the same UNIX system, 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 is listening so that multiple instances of the same version of TimesTen Server can be run on the same system. To achieve this, the logical server name definition in `ttconnect.ini` file might look like:

```
[LocalHost_1122]
Description=Local TimesTen Server 11.2.2 through domain sockets
Network_Address=ttLocalHost
TCP_Port=53397
```

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

If both the client and server are on the same system, 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 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_1122]
Description= Local TimesTen Server 11.2.2 through shared memory
Network_Address=ttShmHost
TCP_Port=53397
```

Creating client DSNs on a TimesTen Client system

A client DSN specifies a remote database and uses the TimesTen Client. The client DSN can be defined as a user or as a system DSN. A client DSN refers to a TimesTen database indirectly by specifying a `hostname, DSN` pair, where the hostname represents the server system on which TimesTen Server is running and the DSN refers to a server DSN that is defined on that host. These are configured within the client DSN connection attributes.

Note: For a complete description of the TimesTen client connection attributes, see "Connection Attributes" in the *Oracle TimesTen In-Memory Database Reference*.

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. For example, you could 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.

Note: If you configure any of the server DSN data store or first connection attributes within the definition of the client DSN, they are ignored. However, the TimesTen Client allows most of the server DSN attributes (except for the DataStore connection attribute) to be passed in as part of the connection or URL string. These are transparently passed on to the server and overrides what is configured in the server DSN.

The following sections describe how to create a client DSN and its attributes on either the Windows or UNIX platforms:

- Creating and configuring client DSNs on Windows

- Creating and configuring client DSNs on UNIX

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.

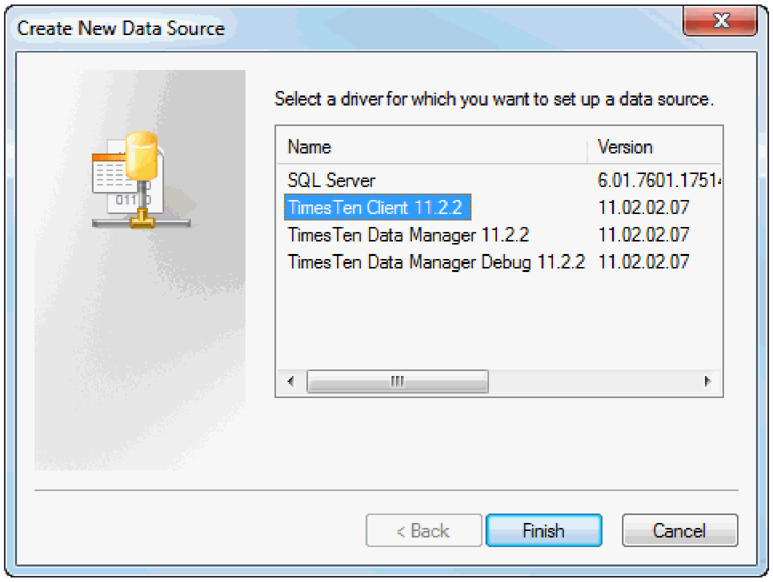
This section includes the following topics:

- Creating a client DSN on Windows
- Setting the timeout interval and authentication
- Accessing a remote database on Windows
- Testing connections

Creating a client DSN on Windows

To define a TimesTen client DSN:

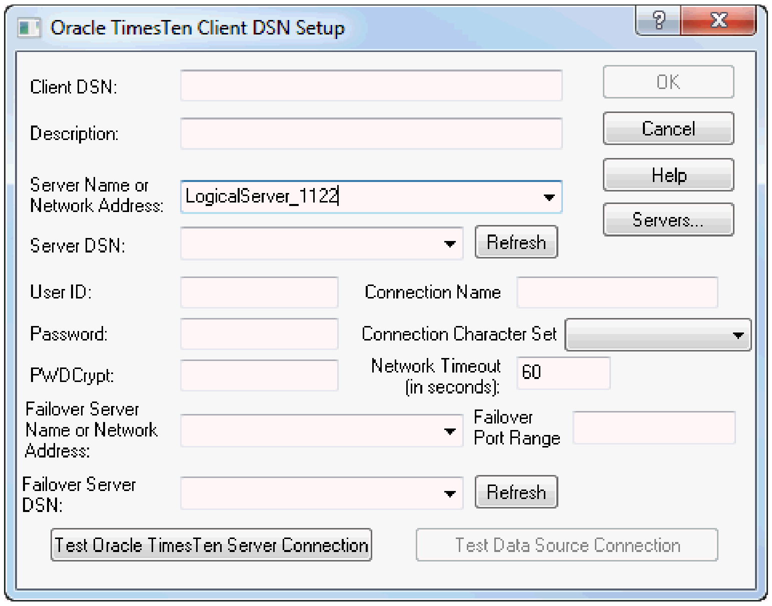
1. On the Windows Desktop from the **Start** menu, select **Settings**, **Control Panel**, **Administrative Tools**, and finally **Data Sources (ODBC)**. This opens the ODBC Data Source Administrator.
2. Choose either **User DSN** or **System DSN**. For a description of user DSNs and system DSNs see "Specifying Data Source Names to identify TimesTen databases" on page 1-4.
3. Click **Add**. This opens the Create New Data Source dialog.



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

Name	Version
SQL Server	6.01.7601.1751
TimesTen Client 11.2.2	11.02.02.07
TimesTen Data Manager 11.2.2	11.02.02.07
TimesTen Data Manager Debug 11.2.2	11.02.02.07

4. Choose **TimesTen Client 11.2.2**. Click **Finish**. This opens the Oracle TimesTen client DSN Setup dialog.


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

Server Name or Network Address: LogicalServer_1122

Network Timeout (in seconds): 60

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 system 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 system.

   - The name can be a host name, IP address or logical server name. The logical server names defined on the client system 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 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 Windows" on page 2-9.

8. In the **Server DSN** field, enter the server DSN corresponding to the database that the client application accesses.

   - 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 system 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 system where the TimesTen Server is running.

For more information about customizing which server DSNs show in this list, see "ODBC Data Sources" on page 1-22.

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*.

10. If you are using automatic client failover, you would configure the **Failover Server Name**, **Failover Port Range**, and **Failover DSN**. For details, see "Configuring automatic client failover" on page 2-20.

Setting the timeout interval and authentication You can define the user name, password and timeout interval for the client/server connection in the client DSN with the `UID`, `PWD`, and `Timeout` attributes. However, configuring the authentication in the client DSN is optional, since you can provide the user name and password when connecting. It is strongly discouraged to supply the password in the client DSN, since the password is stored unencrypted on the client.

For a description of the `UID`, `PWD`, and `Timeout` attributes, see "Connection 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 system.

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.

Accessing a remote database on Windows In this example, the TimesTen Client system is `client.mycompany.com`. The client application is accessing the server DSN on the remote server system, `server.mycompany.com`**.** The logical server name is `LogicalServer_1122`.

Note: These examples reference the sample DSNs preconfigured in a fresh TimesTen installation.

---

1. On the server system `server.mycompany.com`, use the `ttStatus` utility to verify that the TimesTen Server is running and to verify the port number it is listening on.

2. Using the procedure in "Defining server DSNs on a TimesTen Server system" on page 2-7, verify that the server DSN, `sampledb_1122`, is defined as a system DSN on `server.mycompany.com`.

3. On the client system, `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 `LogicalServer_1122`.
   - In the Network Address field, enter `server.mycompany.com`.

- In the Network Port field, enter `53397`. This is the default port number for the TimesTen Server on 64-bit platforms for TimesTen Release 11.2.2. This value should correspond to the value displayed by `ttStatus` in Step 1.

   See "Creating and configuring a logical server name on Windows" on page 2-9 for the procedure to open the TimesTen Server Name dialog and for more details.

4. On the client system, `client.mycompany.com`, create a client DSN that corresponds to the remote server DSN, `sampledbCS_1122`. In the TimesTen client DSN Setup dialog, enter the following values:
   - In the **Client DSN** field, enter `sampledbCS_1122`.
   - In the **Server Name** or **Network Address** field, enter `LogicalServer_1122`.
   - In the **Description** field, enter a description for the server. Entering data into this field is optional.
   - In the **Server DSN** field, enter `sampledb_1122`.
5. Run the client application from the system `client.mycompany.com` using the client DSN, `sampledbCS_1122`. The example below uses the `ttIsql` program installed with TimesTen Client.

   ```
   ttIsql -connStr "DSN=sampledbCS_1122"
   ```

The next 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 53397. The following methods can be used to connect to a Server DS:

1. Define the logical server name `LogicalServer_1122` with `server.mycompany.com` as the Network Address and `53397` as the Network Port. Define `Client_DSN` as the client DSN with `LogicalServer_1122` as the Server name and `Server_DSN` as the server DSN. Then, execute the command:

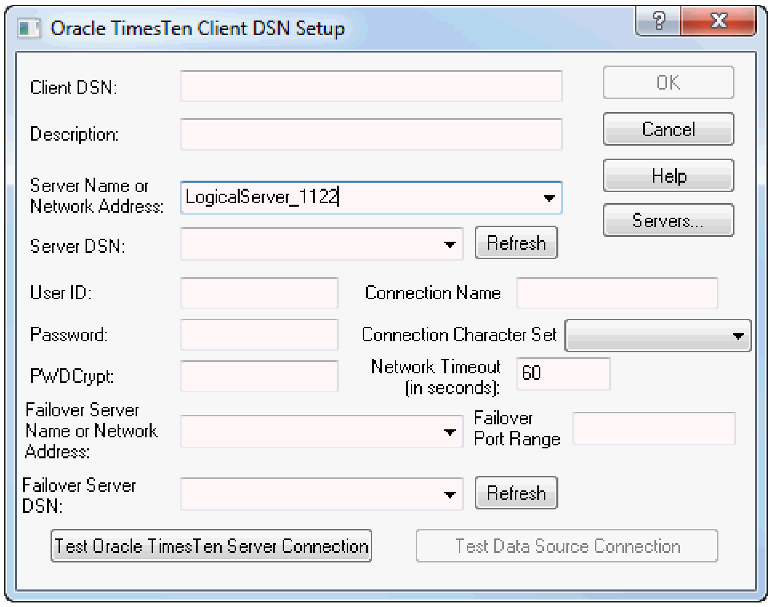
   ```
   ttIsql -connStr "DSN=Client_DSN"
   ```

2. Alternatively, define the logical server name `LogicalServer_tt1122` with `server.mycompany.com` as the Network Address and the default port number as the Network Port. Define `Client_DSN` as the client DSN with `LogicalServer_1122` as the Server name and `Server_DSN` as the server DSN. Overwrite the port number in the command:

   ```
   ttIsql -connStr "DSN=Client_DSN;TCP_Port=53397"
   ```

Testing connections To test client application connections to TimesTen databases:

1. On the Windows Desktop from the **Start** menu, select **Settings**, and then select **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: LogicalServer_1122

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 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 system.
   - The server specified in the **Server Name** or **Network Address** field of the TimesTen Client DSN Setup dialog is defined and the corresponding system exists.
   - The TimesTen Server is running on the server system.

6. Click **Test Data Source Connection** to test the connection to the server DSN. The ODBC Data Source Administrator attempts to connect to the 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 system.
   - 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; 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 "Specifying Data Source Names to identify TimesTen databases" on page 1-4.

> **Note:** The syntax for defining the client DSN in the `odbc.ini` file is described in "odbc.ini file entry descriptions" on page 1-22.

In the ODBC Data Sources section of the `odbc.ini` file, add an entry for the client DSN. The client DSN specifies the location of the TimesTen database with the following attributes:

- ODBC client driver to use for the connection.
- The server system on which the database resides on is specified in the `TTC_Server` attribute.
- The server DSN that specifies the intended database is specified in the `TTC_Server_DSN` attribute.

> **Note:** The server DSN is defined on the server system where the database resides.

For each TimesTen database with which the client connects needs to have two entries:

- Define the client DSN name and provide the name of the ODBC client driver to use in the ODBC Data Sources section.

> **Note:** All available ODBC client drivers are listed in "Connecting using TimesTen ODBC drivers" on page 1-3.

- Create an entry with the client DSN you defined in the ODBC Data Sources section. Within this section, specify the server system and the server DSN.

The following is the syntax for providing the client DSN name and the ODBC client driver to use:

```
[ODBC Data Sources]
Client_DSN=Name-of-ODBC-driver
```

For example, to defined `sampbledbCS_1122` as the client DSN and associate it with the TimesTen Client ODBC driver, you would make the following entry in the ODBC Data Sources section of the `odbc.ini` file:

```
[ODBC Data Sources]
sampledbCS_1122=TimesTen Client 11.2.2
```

After the ODBC Data Sources section, you would add an entry to specify the server system and server DSN for each data source you defined. Each client DSN listed in the ODBC Data Sources section of the `odbc.ini` file requires a its own specification section.

The following is an example specification of the TimesTen client DSN `sampledbCS_1122` where the server is configured with a logical server name of `LogicalServer_1122` and the server DSN is `sampledb_1122`:

```
[sampledbCS_1122]
TTC_Server=LogicalServer_1122
TTC_Server_DSN=sampledb_1122
```

The `TTC_Server*` attributes are the main attributes for a client DSN definition. There are only a few client connection attributes, each of which are for identifying the server DSN. If you provide any server connection attributes in the client definition, these attributes are ignored. General connection attributes may be specified in the client definition. For a description of all client and general connection attributes available for use in the `odbc.ini` file, see "Connection Attributes" in the *Oracle TimesTen In-Memory Database Reference*.

Using automatic client failover

Automatic client failover is for use in High Availability scenarios with a TimesTen active standby pair replication configuration. Consider a scenario where failure of the active TimesTen node has resulted in the original standby node becoming the new active node. With the automatic client failover feature, failover (transfer) to the new active (original standby) node occurs, and applications are automatically reconnected to the new active node. TimesTen provides features that enable applications to be alerted when this occurs, so they can take any appropriate action.

Note: Automatic client failover is complementary to Oracle Clusterware in situations where Oracle Clusterware is used, but the two features are not dependent on each other. For information about Oracle Clusterware, you can refer to "Using Oracle Clusterware to Manage Active Standby Pairs" in the *Oracle TimesTen In-Memory Database Replication Guide*.

The following sections describe how to use and enable automatic client failover:

- Features and functionality of automatic client failover
- Configuring automatic client failover

Features and functionality of automatic client failover

When an application connects to the active node, the connection is registered and this registration is replicated to the standby node. If the active node fails, the standby node becomes the active node and then notifies the client of the failover. At this point, the client has a new connection to the new active node. No state from the original connection, other than the connection handle, is preserved. All client statement handles from the original connection are marked as invalid.

When failover completes, TimesTen makes a callback to a user-defined function that you register. This function takes care of any custom actions you want to occur in a failover situation.

Note: For C developers, details of how to create the callback and to facilitate an automatic client failover are discussed in "Using automatic client failover in your application" in the *Oracle TimesTen In-Memory Database C Developer's Guide*.

For Java developers, details are discussed in "JDBC support for automatic client failover" in the *Oracle TimesTen In-Memory Database Java Developer's Guide*.

The following items list the behavior of automatic client failover in particular failure scenarios:

- If the client library loses the connection to the active node, it fails over and attempts to switch to the standby node.
- If, for some reason, there is no active node (no failover has occurred at the server side and both servers are either standby or idle), applications cannot connect. But automatic client failover continues to alternate between both servers until it finds an active node or times out.

- If a failover has already occurred and the client is already connected to the new active node, the next failover request results in an attempt to reconnect to the original active node. If that fails, alternating attempts are made to connect to the two servers until it times out.

> **Note:** When TimesTen attempts to recover from failure when using automatic client failover, a timeout specifies the duration for all attempts at failover recovery. The connection might be blocked while failover is attempted.
>
> This timeout defaults to 60 seconds or can be set with the value of the `TTC_Timeout` connection attribute. The minimum timeout is 60 seconds, regardless of the `TTC_Timeout` setting. Refer to "TTC_ Timeout" in *Oracle TimesTen In-Memory Database Reference* for information about that attribute.

- If the active node fails before the client registration is successfully propagated by replication to the standby node, the client does not receive a failover message and the connection registration is lost. However, the client library eventually notices (through TCP) that its connection to the original active node is lost and initiates a failover attempt.

> **Notes:**
>
> - These features apply only to client/server connections, not direct connections.
> - Failover connections are created only as needed, not in advance.
> - Using automatic client failover results in one additional database connection per server process, which should be considered as you choose a setting for the TimesTen `Connections` attribute (upper limit of the number of concurrent connections to the database). The number of server processes, in turn, is affected by the setting of the `MaxConnsPerServer` attribute (maximum number of concurrent connections a child server process can handle). For example, if you have 12 connections and `MaxConnsPerServer=3`, then there are four server processes. Therefore, if some of the connections use automatic client failover, there are four additional connections.

Configuring automatic client failover

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

> **Note:** See "Using automatic client failover in your application" in *Oracle TimesTen In-Memory Database C Developer's Guide* for information about connection option persistence after failover.

Configuring automatic client failover on Windows In the Oracle TimesTen Client DSN Setup dialog, after you have configured the rest of the client DSN information as described in

"Creating and configuring client DSNs on Windows" on page 2-13, complete the following fields:

1. In the **Failover Server Name or Network Address** field, specify the logical server or network address of the server system.
   - The name can be a host name, IP address or logical server. The logical server names defined on the client system 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 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 Windows" on page 2-9.
2. In the **Failover Server DSN** field, enter the server DSN corresponding to the standby database.
   - 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 system 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 system where the TimesTen Server is running.
3. Optionally, specify the **Failover Port Range** for the port or port range where TimesTen listens for failover notifications. This connection attribute needs to be specified as a single port number if you have firewall issues between the server hosts and the client hosts. By default, TimesTen uses a port chosen by the operating system. The failover notifications are issued from the server hosts to the client hosts. A port range is set as a lower and upper value separated by hyphen.

   You only need to specify a port range if both of these conditions are true:
   - You have firewall issues between the server hosts and the client hosts.
   - You use multiple client applications configured for automatic client failover on a single client host.

   Each client application configured for automatic client failover on a client host needs to listen on a distinct port for failover notifications.

Configuring automatic client failover on UNIX On the client, configure the following:

1. In the `odbc.ini` file on the client, define the client DSNs for the standby node by configuring `TTC_Server2` to the server on which the standby database resides. Set `TTC_Server_DSN2` to the name of the standby database.

   For example:

```
[MyDSN]
TTC_Server=LogicalServer_1122
TTC_Server_DSN=MyDSN
TTC_Timeout=60
TTC_Server2=LogicalServer2_1122
TTC_Server_DSN2=MyDSN_Standby
```

2. In the `ttconnect.ini` file on the client, define the logical server for the standby node by configuring the `Network_Address` and `TCP_Port` to the server on which the standby database resides.

   ```
   [LogicalServer_1122]
   Description=TimesTen Server 11.2.2
   Network_Address=server.mycompany.com
   TCP_Port=53397

   [LogicalServer2_1122]
   Description=TimesTen Server 11.2.2
   Network_Address=server2.mycompany.com
   TCP_Port=53397
   ```

   Setting any of `TTC_Server2`, `TTC_Server_DSN2`, or `TCP_Port2` implies the following:

   - You intend to use automatic client failover.
   - You understand that a new thread is created for your application to support the failover mechanism.

3. Optionally, configure the `TTC_FailoverPortRange` attribute to specify a port or port range where the failover thread listens for failover notifications. This connection attribute needs to be specified as a single port number if you have firewall issues between the server hosts and the client hosts. By default, a port chosen by the operating system is used. The failover notifications are issued from the server hosts to the client hosts. A port range is set as a lower and upper value separated by hyphen.

   You only need to specify a port range if both of these conditions are true:

   - You have firewall issues between the server hosts and the client hosts.
   - You use multiple client applications configured for automatic client failover on a single client host.

   Each client application configured for automatic client failover on a client host needs to listen on a distinct port for failover notifications.

Notes:

- Like other connection attributes, `TTC_Server2`, `TTC_Server_DSN2`, and `TCP_Port2` can be specified in the connection string, overriding any settings in the DSN.
- If `TTC_Server2` is specified but `TTC_Server_DSN2` and `TCP_Port2` are not, then `TTC_Server_DSN2` is set to the `TTC_Server_DSN` value and `TCP_Port2` is set to the `TCP_Port` value.
- If the client library cannot connect to `TTC_Server_DSN`, it tries the standby, `TTC_Server_DSN2`.
- `TTC_Server` and `TTC_Server2` can have the same setting if it is a virtual IP address.

Running the TimesTen Server

The TimesTen Server is a child process of the TimesTen daemon. If you installed the TimesTen Server, this process is automatically started and stopped when the TimesTen daemon or Data Manager service is started or stopped. You can explicitly start or shut down the daemon or service with the `ttDaemonAdmin` utility.

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

The default ports for TimesTen daemon and TimesTen Servers are described in the "TimesTen Installation" 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 options, see "Modifying the TimesTen Server options" on page 3-9.

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 *`install_dir`*`/info/ttmesg.log` and *`install_dir`*`/info/tterrors.log` files by default. Optionally, you can configure TimesTen to log messages to the `syslog` facility.

See "Modifying informational messages" on page 3-6.

Accessing a remote database on UNIX

In this example, the TimesTen Client application system is a 64-bit Solaris system, `client.mycompany.com`. The client application is accessing the server DSN `sampledb_1122` on the remote server system, another 64-bit Solaris system, `server.mycompany.com`. The logical server name is `LogicalServer_1122`. The instance name of the TimesTen installation is `tt1122_64`.

1. On the server system `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 `sampledb_1122` exists in the system `odbc.ini` file on `server.mycompany.com`.

   There should be an entry in the `odbc.ini` file as follows:

   ```
   [sampledb_1122]
   Driver=install_dir/lib/libtten.so
   DataStore=install_dir/server/sampledb_1122
   ```

3. Create a logical server name entry for the remote TimesTen Server in the `ttconnect.ini` file on `client.mycompany.com`.

   ```
   [LogicalServer_1122]
   Network_Address=server.mycompany.com
   TCP_Port=53397
   ```

   See "Creating and configuring client DSNs on UNIX" on page 2-17 for information on the creating a `ttconnect.ini` file.
4. On the client system, `client.mycompany.com`, create a client DSN corresponding to the remote server DSN, `sampledb_1122`.

There should be an entry in the `odbc.ini` file as follows:

```
[sampledbCS_1122]
TTC_Server=LogicalServer_1122
TTC_Server_DSN=sampledb_1122
```

See "Overview of user and system DSNs" on page 1-5 for information on the location of the proper `odbc.ini` file.

5. Run the client application from the system `client.mycompany.com` using the client DSN, `sampledbCS_1122`. The example below uses the `ttIsql` program that is installed with TimesTen Client.

```
ttIsqlCS -connStr "DSN=sampledbCS_1122"
```

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 `53397`. The following methods can be used to connect to a Server DS:

1. Define the logical server name `LogicalServer_1122` with `server.mycompany.com` as the Network Address and `53397` as the Network Port. Define `Client_DSN` as the client DSN with `LogicalServer_1122` as the server name and `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 `LogicalServer_1122` as the server name and `Server_DSN` as the server DSN. Overwrite the port number in the command:

```
ttIsqlCS -connStr "DSN=Client_DSN;TCP_Port=53397"
```

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=53397"
```

Testing connections

To test client application connections to TimesTen databases:

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

     ```
     telnet server.mycompany.com 53397
     ```

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

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

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

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

- Use the `ttStatus` utility on the server system to determine the status and port number of the TimesTen Server. Generally, the TimesTen Server is started at installation time. If the TimesTen Server is not running, you must start it. For information on starting the TimesTen Server, see "Modifying the TimesTen Server options" on page 3-9.

4. Verify that the client application can connect to the database. If you cannot establish a connection to the database, 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 database has been defined properly in the system `odbc.ini` file on the system where the database resides and where the TimesTen Server is running.

3 Working with the TimesTen Data Manager Daemon

The 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 databases exist, which are in use, and which application processes are connected to which databases
- Manages RAM policy
- Starts replication processes, the TimesTen Server 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 databases communicate with the daemon transparently using TimesTen internal routines.

The following sections discuss interaction with the TimesTen daemon on various platforms:

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

Starting and stopping the TimesTen daemon

The following sections discuss how to start and stop the TimesTen daemon on various platforms:

- Starting and stopping the Oracle TimesTen Data Manager service on Windows
- Starting and stopping the daemon on UNIX

Note: The daemon writes a `timestend.pid` file into the directory from which the daemon was started. By default, the daemon home directory is *`install_dir`*`/info` on UNIX and *`install_dir`*`\srv\info` on Windows. This file contains the daemon process ID. When the process terminates, the `timestend.pid` file is removed.

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 the Windows Administrative Tools.
2. Double-click **Services**. All currently available services are displayed.
3. Select **TimesTen Data Manager 11.2.2**, then click the appropriate button to stop or start the service.

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

Changing the startup mode

When you install the TimesTen Data Manager, the TimesTen Data Manager Service starts automatically whenever the system restarts. In addition, if you installed the TimesTen Server, it is automatically started whenever the TimesTen Data Manager service is started. You can change the startup mode for the TimesTen Data Manager to require manual startup.

To change the startup mode:

1. Open the Windows Administrative Tools.
2. Double-click **Services**. All currently available services are displayed.
3. Double-click the **TimesTen Data Manager 11.2.2** service to examine its properties dialog.
4. In the properties dialog, the Startup type list should indicate **Automatic** (default). You can optionally change it to **Manual**. In either case, you can click **Stop** or **Start** (as applicable) in the properties dialog to stop or start the service. For typical usage, set the Startup type back to the default and click **OK** when you are through.

Starting and stopping the daemon on UNIX

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 executed. To manually start and stop the TimesTen main daemon, you can use the `ttDaemonAdmin` utility with the `-start` or `-stop` option.

The `root` user 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`*
AIX	`/etc/init.d/tt_`*`instance_name`*

Running the setuproot script

If you want the TimesTen instance to start each time the system is restarted, run the `setuproot` script as root. The `setuproot` script is located in the *`install_dir`*`/bin` directory:

```
# cd install_dir/bin
# setuproot -install
```

Shutting down a TimesTen application

A TimesTen application consists of a database that has been allocated shared memory, user 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 user connections gracefully.
2. Shut down all replication and cache agents.
3. Unload the database 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 daemon home directory:

```
install_dir\srv\info
```

On UNIX, the `ttendaemon.options` file is located in the daemon home 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 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.
- Modifying the default database recovery after a fatal error

Use the `ttmodinstall` utility to make changes to the `ttendaemon.options` file for most commonly changed options. See "ttmodinstall" in the *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 database access over NFS-mounted systems
- Enabling Linux large page support
- Modifying the default automatic database recovery after a fatal error
- Configuring a range for shared memory keys

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 system can communicate with the daemon. Processes from other systems would be excluded, so you would not be able to replicate to or from other systems, or grant client access from other systems.

If you have multiple ethernet cards on different subnets, you can specify `-listenaddr` entries to control which systems 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 3-4 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 the following:

- 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 `ttDaemonLog` utility enables you to control the type of events that TimesTen writes to and fetches from the TimesTen user and error logs. For more information, see "ttDaemonLog" in the *Oracle TimesTen In-Memory Database Reference*.

The following options in the `ttendaemon.options` file 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 100 MB.
`-userlog` *logfile* or `-userlog [syslog]`	*logfile*: A user-provided *logfile* specifies the location and name of the log file to use. The default file is *daemon_home*/`tterrors.log.` `syslog`: Specify `-userlog [syslog]` so that the output is sent to one of the following appropriate operating system files: ■ The UNIX syslog ■ The Windows 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 10 MB.
`-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. From the **Start** menu, select **Control Panel**, **Administrative Tools**, and then select **Event Viewer**.
2. In the Event Viewer list, choose **Applications and Services Logs**.

The window displays 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 databases.
- Flush the transaction 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, TimesTen spawns a minimum of 4 subdaemons and specifies the default maximum number of subdaemons at 50. However, you can change these settings by assigning new values to the `-minsubs` and `-maxsubs` options in the `ttendaemon.options` file.

Allowing database 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 transaction 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
```

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.

For more information on HugePages at the operating system level, see "Linux prerequisites" in the *Oracle TimesTen In-Memory Database Installation Guide*.

Modifying the default automatic database recovery after a fatal error

You can modify the default database recovery behavior by setting the `-enablePolicyInactive` option in the `ttendaemon.options` file before you start the TimesTen daemon. For full details on this option, see "Changes to RAM policy after automatic recovery fails" on page 1-26.

Configuring a range for shared memory keys

You can configure a range for all shared memory keys used by TimesTen with the `-shmkeyrange` daemon option. You can constrain shared memory keys to a specific range to prevent shared memory collisions. However, if you use this option, it is your responsibility to ensure that no other shared memory segments use shared memory keys in the specified range.

Note: This option is only available on UNIX or Linux platforms. This option cannot be used for Windows platforms.

The syntax is as follows:

```
-shmkeyrange low-high
```

For example:

```
-shmkeyrange 0x4000000-0x40FFFFFF
```

There can be no space in the low-high clause. The range is inclusive; thus, in the preceding example, both 0x40000000 and 0x40FFFFFF are valid keys. In addition, the minimum range size is 16.

With this option, it is possible to run out of shared memory keys, especially if there multiple invalidations or there are more TimesTen databases in use than originally anticipated.

If a user provides any of the following invalid options for the shared memory keys, then the TimesTen daemon aborts the startup process.

- Invalid numeric strings
- Only one number is specified for the range
- The first number is greater than the second number in the range specification
- The range size is less than 16.
- The range is specified incorrectly.

Managing TimesTen Client/Server options

This section includes the following topics:

Modifying the TimesTen Server options

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

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

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

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

- To enable the TimesTen Server, remove the comment symbol '#' in front of the `-server` *`portno`* entry.
- To disable the TimesTen Server, 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 system tells the TimesTen Server 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 take effect when the Server is restarted.

Specifying multiple connections to the TimesTen Server

By default, TimesTen creates only one connection to a Server for each child process. You can set multiple connects to a single TimesTen Server, either by using the Server connection attributes described in the "Server connection attributes" section in the *Oracle TimesTen In-Memory Database Reference* or by setting the TimesTen daemon options described in this section. These options enable 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.

Changes to TimesTen Server settings do not occur until the TimesTen server is restarted. To restart the Server, use the following command:

```
ttDaemonAdmin -restartserver
```

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 database, add the following line to the `ttendaemon.options` file:

```
-maxConnsPerServer NumberOfClientConnections
```

Note: If you are running a server in multithreaded mode and a server process fails, then all the client connections being handled by that process are terminated. Consider this when configuring the server for multithreaded mode and choose a value for the `maxConnsPerServer` attribute that balances your requirements for reliability and availability versus the more efficient resource usage of multithreaded mode.

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 how connections are distributed among the child server processes spawned for a server DSN

To specify the number of child server processes for a particular server DSN that will use round-robin connection distribution (when `-maxConnsPerServer >1`), 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.

By default (value=1), the first `maxConnsPerServer` client connection to a server DSN is assigned to a single child server process, the next `maxConnsPerServer connection` is assigned to a second child server process and so on.

If `-serversPerDSN` is set to *N* where `N > 1`, then the first `N * maxConnsPerServer` client connections to a server DSN are distributed in round robin fashion over *N* child server processes. If additional client connections beyond `N * maxConnsPerServer` are opened to the same server DSN, then those connections are assigned to new child server processes sequentially as for the case when `-serversPerDSN=1`.

Configuring the thread stack size of the child server processes

You generally should not need to set the `ServerStackSize` attribute. However, if the `ttcserver` process is getting repeatable Access Violations (Windows) or core dumps (Unix), you may consider increasing the `ServerStackSize` attribute.

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 `ServerStackSize` attribute is ignored if the maximum number of client connections per child server process is one, because the sole client connection is serviced by the main thread of the child server process. Consider setting the `ServerStackSize` attribute to 1024 KB if you are running complex SQL queries on your client connections.

Note: These changes to the TimesTen Server 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 system 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 database on the server. Before using shared memory as IPC verify that you have configured your system correctly. See "Installation prerequisites" in the *Oracle TimesTen In-Memory Database Installation Guide*.

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

If this entry is missing, the TimesTen Server 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 databases 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 database 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.

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, Namespace 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 "CREATE USER" 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 "TimesTen instance administrators and users groups" 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. When connecting over a client/server connection, the external user must be defined on the same host the client and server. Thus, in when using an external user, both the client and the server must be on the same host since the operating system provides the authentication of the 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 clear text 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 Database.

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 user name 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 user name 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 Tables" in the *Oracle TimesTen In-Memory Database System Tables and Views 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

If granted the appropriate privileges, you can use the `DROP USER` statement to drop users created in the database. You cannot drop the user in the following instances:

- You cannot drop the instance administrator.
- You cannot drop a user unless all objects owned by that user have first been deleted.
- You cannot drop a user if the user is currently connected to the database.

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

```
Command> drop user terry;
User dropped.
```

The following error occurs if you try to drop the instance administrator:

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

The following error occurs if user Pat tries to drop user Terry when Pat does not have the required `ADMIN` privilege:

```
Command> drop user terry;
15100: User PAT lacks privilege ADMIN
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 been 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:

- Call TimesTen built-in procedures.
- Run TimesTen command-line utilities.
- Initiate a connection with first connection attributes.
- Execute SQL statements.

Built-in procedure, utilities, and connection attributes are documented in the *Oracle TimesTen In-Memory Database Reference*. SQL statements are documented 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 4-13.

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 4-20. 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 then have unrestricted access to all database objects and are 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 "TimesTen 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 "The PUBLIC role" 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, or 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 does 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 "Privilege hierarchy" section 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 4-13.

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 4-20.

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

- Granting administrator privileges
- Granting ALL PRIVILEGES
- Granting privileges 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 4-20.

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 privileges 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 "Connection 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 4-18 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 4-13.

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 4-18.

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, PL/SQL procedure, PL/SQL function, PL/SQL package or synonym
- Dropping a table, view, materialized view, sequence, procedure, function, package or synonym
- Altering a table, view, materialized view, sequence, procedure, function or package

Creating a table, index, view, materialized view, sequence, PL/SQL procedure, PL/SQL function, PL/SQL package or synonym To create a table, view, materialized view, sequence, PL/SQL procedure, PL/SQL function, PL/SQL package, or synonym 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, synonym 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`, `CREATE ANY SYNONYM` and `CREATE ANY PROCEDURE`.

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, package or synonym 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, if a procedure is created in the `HR` schema named `Proc1` and if Pat is granted the `ALTER ANY PROCEDURE` privilege, Pat can successfully alter the procedure `HR.Proc1`.

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*.

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*.

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 4-18.

- 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 PL/SQL functions, procedures and packages
- Object privileges for synonyms

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 4-16.

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 Pat creates a view owned by Terry that references objects owned by Terry, 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, 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 the `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 notifies 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;
```

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

In order to create a materialized view, a user needs the `CREATE MATERIALIZED VIEW` privilege. If the user is creating a materialized view in some other user's schema, the user needs the `CREATE ANY MATERIALIZED VIEW` privilege.

The owner of the materialized view needs to have `CREATE TABLE` privilege as well as `SELECT` privileges on every detail table in that materialized view. However, the owner of the materialized view is automatically granted the `SELECT` privilege on the detail tables if previously granted a higher-level system privilege, such as `SELECT ANY TABLE` or `ADMIN`.

For a user to select from a materialized view that they do not own, the user needs to be granted the object privileges for materialized views, which include `SELECT`, `INDEX` and

REFERENCES. For more details on the privileges required, see the appropriate SQL statements in the *Oracle TimesTen In-Memory Database SQL Reference*.

Behavior of Invalid Materialized Views In order for the materialized view to be valid, the owner of the view must be granted and must keep the SELECT object privilege for all of the detail tables referenced by the materialized view. If the owner of an existing materialized view loses the SELECT privilege on any detail table on which the materialized view is based, the materialized view becomes invalid.

The status of the materialized view is provided in the STATUS column of the SYS.DBA_OBJECTS, SYS.ALL_OBJECTS, and SYS.USER_OBJECTS views. The owner of the materialized view can see the status of its materialized views in the USER_OBJECTS view. Alternatively, execute the ttIsql describe command, which appends INVALID to the materialized view when it becomes invalid.

Note: If the owner of the materialized view was granted with a higher-level system privilege, such as SELECT ANY TABLE or ADMIN, the owner loses the required SELECT privileges on the detail tables if the higher-level system privilege is revoked. At this point, the materialized view becomes invalid.

- Users may still select from an invalid asynchronous materialized view without error. However, users receive an error when selecting from an invalid synchronous materialized view.
- Users that have the privilege to do so can still update the detail tables of the materialized view. However, an invalid materialized view does not reflect these changes. In addition, for asynchronous materialized views, the materialized view log is not updated if no valid materialized views depend on the detail tables.
- REFRESH on an invalid synchronous materialized view fails with an error.
- If the owner of the materialized view has been re-granted the privilege that was previously revoked, a REFRESH on an invalid COMPLETE asynchronous materialized view succeeds and the asynchronous materialized view is now valid.
- In order to fix an invalid materialized view, you must grant the appropriate privileges to the owner of the materialized view and then drop and re-create the materialized view.

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 its 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 fail if the only privilege on the parent table is the REFERENCES privilege.

Object privileges for PL/SQL functions, procedures and packages

For a user to perform operations on PL/SQL functions, PL/SQL procedures or PL/SQL 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.

Object privileges for synonyms

For a user to create or drop private or public synonyms, the user must have the following privileges:

Table 4–1 Privileges for synonyms

Action	Required privilege
Create a private synonym in the user's own schema.	CREATE SYNONYM
Create a private synonym in another user's schema.	CREATE ANY SYNONYM
Create a public synonym.	CREATE PUBLIC SYNONYM
Drop a private synonym in the user's own schema.	No privilege needed.
Drop a private synonym in another user's schema.	DROP ANY SYNONYM
Drop a public synonym.	DROP PUBLIC SYNONYM

In addition, in order to use a synonym, the user must have the appropriate access privileges for the object that the synonym refers to. For example, if you create a synonym for a view, then to select from that view using the synonym, the user would need the SELECT privilege that is necessary to select from a view.

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.

Note: Passthrough does not require any privileges to be granted, since the privilege checking is performed by the Oracle Database with the user credentials. For details on passthrough, see "Setting a passthrough level" in the *Oracle TimesTen Application-Tier 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 group operations, 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, this user may perform any cache group operation. This privilege confers all cache group operation privileges, which are listed in the "Privilege hierarchy" section in the *Oracle TimesTen In-Memory Database SQL Reference*.

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

The privileges that the TimesTen users require depend on the types of cache group operations that you want to perform.

- To create a cache group, a user must be granted either the `CREATE CACHE GROUP` or `CREATE ANY CACHE GROUP` system privilege. In addition, the user must be granted either the `CREATE ANY TABLE` or `CREATE TABLE` privilege to create any underlying cache tables, depending on if the table is owned by the user or not.
- 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` privilege as appropriate. In addition, the user must be granted the `DROP ANY TABLE` privilege to drop any underlying cache tables, if the tables are not owned by the user.

Note: All cache group privileges are described in detail in the "Setting Up a Caching Infrastructure" chapter in the *Oracle TimesTen Application-Tier Database Cache User's Guide*.

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;
```

Note: Users with certain privileges must also be created on the Oracle database to own tables and to store cache grid information. The privileges required for the Oracle Database cache administration user and the TimesTen cache manager user for each cache operation are listed in the "Setting Up a Caching Infrastructure" chapter in the *Oracle TimesTen Application-Tier Database Cache User's Guide*.

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

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

Cache group object privileges

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`: Enables the user to flush a cache group owned by another user.

- `LOAD`: Enables the user to load a cache group owned by another user.
- `UNLOAD`: Enables the user to unload a cache group owned by another user.
- `REFRESH`: Enables 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 "Cache Group Operations" in the *Oracle TimesTen Application-Tier Database Cache User's Guide*.

Viewing user privileges

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

Table 4–2 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 Tables" in the *Oracle TimesTen In-Memory Database System Tables and Views 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 4-16 for more details.

5 Globalization Support

The following sections describe TimesTen globalization support features:

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 database. See "Choosing a database character set" on page 5-2 in this book and the "Supported character sets" section in the *Oracle TimesTen In-Memory Database Reference* for all 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 5-7.

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 database creation. You cannot alter the database character set after database creation, and there is no default value for `DatabaseCharacterSet`. See "Supported character sets" in the *Oracle TimesTen In-Memory Database Reference* for a list of supported character sets.

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

- What languages does the database 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 TimesTen Application-Tier Database Cache (TimesTen Cache) to cache Oracle database tables, or if you loading Oracle database data into a TimesTen table, you must create the database 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 are more efficient in terms of space requirements. However, single-byte character sets limit how many languages you can support.

Character sets and replication

All databases 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 length and byte length semantics are supported to resolve potential ambiguity regarding column length and storage size. Multibyte encoding character sets are supported, such as UTF-8 or `AL32UTF8`. Multibyte encodings require varying amounts of storage per character depending on the character. For example, a UTF-8 character may require from 1 to 4 bytes. If, for example, a column is defined as `CHAR (10)`, all 10 characters fit in this column regardless of character set encoding. However, for `UTF-8` character set encoding, up to 40 bytes are required.

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, where each are three bytes long, and 5 bytes for the English characters, where each 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, a non-Unicode application can 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 "NLS_SORT "in the *Oracle TimesTen In-Memory Database Reference*.

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 "NLS_SORT" in the *Oracle TimesTen In-Memory Database Reference*.

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');
```

Note: For materialized views and cache groups, TimesTen recommends that you explicitly specify the collating sequence using the NLSSORT() SQL function rather than using this attribute in the connection string or DSN definition.

For more details, see "NLS_SORT" in the *Oracle TimesTen In-Memory Database Reference*. 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

There are many SQL functions that operate on character strings, which are detailed in the following sections in the *Oracle TimesTen In-Memory Database SQL Reference*:

- "Character functions returning character values".
- "Character functions returning number values".
- "String functions".
- "LOB functions".
- "Conversion functions" (describing some functions that operate on character strings).
- "General comparison functions" (describing some functions that operate on character strings but not on LOB data types).

Setting globalization support attributes

The globalization support attributes are summarized in the following table:

Parameter	Description
`DatabaseCharacterSet`	Indicates the character encoding used by a database.
`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.
`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 database 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 database. 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 "Connection Attributes" in the *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 VARCHAR2 data types is limited to binary ordering. NLS_SORT=BINARY is the only acceptable collating sequence allowed.
- CHAR semantics are ignored. Characters are single-byte.
- UPPER and LOWER functions support ASCII characters only. Results for non-ASCII characters are undefined. TimesTen does not return an error.
- TIMESTEN8 is not supported in TimesTen Cache.
- There is no support for any LOB data types in TIMESTEN8.

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.

Note: For details on potential issues when migrating a TimesTen database that uses the TIMESTEN8 character set to a database with a different character set, see "Migration, Backup, and Restoration" in the *Oracle TimesTen In-Memory Database Installation Guide*.

Globalization support during migration

Globalization support may cause issues during migration. For complete details, see "Migration, Backup, and Restoration" in the *Oracle TimesTen In-Memory Database Installation Guide* and the description of "ttMigrate" in the *Oracle TimesTen In-Memory Database Reference*.

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:

- Database 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 database 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 database can be displayed.
- Optimizing database 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.

The following sections describe how the `ttIsql` utility is used to perform these types of tasks:

- Conditional control with the IF-THEN-ELSE command construct
- Loading data from an Oracle database into a TimesTen table
- Viewing and changing query optimizer plans
- Managing ODBC functions
- Error recovery with WHENEVER SQLERROR

For more information on `ttIsql` commands, see the "ttIsql" section in 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 database and purging log files.
- Initializing a database 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 database problems by examining database statistics.
- Any other database 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. The following example starts ttIsql in interactive mode and then connects to a TimesTen database by executing the `connect` command with the `MY_DSN` DSN.

```
C:\>ttIsql

Copyright (c) 1996-2013, Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

Command> connect MY_DSN;
Connection successful: DSN=MY_DSN;DataStore=E:\ds\MY_DSN;
DRIVER=E:\WINNT\System32\TTdv1122.dll;
(Default setting AutoCommit=1)

Command>
```

When connecting to the database using `ttIsql`, you can also specify the DSN or connection string on the `ttIsql` command line. The `connect` command is implicitly executed.

```
C:\>ttIsql -connstr "DSN=MY_DSN"
```

```
Copyright (c) 1996-2013, Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

connect "DSN=MY_DSN";
Connection successful: DSN=MY_DSN;DataStore=E:\ds\MY_DSN;
DRIVER=E:\WINNT\System32\TTdv1122.dll;
(Default setting AutoCommit=1)

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 looks like the following:

```
C:\>ttIsql -f create.sql -connstr "DSN=MY_DSN"

Copyright (c) 1996-2013, Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

connect "DSN=MY_DSN"
Connection successful: DSN=MY_DSN;DataStore=E:\ds\MY_DSN;
DRIVER=E:\WINNT\System32\TTdv1122.dll;
(Default setting AutoCommit=1)

run "create.sql"

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

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>
```

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 "ttIsql" section in the *Oracle TimesTen In-Memory Database Reference*.

```
Usage: ttIsql [-h | -help | -helpcmds | -helpfull | -V]
        ttIsql [-f <filename>]
               [-v <verbosity>]
               [-e <commands>]
               [-interactive]
               [-N <ncharEncoding>]
```

```
                [-wait]
                [{<DSN> | -connstr <connection_string>}]
```

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 always takes 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` automatically connects to a DSN called `MY_DSN`, turns off autocommit, and displays the size of the database, as shown below:

```
C:\>ttIsql

Copyright (c) 1996-2013, 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\TTdv1122.dll;
(Default setting AutoCommit=1)
Command> autocommit 0;
Command> dssize;
The following values are in KB:

  PERM_ALLOCATED_SIZE:      40960
  PERM_IN_USE_SIZE:         9453
  PERM_IN_USE_HIGH_WATER:   9453
  TEMP_ALLOCATED_SIZE:      32768
  TEMP_IN_USE_SIZE:         9442
  TEMP_IN_USE_HIGH_WATER:   9885
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 the ttIsql 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 the ttIsql '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 prints 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.
<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 prints 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.

Keystroke	Action
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 the ttIsql 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 the ttIsql 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;
```

Using the ttIsql edit command

You can use the `ttIsql edit` command to edit a file or edit `ttIsql` commands in a text editor. The `ttIsql edit` command starts a text editor such as `emacs`, `gedit`, or `vi`. For more information on changing the default text editor, see "Changing the default text editor for the ttIsql edit command" on page 6-9.

The syntax for the `ttIsql edit` command is as follows:

```
Command> edit [ file | !history_search_command ]
```

You can only use one parameter at a time. The `history_search_command` parameter is defined as the `!` character followed by the number of the command or a search string. If you do not specify a `!` character, the `ttIsql edit` command interprets the parameter as `file`. *`file`* is the name of the file that you want to edit. If you do not specify a parameter or specify `!!`, the last `ttIsql` command is edited.

When you specify a `file` parameter, the editor edits the specified file. If TimesTen does not find an exact file match for the specified `file` parameter in the current working directory, it searches for *`file`*`.sql`. If neither file exists, the editor creates the specified file in the current working directory. You can specify a path in the `file` parameter.

The following example edits the `new.sql` file:

```
Command> edit new.sql;
```

The following example edits the `new.sql` file in the `/scripts` directory:

```
Command> edit /scripts/new.sql;
```

If you execute the `ttIsql edit` command with a `file` parameter, `ttIsql` does not execute the contents of the file after you exit the editor.

You can edit a SQL statement that is stored in the history list of the current `ttIsql` session. When calling the `ttIsql edit` command specify the `!` character followed by the number of the command or a search string. The editor opens the `ttIsql` command in a temporary file that you can save in a preferred location. For more information on using the `ttIsql history` command, see "Using the ttIsql command history" on page 6-7.

The following example edits `ttIsql` command `2`:

```
Command> edit !2;
```

The following example searches for and edits the last `ttIsql` command that contains the search string `create`:

```
Command> edit !create;
```

The following example executes a `CREATE TABLE` statement and then uses the `edit` command to edit the `CREATE TABLE` statement in a text editor:

```
Command> CREATE TABLE t1
(c1 VARCHAR(10) NOT INLINE NOT NULL, c2 VARCHAR(144) INLINE NOT NULL);
Command> edit;
```

The prior example is equivalent to using the `ttIsql edit` command with the `!!` parameter:

```
Command> CREATE TABLE t1
(c1 VARCHAR(10) NOT INLINE NOT NULL, c2 VARCHAR(144) INLINE NOT NULL);
Command> edit !!;
```

If you execute the `ttIsql edit` command with a `history_search_command` parameter, `ttIsql` executes the contents of the file after you exit the editor. The contents of the file are executed as a single `ttIsql` command. If you do not want to execute the contents of the file, delete the contents of the file and save the file before you exit the editor.

Changing the default text editor for the ttIsql edit command

You can specify the default editor by defining the `ttIsql _EDITOR` define alias. The following example sets the default editor to `vi`:

```
Command> DEFINE _EDITOR=vi
```

If you do not define the `_EDITOR` define alias, `ttIsql` uses the editor specified by the `VISUAL` environment variable. If the `_EDITOR` define alias and the `VISUAL` environment variables are not set, `ttIsql` uses the editor specified by the `EDITOR` environment variable. When `_EDITOR`, `VISUAL`, and EDITOR are not set, `vi` is used for Unix and `notepad.exe` is used for Windows.

Working with character sets

The `ttIsql` utility supports the character sets listed in "Supported character sets" in the *Oracle TimesTen In-Memory Database Reference*. The ability of `ttIsql` to display characters depends on the native operating system 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.

Displaying database structure information

There are several `ttIsql` commands that display information on database structures. The most useful commands are summarized below:

- `describe` - Displays information on database objects.
- `cachegroups` - Displays the attributes of cache groups.
- `dssize` - Reports the current sizes of the permanent and temporary database memory regions.
- `tablesize` - Displays the size of tables that have been analyzed with the `ttComputeTabSizes` tool.
- `monitor` - Displays a summary of the current state of the database.

Using the ttIsql describe command

Use the `describe` command to display information on individual database objects. Displays parameters for prepared SQL statements and built-in procedures. The argument to the `describe` command can be the name of a table, cache group, view, materialized view, materialized view log, sequence, synonym, 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>
```

Using the ttIsql cachegroups command

The `cachegroups` command is used to provide detailed information on cache groups defined in the current database. 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;
Cache Group CACHEUSER.READCACHE:
 Cache Group Type: Read Only
 Autorefresh: Yes
 Autorefresh Mode: Incremental
 Autorefresh State: Paused
 Autorefresh Interval: 5 Seconds
 Autorefresh Status: ok
 Aging: No aging defined
 Root Table: ORATT.READTAB
 Table Type: Read Only
Cache Group CACHEUSER.WRITECACHE:
 Cache Group Type: Asynchronous Writethrough global (Dynamic)
 Autorefresh: No
 Aging: LRU on
 Root Table: ORATT.WRITETAB
 Table Type: Propagate
2 cache groups found.
```

Using the ttIsql dssize command

The `dssize` command is used to report the current memory status of the permanent and temporary memory regions as well as the maximum, allocated and in-use sizes for the database.

The following example uses the `k` option to print the database size information in KB:

```
Command> dssize k;
The following values are in KB:

  PERM_ALLOCATED_SIZE:      40960
  PERM_IN_USE_SIZE:         9742
  PERM_IN_USE_HIGH_WATER:   9742
  TEMP_ALLOCATED_SIZE:      32768
  TEMP_IN_USE_SIZE:         9442
  TEMP_IN_USE_HIGH_WATER:   9505
```

For more information on the `dssize` command, see "ttIsql" in the *Oracle TimesTen In-Memory Database Reference*.

Using the ttIsql tablesize command

The `tablesize` command displays the detailed analysis of the amount of space used by a table. Once you call the `ttComputeTabSizes` built-in procedure, which analyzes the table size of the indicated tables, the `tablesize` command displays the total size data for all analyzed tables.

Note: For more details, see "ttComputeTabSizes" in the *Oracle TimesTen In-Memory Database Reference*.

Executing the `tablesize` command with no arguments displays available sizing information for all tables that have had the `ttComputeTabSizes` computation run. When you provide a table as an argument, `tablesize` displays available sizing only for the indicated table.

The syntax for `tablesize` is as follows:

```
tablesize [[owner_name_pattern.]table_name_pattern]
```

The following example invokes the `ttComputeTabSizes` built-in procedure to calculate the table size of the `employees` table. Then, the `tablesize` command displays the sizing information gathered for the `employees` table.

```
Command> call ttComputeTabSizes('employees');
Command> tablesize employees;

Sizes of USER1.EMPLOYEES:

  INLINE_ALLOC_BYTES:   60432
  NUM_USED_ROWS:        107
  NUM_FREE_ROWS:        149
  AVG_ROW_LEN:          236
  OUT_OF_LINE_BYTES:    0
  METADATA_BYTES:       1304
  TOTAL_BYTES:          61736
  LAST_UPDATED:         2011-06-29 12:55:28.000000

1 table found.
```

These values provide insights into overhead and how the total space is used for the table.

For example:

- The `NUM_FREE_ROWS` value describes the number of rows allocated for the table, but not currently in use. Space occupied by free rows cannot be used by the system for storing other system objects or structures.
- Use the `TOTAL_BYTES` value to calculate how much permanent space your table occupies.
- `LAST_UPDATED` is the time of the last size computation. If you want a more recent computation, re-execute `ttComputeTabSizes` and display the new output.

You can find a description for each calculated value in the "SYS.ALL_TAB_SIZES" section in the *Oracle TimesTen In-Memory Database System Tables and Views Reference*.

Using the ttIsql monitor command

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 database was loaded into memory.

```
Command> monitor;
TIME_OF_1ST_CONNECT: Wed Apr 20 10:34:17 2011
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:
```

Listing database objects by object type

You can use `ttIsql` to list tables, indexes, views, sequences, synonyms, PL/SQL functions, procedures and packages in a database. Commands prefixed by `all` display all of this type of object. 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 database objects:

- `tables` and `alltables` - Lists tables.
- `indexes` and `allindexes` - Lists indexes.

- `views` and `allviews` - Lists views.
- `sequences` and `allsequences` - Lists sequences.
- `synonyms` and `allsynonyms` - Lists synonyms.
- `functions` and `allfunctions` - Lists PL/SQL functions.
- `procedures` and `allprocedures` - Lists PL/SQL procedures.
- `packages` and `allpackages` - Lists PL/SQL packages.

Note: For details on each of these commands, see the "ttIsql" section in the *Oracle TimesTen In-Memory Database Reference*.

The following example demonstrates the `procedures` and `allprocedures` commands. 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 database.

```
$ ttisql myDSN
Copyright (c) 1996-2013, 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) 1996-2013, 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 connection attributes

You can view and set connection attributes with the `ttIsql` `show` and `set` commands. For a list of the attributes that you can view and set with `ttIsql`, see "Connection 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;
```

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. This can also be set as an attribute of the `set` command.
- `commit` - Commits the current transaction.
- `commitdurable` - Commits the current transaction and ensures that the committed work is recovered in case of database failure.
- `rollback` - Rolls back the current transaction.
- `isolation` - Changes the transaction isolation level. This can also be set as an attribute of the `set` command.
- `sqlquerytimeout` - Specifies the number of seconds to wait for a SQL statement to execute before returning to the application. This can also be set as an attribute of the `set` command.

When starting `ttIsql`, the autocommit feature is turned on by default, even within a SQL script. In this mode, every SQL operation against the database is committed automatically. When autocommit is turned off, then automatic commit depends on the setting for the `DDLCommitBehavior` connection attribute and the user executing DDL. For more information, see "Relationship between autocommit and DDLCommitBehavior" on page 7-4.

To turn the autocommit feature off, execute the `ttIsql` `autocommit` command with an argument of 0. When autocommit is turned off, transactions must be committed or rolled back manually by executing the `ttIsql` `commit`, `commitdurable` or `rollback` commands. The `commitdurable` command ensures that the transaction's effect is preserved in case of database failure. If autocommit is off when `ttIsql` exits, any uncommitted statements are rolled back and reported by `ttIsql`.

The `ttIsql` `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 modified without an argument then the current isolation level is reported.

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

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

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

```
E:\>ttIsql
Copyright (c) 1996-2013, 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\TTdv1122.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.
```

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` automatically prompts 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\System32\TTdv1122.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 "ttIsql" section in 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\TTdv1122.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;
Command> showplan 1;
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:           RowLkRangeScan
  TBLNAME:             T1
  IXNAME:              T1
  INDEXED CONDITION:   T1.KEY = _QMARK_1
  NOT INDEXED:         <NULL>

The command succeeded.
Command> exec;
```

```
Executing prepared command id = 0.

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 '_QMARK_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*.

Using, declaring, and setting variables

The following sections describe how to declare, set and use bind variables in ttIsql:

- Declaring and setting bind variables
- Automatically creating bind variables for retrieved columns

Declaring and setting bind variables

You can declare and set variables and arrays in ttIsql that can be referenced in a SQL statement, SQL script, or PL/SQL block. The variables declared using the `variable` and `setvariable` command must be one of the following data types: `NUMBER`, `CHAR`, `NCHAR`, `VARCHAR2`, `NVARCHAR2`, `CLOB`, `NCLOB`, `BLOB`, or `REFCURSOR`. However, when binding arrays, Timesten supports only binding arrays of the `NUMBER`, `CHAR`, `NCHAR`, `VARCHAR2`, or `NVARCHAR2` data types.

Note: All variables that are declared exist for the life of the ttIsql session. However, if you declare a new variable with the same name, the new variable replaces the old variable.

The following examples declare bind variables with the `variable` or `var` command for a number, character string, and an array. Each is assigned to a value either when declared or by using the `setvariable` or `setvar` command.

Note: For details on the syntax for these commands, see "ttIsql" in the *Oracle TimesTen In-Memory Database Reference*.

```
Command> VARIABLE house_number NUMBER := 268;
Command> PRINT house_number;
HOUSE_NUMBER            : 268

Command> VARIABLE street_name VARCHAR2(15);
Command> SETVARIABLE street_name := 'Oracle Parkway';

Command> VARIABLE occupants[5] VARCHAR2(15);
Command> SETVARIABLE occupants[1] := 'Pat';
Command> SETVARIABLE occupants[2] := 'Terry';
Command> PRINT occupants;
OCCUPANTS            : ARRAY [ 5 ] (Current Size 2)
OCCUPANTS[1] : Pat
OCCUPANTS[2] : Terry
```

The following is an example of binding multiple values in an array using square brackets to delineate the values and commas to separate each value for the array:

```
Command> VARIABLE occupants[5] VARCHAR2(15) := ['Pat', 'Terry'];
Command> PRINT occupants;
OCCUPANTS : ARRAY [ 5 ] (Current Size 2)
OCCUPANTS[1] : Pat
OCCUPANTS[2] : Terry
```

When using array binds, PL/SQL enables you to bind each variable to a PL/SQL variable with the following declaration, where *`TypeName`* is any unique identifier for the PL/SQL data type and *`DataType`* can be specified as `CHAR`, `NCHAR`, `VARCHAR2`, or `NVARCHAR2`.

```
TYPE TypeName IS TABLE OF DataType(<precision>) INDEX BY BINARY_INTEGER;
```

If the variable is declared as array of `NUMBER`, you can bind it to a PL/SQL variable of the following data types: `NUMBER`, `INTEGER`, `FLOAT`, or `DOUBLE PRECISION`. To do so, use the appropriate declaration:

```
TYPE TypeName IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
TYPE TypeName IS TABLE OF INTEGER INDEX BY BINARY_INTEGER;
TYPE TypeName IS TABLE OF FLOAT INDEX BY BINARY_INTEGER;
TYPE TypeName IS TABLE OF DOUBLE PRECISION INDEX BY BINARY_INTEGER;
```

The following example declares the `occupants` `VARCHAR2` array, which is then declared and used within a PL/SQL block:

```
Command> VARIABLE occupants[5] VARCHAR2(15);
Command> SETVARIABLE occupants[1] := 'Pat';
Command> SETVARIABLE occupants[2] := 'Terry';
Command> DECLARE
       > TYPE occuname IS TABLE OF VARCHAR2(15) INDEX BY BINARY_INTEGER;
       > x occuname;
       > BEGIN
       > x := :occupants;
       > FOR LROW IN x.FIRST..x.LAST LOOP
       >   x(LROW) := x(LROW) || ' Doe';
       > END LOOP;
       > :occupants := x;
       > END;
       > /

PL/SQL procedure successfully completed.

Command> PRINT occupants;
OCCUPANTS            : ARRAY [ 5 ] (Current Size 2)
OCCUPANTS[1] : Pat Doe
OCCUPANTS[2] : Terry Doe
```

Automatically creating bind variables for retrieved columns

When you set `autovariables` on in ttIsql, TimesTen creates an automatic bind variable named after each column in the last fetched row. An automatic bind variable can be used in the same manner of any bind variable.

The following example selects all rows from the `employees` table. Since all columns are retrieved, automatic variables are created and named for each column. The bind variable contains the last value retrieved for each column.

```
Command> SET AUTOVARIABLES ON;
Command> SELECT * FROM employees;
...
< 204, Hermann, Baer, HBAER, 515.123.8888, 1994-06-07 00:00:00, PR_REP, 10000,
 <NULL>, 101, 70 >
< 205, Shelley, Higgins, SHIGGINS, 515.123.8080, 1994-06-07 00:00:00, AC_MGR,
12000, <NULL>, 101, 110 >
< 206, William, Gietz, WGIETZ, 515.123.8181, 1994-06-07 00:00:00, AC_ACCOUNT,
8300, <NULL>, 205, 110 >

Command> PRINT;
EMPLOYEE_ID          : 206
FIRST_NAME           : William
LAST_NAME            : Gietz
EMAIL                : WGIETZ
PHONE_NUMBER         : 515.123.8181
HIRE_DATE            : 1994-06-07 00:00:00
JOB_ID               : AC_ACCOUNT
SALARY               : 8300
COMMISSION_PCT       : <NULL>
MANAGER_ID           : 205
DEPARTMENT_ID        : 110
```

If you provide an alias for a column name, the automatic bind variable name uses the alias, rather than the column name.

```
Command> SET AUTOVARIABLES ON;
Command> SELECT employee_id ID, First_name SURNAME, last_name LASTNAME
 FROM employees;

ID, SURNAME, LASTNAME
...
< 204, Hermann, Baer >
< 205, Shelley, Higgins >
< 206, William, Gietz >
107 rows found.
Command> PRINT;
ID                   : 206
SURNAME              : William
LASTNAME             : Gietz
```

For any query that fetches data without a known named column, set `columnlabels on` to show the column names. The following example shows that the columns returns from `ttConfiguration` built-in procedure are `paramname` and `paramvalue`.

```
Command> SET AUTOVARIABLES ON;
Command> SET COLUMNLABELS ON;

Command> call TTCONFIGURATION('PLSQL');

PARAMNAME, PARAMVALUE
< PLSQL, 1 >
1 row found.

Command> IF :paramvalue = 1 THEN "e:PLSQL is enabled";
PLSQL is enabled
Command> IF NOT  :paramvalue = 1 THEN "e:PLSQL is not enabled";
```

You can also use the `describe` command to show the column names. The following example uses the describe command to display the column names for the `ttConfiguration` built-in procedure.

```
Command> DESCRIBE TTCONFIGURATION;

Procedure TTCONFIGURATION:
  Parameters:
    PARAMNAME                       TT_VARCHAR (30)
  Columns:
    PARAMNAME                       TT_VARCHAR (30) NOT NULL
    PARAMVALUE                      TT_VARCHAR (1024)

1 procedure found.
```

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 the *Oracle TimesTen In-Memory Database PL/SQL Developer's Guide* for more examples. For information on error handling in `ttIsql` for PL/SQL objects, see "Showing errors in ttIsql" in the *Oracle TimesTen In-Memory Database PL/SQL Developer's Guide*.

Passing data from PL/SQL 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 database is.

Create the `tt_space_info` PL/SQL procedure and use SQL to provide values for the `permpct`, `permmaxpct`, `temppct`, 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_size * 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> VARIABLE permpct NUMBER
Command> VARIABLE permpctmax NUMBER
Command> VARIABLE temppct NUMBER
Command> VARIABLE 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.
```

Conditional control with the IF-THEN-ELSE command construct

The `IF-THEN-ELSE` command construct enables you to implement conditional branching logic in a ttIsql session. The `IF` command tests a condition and decides whether to execute commands within the `THEN` clause or the optional `ELSE` clause. The commands executed can be SQL statements, SQL scripts, PL/SQL blocks, or TimesTen utilities.

Note: For details on the syntax of the `IF-THEN-ELSE` construct, see the "ttIsql" section in the *Oracle TimesTen In-Memory Database Reference*.

The following example creates and tests a bind variable to see if PL/SQL is enabled. It uses the `autovariables` command to create the bind variable from the result of the call to `ttConfiguration`. The value can be tested within the `IF-THEN-ELSE` conditional by testing the `paramvalue` variable.

Note: For more details on the `autovariables` command, see "Automatically creating bind variables for retrieved columns" on page 6-21.

```
Command> SET AUTOVARIABLES ON;
Command> CALL TTCONFIGURATION('PLSQL');
PARAMNAME, PARAMVALUE
< PLSQL, 1 >
```

```
1 row found.
Command> IF :paramvalue = 1 THEN "e:PLSQL is enabled"
> ELSE "e:PLSQL is not enabled";
PLSQL is enabled
```

The following example checks to see that the `employees` table exists. If it does not, it executes the SQL script that creates the `employees` table; otherwise, a message is printed out.

```
Command> IF 0 = "SELECT COUNT(*) FROM SYS.TABLES
       > WHERE TBLNAME LIKE 'employees';"
       > THEN "e:EMPLOYEES table already exists"
       > ELSE "@HR_CRE_TT.SQL;";
EMPLOYEES table already exists
```

Loading data from an Oracle database into a TimesTen table

You can load the results of a SQL query from a back-end Oracle database into a single table on TimesTen without creating a cache grid, cache group, and cache table to contain the results. TimesTen provides tools that execute a user-provided `SELECT` statement on the Oracle database and load the result set into a table on TimesTen.

The following are the major steps that are performed to accomplish this task:

1. Create a table with the correct columns and data types on TimesTen.
2. Provide a `SELECT` statement that is executed on the Oracle database to generate the desired result set.
3. Load the result set into the table on TimesTen.

TimesTen provides two methods to accomplish these tasks:

- The `ttIsql` utility provides the `createandloadfromoraquery` command that, once provided the TimesTen table name and the `SELECT` statement, automatically creates the TimesTen table, executes the `SELECT` statement on the Oracle database, and loads the result set into the TimesTen table. This command is described fully in "Use ttIsql to create a table and load SQL query results" on page 6-27.
- The `ttTableSchemaFromOraQueryGet` built-in procedure evaluates the user-provided `SELECT` statement to generate a `CREATE TABLE` statement that can be executed to create a table on TimesTen, which would be appropriate to receive the result set from the `SELECT` statement. The `ttLoadFromOracle` built-in procedure executes the `SELECT` statement on the Oracle database and loads the result set into the TimesTen table. These built-in procedures are described in "Use TimesTen built-in procedures to recommend a table and load SQL query results" on page 6-29.

Both methods require the following:

- Both the TimesTen and Oracle databases involved must be configured with the same national database character set.
- When you connect to the TimesTen database, the connection must contain the same connection attributes that are required when using cache groups, as follows:
  - The user name, which must be the same on both the TimesTen and Oracle databases

    Note: The correct privileges must be granted to these users on each database for the SQL statements that are executed on their behalf.

  - The correct passwords for each user as appropriate in the `PWD` and `OraclePWD` connection attributes
  - The `OracleNetServiceName` connection attributes that identifies the Oracle database instance

- For either method, the user provides the following:
  - The table name on the TimesTen database where the results of the SQL query is loaded. If the owner of the table is not provided, the table is created with the current user as the owner. The table name is not required to be the same name as the table name on the Oracle database against which the SQL statement is executed. This table does not require a primary key. If the table already exists, a warning is issued and the retrieved rows are appended to the table.
  - Optionally, the number of parallel threads that you would like to be used in parallel when loading the table with the result set. This defaults to four.
  - The SQL `SELECT` statement that is executed against the Oracle database to obtain the required rows. The tables specified within this `SELECT` statement must be fully qualified, unless the tables are within the schema of the current Oracle Database user. The query cannot have any parameter bindings.

    The `SELECT` list should contain either simple column references or column aliases. For example, any expressions in the `SELECT` list should be provided with a column alias. You can also use the column alias to avoid duplication of column names in the result table. For example, instead of using `SELECT C1+1 FROM T1`, use `SELECT C1 + 1 C2 FROM T1`, which would create a column named `C2`.

TimesTen evaluates the `SELECT` statement and uses the column names, data types, and nullability information to create the table on TimesTen into which the result set is loaded. The column names and data types (either the same or mapped) are taken from the tables on the Oracle database involved in the `SELECT` statement. However, other Oracle Database table definition information (such as `DEFAULT` values, primary key, foreign key relationships, and so on) are not used when creating the `CREATE TABLE` statement for the TimesTen table.

Note: If the evaluation returns any unsupported data types or if the query cannot be executed on the Oracle database, such as from a syntax error, a warning is logged and a comment is displayed for the unsupported column in the output. However, if the data type is not supported by TimesTen, you can cast the data type directly in the `SELECT` list to a TimesTen supported data type.

The load process does not check that the column data types and sizes in the TimesTen table match the data types and sizes of the result set. Instead, the insert is attempted and if the column data types cannot be mapped or the Oracle Database data from the SQL query exceeds the TimesTen column size, TimesTen returns an error.

The load is automatically committed every 256 rows. If an error is encountered during the load, it terminates the load, but does not roll back any committed transactions. Any errors returned from the Oracle database are reported in the same manner as when using cache groups.

Because you can use the `createandloadfromoraquery` command and the `ttLoadFromOracle` built-in procedure to load into an existing TimesTen table, the following restrictions apply:

- You cannot load into system tables, dictionary tables, temporary tables, detail tables of views, materialized view tables, materialized view log tables, or tables already in a cache group. In addition, you cannot use a synonym for the table name.
- If you load the result set into an existing table that is the referencing table (child table) of a foreign key constraint, the constraint is not validated. As a result, rows that are missing a parent row may be loaded. Instead, you should verify all foreign keys after the table is loaded.

The following sections provide more details on each individual method:

- Use ttIsql to create a table and load SQL query results
- Use TimesTen built-in procedures to recommend a table and load SQL query results

Use ttIsql to create a table and load SQL query results

The `ttIsql` utility provides the `createandloadfromoraquery` command, which takes a table name, the number of parallel threads, and a `SELECT` statement that is to be executed on the Oracle database as input parameters. From these parameters, TimesTen performs the following:

1. Evaluates the SQL query and creates an appropriate table, if not already created, with the provided table name where the columns are those named in the SQL query with the same (or mapped) data types as those in the Oracle Database tables from which the resulting data is retrieved.
2. Loads the results of the SQL query as executed on the Oracle database into this table. The call returns a single number indicating the number of rows loaded. Any subsequent calls to this command append retrieved rows to the table.

Note: See the `createandloadfromoraquery` command in "ttIsql" in the *Oracle TimesTen In-Memory Database Reference* for full details on syntax, requirements, restrictions, and required privileges.

The following `ttIsql` example connects providing the DSN, user name, password for the user on TimesTen, and the password for the same user name on the Oracle database. Then, it executes the `createandloadfromoraquery` command to evaluate the `SELECT` statement. The `employees` table is created on TimesTen with the same column names and data types as the columns and data types of the retrieved rows. Then, the table is populated with the result set from the Oracle database over two parallel threads.

```
ttisql -connstr "DSN=cachedb1_1122;UID=oratt;PWD=timesten;OraclePWD=oracle"

Copyright (c) 1996-2013, Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql
connect -connstr "DSN=mydb;UID=oratt;PWD=timesten;OraclePWD=oracle";
Connection successful: DSN=mydb;UID=oratt;
DataStore=/timesten/install/info/DemoDataStore/mydb;DatabaseCharacterSet=US7ASCII;
ConnectionCharacterSet=US7ASCII;DRIVER=/timesten/install/lib/libtten.so;
PermSize=40;TempSize=32;TypeMode=0;OracleNetServiceName=inst1;

(Default setting AutoCommit=1)
Command> createandloadfromoraquery employees 2 SELECT * FROM hr.employees;
Mapping query to this table:
    CREATE TABLE "ORATT"."EMPLOYEES" (
    "EMPLOYEE_ID" number(6,0) NOT NULL,
    "FIRST_NAME" varchar2(20 byte),
    "LAST_NAME" varchar2(25 byte) NOT NULL,
    "EMAIL" varchar2(25 byte) NOT NULL,
    "PHONE_NUMBER" varchar2(20 byte),
    "HIRE_DATE" date NOT NULL,
    "JOB_ID" varchar2(10 byte) NOT NULL,
    "SALARY" number(8,2),
    "COMMISSION_PCT" number(2,2),
    "MANAGER_ID" number(6,0),
    "DEPARTMENT_ID" number(4,0)
     )
Table employees created
107 rows loaded from oracle.
```

Execute the `DESCRIBE` command to show the new table:

Note: In this example, the table owner is not specified, so it defaults to the current user. In this example, the current user is `oratt`.

```
Command> DESCRIBE employees;

Table ORATT.EMPLOYEES:
  Columns:
    EMPLOYEE_ID                     NUMBER (6) NOT NULL
    FIRST_NAME                      VARCHAR2 (20) INLINE
    LAST_NAME                       VARCHAR2 (25) INLINE NOT NULL
    EMAIL                           VARCHAR2 (25) INLINE NOT NULL
    PHONE_NUMBER                    VARCHAR2 (20) INLINE
    HIRE_DATE                       DATE NOT NULL
    JOB_ID                          VARCHAR2 (10) INLINE NOT NULL
    SALARY                          NUMBER (8,2)
    COMMISSION_PCT                  NUMBER (2,2)
    MANAGER_ID                      NUMBER (6)
    DEPARTMENT_ID                   NUMBER (4)

1 table found.
(primary key columns are indicated with *)

Command> SELECT * FROM employees;
< 114, Den, Raphaely, DRAPHEAL, 515.127.4561, 2002-12-07 00:00:00, PU_MAN,
11000, <NULL>, 100, 30 >
< 115, Alexander, Khoo, AKHOO, 515.127.4562, 2003-05-18 00:00:00, PU_CLERK,
3100, <NULL>, 114, 30 >
…
< 205, Shelley, Higgins, SHIGGINS, 515.123.8080, 2002-06-07 00:00:00,
AC_MGR, 12008, <NULL>, 101, 110 >
< 206, William, Gietz, WGIETZ, 515.123.8181, 2002-06-07 00:00:00,
AC_ACCOUNT, 8300, <NULL>, 205, 110 >
107 rows found.
```

The following example uses the `createandloadfromoraquery` command to create the `oratt.emp` table on TimesTen and populate it in parallel over four threads with data from the `hr.employees` table on the Oracle database, where `employee_id` is less than 200.

```
Command> createandloadfromoraquery emp 4 SELECT * FROM hr.employees
 WHERE employee_id < 200;
Mapping query to this table:
    CREATE TABLE "ORATT"."EMP" (
    "EMPLOYEE_ID" number(6,0) NOT NULL,
    "FIRST_NAME" varchar2(20 byte),
    "LAST_NAME" varchar2(25 byte) NOT NULL,
    "EMAIL" varchar2(25 byte) NOT NULL,
    "PHONE_NUMBER" varchar2(20 byte),
    "HIRE_DATE" date NOT NULL,
    "JOB_ID" varchar2(10 byte) NOT NULL,
    "SALARY" number(8,2),
    "COMMISSION_PCT" number(2,2),
    "MANAGER_ID" number(6,0),
    "DEPARTMENT_ID" number(4,0)
     )

Table emp created
100 rows loaded from oracle.
```

Then, the following `createandloadfromoraquery` retrieves all employees whose id is > 200 and the result set is appended to the existing table in TimesTen. A warning tells you that the table already exists and that 6 rows were added to it.

```
Command> createandloadfromoraquery emp 4 SELECT * FROM hr.employees
 WHERE employee_id > 200;
Warning  2207: Table ORATT.EMP already exists
6 rows loaded from oracle.
```

A parallel load operation may take a long time to execute and you may want to cancel the operation. For more information on cancelling a parallel load operation, see "Cancel a parallel load operation" on page 6-31.

Use TimesTen built-in procedures to recommend a table and load SQL query results

While the `createandloadfromoraquery` command automatically performs all of the tasks for creating the TimesTen table and loading the result set from the Oracle database into it, the following two built-in procedures separate the same functionality into the following two steps:

1. The `ttTableSchemaFromOraQueryGet` built-in procedure evaluates the SQL query and generates the `CREATE TABLE` SQL statement that you can choose to execute. In order to execute this statement, the user should have all required privileges to execute the query on the Oracle database. This enables you to view the table structure without execution. However, it does require you to execute the recommended `CREATE TABLE` statement yourself.
2. The `ttLoadFromOracle` built-in procedure executes the SQL query on the back-end Oracle database and then loads the result set into the TimesTen table. It requires the TimesTen table name where the results are loaded, the Oracle Database SQL `SELECT` statement to obtain the required rows, and the number of parallel threads that you would like to be used in parallel when loading the table with this result set.

   The call returns a single number indicating the number of rows loaded. Any subsequent calls append the retrieved rows to the table.

Note: See "ttTableSchemaFromOraQueryGet" and "ttLoadFromOracle" in the *Oracle TimesTen In-Memory Database Reference* for full details on syntax, requirements, restrictions, and required privileges.

The following example connects providing the DSN, user name, password for the user on TimesTen, the password for a user with the same name on the Oracle database, and the `OracleNetServiceName` for the Oracle database instance. Then, it calls the `ttTableSchemaFromOraQueryGet` built-in procedure to evaluate the `SELECT` statement and return a recommended `CREATE TABLE` statement for the `employees` table. Finally, the example calls the `ttLoadFromOracle` built-in procedure to load the `employees` table with the result set from the Oracle database. The load is performed in parallel over four threads, which is the default.

Note: If `autocommit` is set to off, then the user must either commit or rollback manually after loading the table.

```
$ ttisql "DSN=mydb;uid=oratt;pwd=timesten;
OraclePwd=oracle;OracleNetServiceName=inst1"
Copyright (c) 1996-2013, Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.
connect "DSN=mydb;uid=oratt;pwd=timesten;
OraclePwd=oracle;OracleNetServiceName=inst1";
Connection successful: DSN=mydb;UID=oratt;
DataStore=/timesten/install/info/DemoDataStore/mydb;
DatabaseCharacterSet=US7ASCII;ConnectionCharacterSet=US7ASCII;
DRIVER=/timesten/install/lib/libtten.so;PermSize=40;TempSize=32;
TypeMode=0;OracleNetServiceName=inst1;
(Default setting AutoCommit=1)

Command> call ttTableSchemaFromOraQueryGet('hr','employees',
 'SELECT * FROM hr.employees');
< CREATE TABLE "HR"."EMPLOYEES" (
"EMPLOYEE_ID" number(6,0) NOT NULL,
"FIRST_NAME" varchar2(20 byte),
"LAST_NAME" varchar2(25 byte) NOT NULL,
"EMAIL" varchar2(25 byte) NOT NULL,
"PHONE_NUMBER" varchar2(20 byte),
"HIRE_DATE" date NOT NULL,
"JOB_ID" varchar2(10 byte) NOT NULL,
"SALARY" number(8,2),
"COMMISSION_PCT" number(2,2),
"MANAGER_ID" number(6,0),
"DEPARTMENT_ID" number(4,0)
 ) >
1 row found.

Command> CALL ttLoadFromOracle ('HR','EMPLOYEES','SELECT * FROM HR.EMPLOYEES');
< 107 >
1 row found.

Command> SELECT * FROM hr.employees;
< 100, Steven, King, SKING, 515.123.4567, 2003-06-17 00:00:00, AD_PRES, 24000,
<NULL>, <NULL>, 90 >
< 101, Neena, Kochhar, NKOCHHAR, 515.123.4568, 2005-09-21 00:00:00, AD_VP, 17000,
<NULL>, 100, 90 >
...
< 205, Shelley, Higgins, SHIGGINS, 515.123.8080, 2002-06-07 00:00:00, AC_MGR,
12008, <NULL>, 101, 110 >
< 206, William, Gietz, WGIETZ, 515.123.8181, 2002-06-07 00:00:00, AC_ACCOUNT,
8300, <NULL>, 205, 110 >
107 rows found.
```

The following example creates a table on the Oracle database, where `employee_id` is a column with a `PRIMARY KEY` constraints and `email` is a column with a `UNIQUE` constraint.

```
SQL> CREATE TABLE employees
    ( employee_id    NUMBER(6) PRIMARY KEY
    , first_name     VARCHAR2(20)
    , last_name      VARCHAR2(25) NOT NULL
    , email          VARCHAR2(25) NOT NULL UNIQUE
    , phone_number   VARCHAR2(20)
    , hire_date      DATE NOT NULL
    , job_id         VARCHAR2(10) NOT NULL
    , salary         NUMBER(8,2)
    , commission_pct NUMBER(2,2)
    , manager_id     NUMBER(6)
    , department_id  NUMBER(4)
    ) ;

Table created.
```

Then, the following `ttTableSchemaFromOraQueryGet` built-in procedure evaluates the SQL query and generates a `CREATE TABLE` SQL statement. Note that in the suggested `CREATE TABLE` SQL statement the `PRIMARY KEY` and `UNIQUE` constraints are not carried over from the Oracle database. Nullability constraints are carried over from the Oracle database. This also applies to the `createandloadfromoraquery` command.

```
Command> call ttTableSchemaFromOraQueryGet ('oratt', 'employees', 'select * from
oratt.employees');
< CREATE TABLE "ORATT"."EMPLOYEES" (
"EMPLOYEE_ID" number(6,0) NOT NULL,
"FIRST_NAME" varchar2(20 byte),
"LAST_NAME" varchar2(25 byte) NOT NULL,
"EMAIL" varchar2(25 byte) NOT NULL,
"PHONE_NUMBER" varchar2(20 byte),
"HIRE_DATE" date NOT NULL,
"JOB_ID" varchar2(10 byte) NOT NULL,
"SALARY" number(8,2),
"COMMISSION_PCT" number(2,2),
"MANAGER_ID" number(6,0),
"DEPARTMENT_ID" number(4,0)
 ) >
1 row found.
```

A parallel load operation may take a long time to execute and you may want to cancel the operation. For more information on cancelling a parallel load operation, see "Cancel a parallel load operation" on page 6-31.

Cancel a parallel load operation

You can cancel and cleanly stop all threads that are performing a parallel load operation with either the `SQLCancel(`*`hstmt`*`)` ODBC function or by pressing Ctrl-C in the `ttIsql` utility. For more information, see "Supported ODBC functions" in the *Oracle TimesTen In-Memory Database C Developer's Guide* and "Canceling ODBC functions" on page 6-38 in this book.

A parallel load operation periodically commits, so any successful operations are not rolled back. When you issue the cancel command, TimesTen performs the cancel operation:

- Before insert threads are spawned.
- After an insert batch commit (every 256 rows)
- After the main thread completes a fetch from the Oracle database.

To retry a cancelled parallel load operation, delete previously inserted rows from the TimesTen database to avoid duplicate rows.

Viewing and changing query optimizer plans

The following sections describe how to view the query optimizer plans, commands in the SQL command cache, or query plans for commands in the SQL command cache:

- Using the showplan command
- Viewing commands and explain plans from the SQL command cache

Using the showplan command

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 `ttIsql` 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 9-1.

- `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.
- `trytmprange` - Enables or disables the use of temporary range indexes.
- `tryrange` - 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 the `ttIsql autocommit` command to turn off autocommit.

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> 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> 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:             T2
  IXNAME:              <NULL>
  INDEXED CONDITION:   <NULL>
  NOT INDEXED:         <NULL>


  STEP:                2
  LEVEL:               3
  OPERATION:           TblLkSerialScan
  TBLNAME:             T3
  IXNAME:              <NULL>
  INDEXED CONDITION:   <NULL>
  NOT INDEXED:         T2.B = T3.C


  STEP:                3
  LEVEL:               2
  OPERATION:           NestedLoop
  TBLNAME:             <NULL>
  IXNAME:              <NULL>
  INDEXED CONDITION:   <NULL>
  NOT INDEXED:         <NULL>


  STEP:                4
  LEVEL:               2
  OPERATION:           TblLkSerialScan
  TBLNAME:             T1
  IXNAME:              <NULL>
  INDEXED CONDITION:   <NULL>
  NOT INDEXED:         T1.A = T2.B AND T1.A = T2.B


  STEP:                5
  LEVEL:               1
  OPERATION:           NestedLoop
  TBLNAME:             <NULL>
  IXNAME:              <NULL>
  INDEXED CONDITION:   <NULL>
  NOT INDEXED:         <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:           TmpRangeScan
  TBLNAME:             T2
  IXNAME:              <NULL>
  INDEXED CONDITION:   <NULL>
  NOT INDEXED:         <NULL>


  STEP:                2
  LEVEL:               3
  OPERATION:           RowLkSerialScan
  TBLNAME:             T3
  IXNAME:              <NULL>
  INDEXED CONDITION:   <NULL>
  NOT INDEXED:         T2.B = T3.C


  STEP:                3
  LEVEL:               2
  OPERATION:           NestedLoop
  TBLNAME:             <NULL>
  IXNAME:              <NULL>
  INDEXED CONDITION:   <NULL>
  NOT INDEXED:         <NULL>


  STEP:                4
  LEVEL:               2
  OPERATION:           RowLkSerialScan
  TBLNAME:             T1
  IXNAME:              <NULL>
  INDEXED CONDITION:   <NULL>
  NOT INDEXED:         T1.A = T2.B AND T1.A = T2.B


  STEP:                5
  LEVEL:               1
  OPERATION:           NestedLoop
  TBLNAME:             <NULL>
  IXNAME:              <NULL>
  INDEXED CONDITION:   <NULL>
  NOT INDEXED:         <NULL>

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

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` `ttIsql` 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, serial scans and nested loops.

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.

Viewing commands and explain plans from the SQL command cache

The following sections describe how to view commands and their explain plans:

- View commands in the SQL command cache
- Display query plan for statement in SQL command cache

View commands in the SQL command cache

The `ttIsql cmdcache` command invokes the `ttSqlCmdCacheInfo` built-in procedure to display the contents of the TimesTen SQL command cache. See "Displaying commands stored in the SQL command cache" on page 9-3 for full details on this procedure.

If you execute the `cmdcache` command without parameters, the full SQL command cache contents are displayed. Identical to the `ttSqlCmdCacheInfo` built-in procedure, you can provide a command ID to specify a specific command to be displayed.

In addition, the `ttIsql cmdcache` command can filter the results so that only those commands that match a particular owner or query text are displayed.

The syntax for the `cmdcache` command is as follows:

```
cmdcache [[by {sqlcmdid | querytext | owner}] <query_substring>
```

If you provide the `owner` parameter, the results are filtered by the owner, identified by the *<query_substring>*, displayed within each returned command. If you provide the `querytext` parameter, the results are filtered so that all queries are displayed that contain the substring provided within the *<query_substring>*. If only the *<query_substring>* is provided, such as `cmdcache` *<query_substring>*, the command assumes to filter the query text by the *<query_substring>*.

Display query plan for statement in SQL command cache

The `ttIsql explain` command displays the query plan for an individual command.

- If you provide a command ID from the SQL command cache, the `explain` command invokes the `ttSqlCmdQueryPlan` built-in procedure to display the query plan for an individual command in the TimesTen SQL command cache. If you want the explain plan displayed in a formatted method, execute the explain command instead of calling the `ttSqlCmdQueryPlan` built-in procedure. Both provide the same information, but the `ttSqlCmdQueryPlan` built-in procedure provides the data in a raw data format. See "Viewing query plans associated with commands stored in the SQL command cache" on page 9-9 for full details on the `ttSqlCmdQueryPlan` built-in procedure.
- If you provide a SQL statement or the history item number, the `explain` command compiles the SQL statements necessary to display the explain plan for this particular SQL statement.

The syntax for the `explain` command is as follows:

```
explain [plan for] {[<Connid>.]<ttisqlcmdid> | sqlcmdid <sqlcmdid> | <sqlstmt>
| !<historyitem>}
```

Identical to the `ttSqlCmdQueryPlan` built-in procedure, you can provide a command ID to specify a specific command to be displayed. The command ID can be retrieved with the `cmdcache` command, as described in "View commands in the SQL command cache" on page 6-35.

The following example provides an explain plan for command ID `38001456`:

```
Command> EXPLAIN SQLCMDID 38001456;

Query Optimizer Plan:
 Query Text: select * from all_objects where object_name = 'DBMS_OUTPUT'

  STEP:             1
  LEVEL:            12
  OPERATION:        TblLkRangeScan
  TABLENAME:        OBJ$
  TABLEOWNERNAME:   SYS
  INDEXNAME:        USER$.I_OBJ
  INDEXEDPRED:
  NONINDEXEDPRED:   (RTRIM( NAME ))  = DBMS_OUTPUT;NOT( 10 = TYPE#) ;
(  FLAGS ^ 128 = 0) ;


  STEP:             2
  LEVEL:            12
  OPERATION:        RowLkRangeScan
  TABLENAME:        OBJAUTH$
  TABLEOWNERNAME:   SYS
  INDEXNAME:        OBJAUTH$.I_OBJAUTH1
  INDEXEDPRED:      ( (GRANTEE#=1 )  OR (GRANTEE#=10 ) )  AND ( (PRIVILEGE#=8 ) )
  NONINDEXEDPRED:   OBJ# = OBJ#;


  STEP:             3
  LEVEL:            11
  OPERATION:        NestedLoop(Left OuterJoin)
  TABLENAME:
  TABLEOWNERNAME:
  INDEXNAME:
  INDEXEDPRED:
  NONINDEXEDPRED:
...
 STEP:             21
  LEVEL:            1
  OPERATION:        Project
  TABLENAME:
  TABLEOWNERNAME:
  INDEXNAME:
  INDEXEDPRED:
  NONINDEXEDPRED:

Command>
```

In addition, the `ttIsql explain` command can generate an explain plan for any SQL query you provide. For example, the following shows the explain plan for the SQL query: "`SELECT * FROM EMPLOYEES;`"

```
Command> EXPLAIN SELECT * FROM EMPLOYEES;
```

```
Query Optimizer Plan:

  STEP:                1
  LEVEL:               1
  OPERATION:           TblLkRangeScan
  TBLNAME:             EMPLOYEES
  IXNAME:              EMP_NAME_IX
  INDEXED CONDITION:   <NULL>
  NOT INDEXED:         <NULL>
```

You can also retrieve explain plans based upon the command history. The following example shows how you explain a previously executed SQL statement using the history command ID:

```
Command> SELECT * FROM all_objects WHERE object_name = 'DBMS_OUTPUT';
< SYS, DBMS_OUTPUT, <NULL>, 241, <NULL>, PACKAGE, 2009-10-13 10:41:11, 2009-10-13
10:41:11, 2009-10-13:10:41:11, VALID, N, N, N, 1, <NULL> >
< PUBLIC, DBMS_OUTPUT, <NULL>, 242, <NULL>, SYNONYM, 2009-10-13 10:41:11,
2009-10-13 10:41:11, 2009-10-13:10:41:11, INVALID, N, N, N, 1, <NULL> >
< SYS, DBMS_OUTPUT, <NULL>, 243, <NULL>, PACKAGE BODY, 2009-10-13 10:41:11,
2009-10-13 10:41:11, 2009-10-13:10:41:11, VALID, N, N, N, 2, <NULL> >
3 rows found.
Command> HISTORY;
1     connect "DSN=cache";
2     help cmdcache;
3     cmdcache;
4     explain select * from dual;
5     select * from all_objects where object_name = 'DBMS_OUTPUT';
Command> EXPLAIN !5;

Query Optimizer Plan:

  STEP:                1
  LEVEL:               10
  OPERATION:           TblLkRangeScan
  TBLNAME:             SYS.OBJ$
  IXNAME:              USER$.I_OBJ
  INDEXED CONDITION:   <NULL>
  NOT INDEXED:         O.FLAGS & 128 = 0 AND CAST(RTRIM (O.NAME) AS VARCHAR2(30
BYTE) INLINE) = 'DBMS_OUTPUT' AND O.TYPE# <> 10

  STEP:                2
  LEVEL:               10
  OPERATION:           RowLkRangeScan
  TBLNAME:             SYS.OBJAUTH$
  IXNAME:              OBJAUTH$.I_OBJAUTH1
  INDEXED CONDITION:   (OA.GRANTEE# = 1 OR OA.GRANTEE# = 10) AND OA.PRIVILEGE# = 8
  NOT INDEXED:         OA.OBJ# = O.OBJ#

  STEP:                3
  LEVEL:               9
  OPERATION:           NestedLoop(Left OuterJoin)
  TBLNAME:             <NULL>
  IXNAME:              <NULL>
  INDEXED CONDITION:   <NULL>
  NOT INDEXED:         <NULL>

  STEP:                4
  LEVEL:               9
  OPERATION:           TblLkRangeScan
  TBLNAME:              SYS.OBJAUTH$
  IXNAME:               OBJAUTH$.I_OBJAUTH1
  INDEXED CONDITION:    (OBJAUTH$.GRANTEE# = 1 OR OBJAUTH$.GRANTEE# = 10) AND
(OBJAUTH$.PRIVILEGE# = 2 OR OBJAUTH$.PRIVILEGE# = 3 OR OBJAUTH$.PRIVILEGE# = 4 OR
OBJAUTH$.PRIVILEGE# = 5 OR OBJAUTH$.PRIVILEGE# = 8)
  NOT INDEXED:          O.OBJ# = OBJAUTH$.OBJ#
...
 STEP:                 19
  LEVEL:                1
  OPERATION:            NestedLoop(Left OuterJoin)
  TBLNAME:              <NULL>
  IXNAME:               <NULL>
  INDEXED CONDITION:    <NULL>
  NOT INDEXED:          O.OWNER# = 1 OR (O.TYPE# IN (7,8,9) AND (NOT( ISNULLROW
(SYS.OBJAUTH$.ROWID)) OR NOT( ISNULLROW (SYS.SYSAUTH$.ROWID)))) OR (O.TYPE# IN
(1,2,3,4,5) AND NOT( ISNULLROW (SYS.SYSAUTH$.ROWID))) OR (O.TYPE# = 6 AND NOT(
ISNULLROW (SYS.SYSAUTH$.ROWID))) OR (O.TYPE# = 11 AND NOT( ISNULLROW
(SYS.SYSAUTH$.ROWID))) OR (O.TYPE# NOT IN (7,8,9,11) AND NOT( ISNULLROW
(SYS.OBJAUTH$.ROWID))) OR (O.TYPE# = 28 AND NOT( ISNULLROW (SYS.SYSAUTH$.ROWID)))
OR (O.TYPE# = 23 AND NOT( ISNULLROW (SYS.SYSAUTH$.ROWID))) OR O.OWNER# = 10
```

Managing ODBC functions

You can perform the following on ODBC functions within ttIsql:

- Canceling ODBC functions
- Timing ODBC function calls

Canceling ODBC functions

The ttIsql command attempts to cancel an ongoing ODBC function when the user presses Ctrl-C.

Timing ODBC function calls

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

For example, when executing the `ttIsql 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\ TTdv1122.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>
```

Error recovery with WHENEVER SQLERROR

Execute the `WHENEVER SQLERROR` command to prescribe what to do when a SQL error occurs. `WHENEVER SQLERROR` can be used to set up a recovery action for SQL statements, SQL script, or PL/SQL block.

By default, if a SQL error occurs while in ttIsql, the error information is displayed and ttIsql continues so that you can enter a new command. The default setting is `WHENEVER SQLERROR CONTINUE NONE`. You can also specify that ttIsql exits each time an error occurs, which may not be the best action for interactive use or when executing a SQL script or a PL/SQL block.

Note: For syntax of the `WHENEVER SQLERROR` command, see the "ttIsql" section in the *Oracle TimesTen In-Memory Database Reference*.

The following example uses `EXIT` to return an error code of 255 and executes a `COMMIT` statement to save all changes to the current connection before exiting ttIsql. The example retrieves the error code using the C shell `echo $status` command.

```
Command> WHENEVER SQLERROR EXIT 255 COMMIT;
Command> SELECT emp_id FROM employee;
 2206: Table PAT.EMPLOYEE not found
WHENEVER SQLERROR exiting.
$ echo $status
255
```

The following example demonstrates how the `WHENEVER SQLERROR` command can execute ttIsql commands or TimesTen utilities when an error occurs, even if the error is from another TimesTen utility:

```
Command> WHENEVER SQLERROR EXEC "DSSIZE;CALL TTSQLCMDCACHEINFOGET();";
Command> CALL TTCACHEPOLICYGET;
 5010: No OracleNetServiceName specified in DSN
The command failed.

DSSIZE;

  PERM_ALLOCATED_SIZE:      32768
  PERM_IN_USE_SIZE:         9204
  PERM_IN_USE_HIGH_WATER:   9204
  TEMP_ALLOCATED_SIZE:      40960
  TEMP_IN_USE_SIZE:         7785
  TEMP_IN_USE_HIGH_WATER:   7848

CALL TTSQLCMDCACHEINFOGET();

CMDCOUNT, FREEABLECOUNT, SIZE
< 10, 7, 41800 >
1 row found.
```

The following demonstrates the `SUPPRESS` command option. It suppresses all error messages and continues to the next command. The example shows that the error messages can be turned back on in the existing connection with another command option, which in this case is the `EXIT` command.

```
Command> WHENEVER SQLERROR SUPPRESS;
Command> SELECT *;
Command> WHENEVER SQLERROR EXIT;
Command> SELECT *;
 1001: Syntax error in SQL statement before or at: "", character position: 9
select *
        ^
WHENEVER SQLERROR exiting.
```

The following example sets a bind variable called `retcode`, the value of which is returned when a SQL error occurs:

```
Command> VARIABLE retcode NUMBER := 111;
Command> WHENEVER SQLERROR EXIT :retcode;
Command> INSERT INTO EMPLOYEES VALUES (
       > 202, 'Pat', 'Fay', 'PFAY', '603.123.6666',
       > TO_DATE ('17-AUG-1997', 'DD-MON-YYYY'),
       > 'MK_REP', 6000, NULL, 201, 20);
  907: Unique constraint (EMPLOYEES on PAT.EMPLOYEES) violated at Rowid
 <BMUFVUAAACOAAAAIiB>
WHENEVER SQLERROR exiting.
$ echo $status;
111
```

7 Transaction Management

TimesTen supports transactions that provide atomic, consistent, isolated and durable (ACID) access to data. The following sections describe how you can configure transaction features.

Transaction overview

All operations on a TimesTen database, even those that do not modify or access application data, are executed within a transaction. When running an operation and there is no outstanding transaction, one is started automatically on behalf of the application. Transactions are completed by an explicit or implicit commit or rollback. When completed, resources that were acquired or opened by the transaction are released and freed, such as locks and cursors.

Use the following SQL statements to commit or rollback your transaction:

- The SQL `COMMIT` statement commits the current transaction. Updates made in the transaction are made available to concurrent transactions.
- The SQL `ROLLBACK` statement rolls back the current transaction. All updates made in the transaction are undone.

Note: For the syntax of the `COMMIT` and `ROLLBACK` statements, see "SQL Statements" in the *Oracle TimesTen In-Memory Database SQL Reference*.

Read-only transactions do not require a commit. When executing write operations, complete transactions to release locks. When possible, keep write transactions short in duration. Any long-running transactions can reduce concurrency and decrease throughput because locks are held for a longer period of time, which blocks concurrent transactions. Also, long-running transactions can prevent transaction log files from being purged, causing these files to accumulate on disk.

A connection can have only one outstanding transaction at any time and cannot be explicitly closed if it has an open transaction.

Transaction implicit commit behavior

The following sections describe how you can configure whether the application enables implicit commit behavior or requires explicit commit behavior for DML or DDL statements:

- Transaction autocommit behavior
- TimesTen DDL commit behavior
- Relationship between autocommit and DDLCommitBehavior

Transaction autocommit behavior

Autocommit configures whether TimesTen issues an implicit commit after DML or DDL statements. By default, autocommit is enabled, following the ODBC and JDBC specifications.

When autocommit is on, the following behavior occurs:

- An implicit commit is issued immediately after a statement executes successfully.
- An implicit rollback is issued immediately after a statement execution fails, such as a primary key violation.
- If the statement generates a result set that opens a cursor, the automatic commit is not issued until that cursor and any other open cursors in the transaction have been explicitly closed. Any statements executed while a cursor is open is not committed until all cursors have been closed.

  Fetching all rows of a result set does not automatically close its cursor. After the result set has been processed, its cursor must be explicitly closed if using the read committed isolation level or the transaction must be explicitly committed or rolled back if using Serializable isolation level.

  Note: Even with durable commits and autocommit enabled, you could lose work if there is a failure or the application exits without closing cursors.

- If you are 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 are committed within this transaction. If an error occurs due to a problem on a particular row, only the successfully modified rows preceding the row with the error are committed in this transaction. The `pirow` parameter to the ODBC `SQLParamOptions` function contains the number of the rows in the batch that had a problem.

Commits can be costly for performance and intrusive if they are implicitly executed after every statement. TimesTen recommends you disable autocommit so that all commits are intentional. Disabling autocommit provides control over transactional boundaries, enables multiple statements to be executed within a single transaction, and improves performance, since there is no implicit commit after every statement.

If autocommit is disabled, transactions must be explicitly completed with a commit or rollback after any of the following:

- Completing all the work that was to be done in the transaction.
- Issuing a transaction-consistent (blocking) checkpoint request.
- Updating column and table statistics to be used by the query optimizer.
- Calling a TimesTen built-in procedure that does not generate a result set in order for the new setting specified in the procedure to take effect, such as the `ttLockWait` procedure.

You must establish a connection to a database before changing the autocommit setting. To disable autocommit, perform one of the following:

- In ODBC-based applications, execute `SQLSetConnectOption` function with `SQL_AUTOCOMMIT_OFF`.
- In JDBC applications, `Connection.setAutoCommit(false)` method.
- When running `ttIsql`, issue the `autocommit 0` command.

TimesTen DDL commit behavior

Traditionally, in TimesTen databases, DDL statements are executed as part of the current transaction and are committed or rolled back along with the rest of the transaction. However, the default behavior for the Oracle Database is that it issues an implicit `COMMIT` before and after any DDL statement.

You can configure for either behavior with the `DDLCommitBehavior` connection attribute, as follows:

- 0 - Oracle Database behavior. An implicit transaction commit is performed before the execution of each DDL statement and a durable commit is performed after the execution of each DDL statement. This is the default.
- 1 - Traditional TimesTen behavior. Execution of DDL statements does not trigger implicit transaction commits.

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`

The consequences of setting `DDLCommitBehavior=0` include the following:

- DDL changes cannot be rolled back.
- DDL statements delete records from global temporary tables unless the tables were created with the `ON COMMIT PRESERVE ROWS` clause.
- Tables created with the `CREATE TABLE . . . AS SELECT` statement are visible immediately.
- `TRUNCATE` statements are committed automatically. However, the truncate of the parent and child tables must be truncated in separate transactions, with the child table truncated first. You cannot truncate a parent table unless the child table is empty. The truncation of child and parent table can only be in the same transaction if you set `DDLCommitBehavior` to 1.

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

Relationship between autocommit and DDLCommitBehavior

Both autocommit and `DDLCommitBehavior` configure if and when implicit commits occur for SQL statements.

- Autocommit applies to both DDL and DML statements. Enabling for implicit commits of DDL statements overlaps in both options. If autocommit is enabled and `DDLCommitBehavior` is disabled, autocommit only commits after the DDL statement. However, if both autocommit and `DDLCommitBehavior` is enabled, an implicit commit occurs both before and after the DDL statement.
- To enable `DDLCommitBehavior`, you set the `DDLCommitBehavior` DSN attribute. To enable or disable autocommit, the application executes an ODBC function or JDBC method.

Table 7–1 shows what behavior occurs when you enable or disable one option in conjunction with the other:

Table 7–1 Relationship between autocommit and DDLCommitBehavior

Autocommit	DDLCommitBehavior	Relationship
ON	ON	All statements are automatically committed, unless you have an open cursor. DDL statements are implicitly committed before and after execution.
OFF	ON	Recommended setting. DDL statements are implicitly committed before and after execution. All other statements require an explicit commit.
ON	OFF	All statements are implicitly committed after execution, unless you have an open cursor. A commit is issued after the DDL is processed and not before.
OFF	OFF	All statements require an explicit commit, including DDL statements.

Ensuring ACID semantics

As a relational database, TimesTen is ACID compliant:

- **Atomic**: All TimesTen transactions are atomic: Either all database operations in a single transaction occur or none of them occur.
- **Consistent**: Any transaction can bring the database from one consistent state to another.
- **Isolated**: Transactions can be isolated. TimesTen has two isolation levels: read committed and serializable, which together with row level locking provide multi-user concurrency control.
- **Durable**: Once a transaction has been committed, it remains committed.

The following sections detail how TimesTen ensures ACID semantics for transactions:

- Transaction atomicity, consistency, and isolation
- Transaction consistency and durability

Transaction atomicity, consistency, and isolation

Locking and transaction logs are used to ensure ACID semantics as a transaction modifies data in a database as follows:

- **Locking**: TimesTen acquires locks on data items that the transaction writes and, depending on the transaction isolation level, data items that the transaction reads. See "Concurrency control through isolation and locking" on page 7-6.
- **Transaction logging**: All TimesTen transactions are atomic. Either all or none of the effects of the transaction are applied to the database. Modifications to the database are recorded in a transaction log. Atomicity is implemented by using the transaction log to undo the effects of a transaction if it is rolled back. Rollback can be caused explicitly by the application or during database recovery because the transaction was not committed at the time of failure. See "Transaction logging" on page 7-14.

The following table shows how TimesTen uses locks and transaction logs:

If	Then
Transaction is terminated successfully (committed)	■ Transaction log is posted to disk if the `DurableCommits` attribute is turned on. See "Transaction consistency and durability" on page 7-5 for more information. ■ Locks that were acquired on behalf of the transaction are released and the corresponding data becomes available to other transactions to read and modify. ■ All open cursors in the transaction are automatically closed.
Transaction is rolled back	■ Transaction 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. ■ All open cursors in the transaction are automatically closed.
System fails (data not committed)	■ On first connect, TimesTen automatically performs database recovery by reading the latest checkpoint image and applying the transaction log to restore the database to its most recent transactionally consistent state. See "Checkpoint operations" on page 7-9.
Application fails	■ All outstanding transactions are rolled back.

TimesTen supports temporary databases, which have essentially no checkpoints. However, they do have a transaction log so that transactions can be rolled back. Recovery is never performed for such databases. They are destroyed after a database or application shuts down or fails. For information on temporary databases, see "Database overview" on page 8-1.

Transaction consistency and durability

The TimesTen Data Manager provides consistency and durability with a combination of checkpointing and transaction logging.

- A checkpoint operation writes the current in-memory database image to a checkpoint file on disk that has the effect of making all transactions that have been committed at the time of the checkpoint operation consistent and durable.
- All transactions are logged to an in-memory transaction log buffer, which is written to disk in one of the following ways:

- – Guaranteed durability through a durable (synchronous) commit
- – Delayed durability through a non-durable (asynchronous) commit:

Note: Checkpointing and logging are further described in "Checkpoint operations" on page 7-9 and "Transaction logging" on page 7-14.

Concurrency control through isolation and locking

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.

The following sections describe transaction isolation and locking levels:

- Transaction isolation levels
- Locking granularities

Transaction isolation levels

Transaction isolation enables each active transaction to operate as if there were no other transactions active in the system. Isolation levels determine if row-level locks are acquired when performing read operations. When a statement is issued to update a table, locks are acquired to prevent other transactions from modifying the same data until the updating transaction completes and releases its locks.

The `Isolation` connection attribute sets the isolation level for a connection. Isolation levels have no effect if using database-level locking because transactions cannot be run concurrently. The isolation level cannot be changed in the middle of a transaction.

TimesTen supports the following two transaction isolation levels:

- *ANSI Read Committed isolation*: The read committed isolation level is the recommended mode of operation for most applications, and is the default mode. It enables transactions that are reading data to execute concurrently with a transaction that is updating the same data. TimesTen makes multiple versions of data items to allow non-serializable read and write operations to proceed in parallel.

  Read operations do not block write operations and write operations do not block read operations, even when they read and write the same data. Read operations do not acquire locks on scanned rows. Write operations acquire locks that are held until the transaction commits or rolls back. Readers share a committed copy of the data, whereas a writer has its own uncommitted version. Therefore, when a transaction reads an item that is being updated by another in-progress transaction, it sees the committed version of that item. It cannot see an uncommitted version of an in-progress transaction.

  Read committed isolation level provides for better concurrency at the expense of decreased isolation because of the possibility of non-repeatable reads or phantom rows within a transaction. If an application executes the same query multiple times within the same transaction, the commit of an update from another transaction may cause the results from the read operation to retrieve different results. A phantom row appears in modified form in two different reads, in the same transaction, due to early release of read locks during the transaction.

To set read committed isolation level, if previously modified since this is the default, do one of the following:

– ODBC applications execute the `SQLSetConnectOption` ODBC function with the `SQL_TXN_ISOLATION` flag set to `SQL_TXN_READ_COMMITTED`.

– Connect with `isolation=1` in the connection string.

– When using `ttIsql`, execute `ISOLATION 1` or `ISOLATION READ_COMMITTED`.

- *ANSI Serializable isolation*: All locks acquired within a transaction by a read or write operation are held until the transaction commits or rolls back. Read operations block write operations, and write operations block read operations. 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.

  Serializable isolation level provides for repeatable reads and increased isolation at the expense of decreased concurrency. A transaction that executes the same query multiple times within the same transaction 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.

  To set the isolation level to Serializable, do one of the following:

  – ODBC applications execute the `SQLSetConnectOption` ODBC function with the `SQL_TXN_ISOLATION` flag set to `SQL_TXN_SERIALIZABLE`.

  – Connect with `isolation=0` in the connection string.

  – When using `ttIsql`, execute `isolation 0` or `isolation serializable`.

  To ensure that materialized views are always in a consistent state, all view maintenance operations are performed under Serializable isolation, even when the transaction is in read committed isolation. This means that the transaction obtains read locks for any data items read during view maintenance. However, the transaction releases the read locks at the end of the `INSERT`, `UPDATE` or `CREATE VIEW` statement that triggered the view maintenance, instead of holding them until the end of the transaction.

Note: The `ttXactAdmin` utility generates a report showing lock holds and lock waits for all outstanding transactions. It can be used to troubleshoot lock contention problems where operations are being blocked, or encountering lock timeout or deadlock errors. It can also be used to roll back a specified transaction.

Locking granularities

TimesTen supports row-level locks, table-level locks and database-level locks:

Note: Different connections can coexist with different levels of locking, but the presence of even one connection using database-level locking leads to reduced concurrency. For performance information, see "Choose the best method of locking" on page 10-9.

- *Row-level locking*: Transactions usually obtain locks on the individual rows that they access. Row-level locking is the recommended mode of operation because it provides the finest granularity of concurrency control. It allows concurrent transactions to update different rows of the same table. However, row-level locking requires space in the database's temporary memory region to store lock information.

Row-level locking is the default. However, if it has been modified to another type of locking and you want to re-enable row-level locking, do one of the following:

– Set the `LockLevel` connection attribute to 0.
– Call the `ttLockLevel` built-in procedure with the *`lockLevel`* parameter set to `Row`. This procedure changes the lock level between row-level and database-level locking on the *next* transaction and for all subsequent transactions for this connection.
– Execute the `ttOptSetFlag` procedure to set the *`RowLock`* parameter to 1, which enables the optimizer to consider using row locks.

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

- *Table-level locking*: Table-level locking is recommended when concurrent transactions access different tables or a transaction accesses most of the rows of a particular table. Table-level locking provides better concurrency than database-level locking. Row-level locking provides better concurrency than table-level locking. Table-level locking requires only a small amount of space in the temporary memory region to store lock information.

  Table-level locking provides the best performance for the following:

  – Queries that access a significant number of rows of a table
  – When there are very few concurrent transactions that access a table
  – When temporary space is inadequate to contain all row locks that an operation, such as a large insert or a large delete, might acquire

  To enable table-level locking, execute the `ttOptSetFlag` procedure to set the *`TblLock`* parameter to 1, which enables the optimizer to consider using table locks. In addition, set *`RowLock`* to 0 so that the optimizer does not consider row-level locks.

  If both table-level and row-level locking are disabled, TimesTen defaults to 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. Even though table-level locking provides better performance than row-level locking because of reduced locking overhead, the optimizer often chooses row-level locking for better concurrency. For more information, see "ttOptSetFlag" in the *Oracle TimesTen In-Memory Database Reference*.

  Note: When multiple locks have been obtained within the same transaction, the locks are released sequentially when the transaction ends.

- *Database-level locking*: Database-level locking serializes all transactions, which effectively allows no concurrency on the database. When a transaction is started, it acquires an exclusive lock on the database, which ensures that there is no more than one active transaction in the database at any given time. It releases the lock when the transaction is completed.

Database-level locking often provides better performance than row-level locking, due to reduced locking overhead. In addition, it provides higher throughput than row-level locking when running a single stream of transactions such as a bulk load operation. However, its applicability is limited to applications that never execute multiple concurrent transactions. With database-level locking, every transaction effectively runs in ANSI Serializable isolation, since concurrent transactions are disallowed.

To enable database-level locking, do one of the following:

– Set the `LockLevel` connection attribute to 1.

– Call the `ttLockLevel` built-in procedure with the *`lockLevel`* parameter set to `DS`. This procedure changes the lock level between row-level and database-level locking on the *next* transaction and for all subsequent transactions for this connection.

Setting wait time for acquiring a lock

Set the `LockWait` connection attribute to the maximum amount of time that a statement waits to acquire a lock before it times out. The default is 10 seconds. For more information, see "LockWait" in *Oracle TimesTen In-Memory Database Reference*.

If a statement within a transaction waits for a lock and the lock wait interval has elapsed, an error is returned. After receiving the error, the application can reissue the statement.

Lock wait intervals are imprecise due to the scheduling of the database's managing subdaemon process to detect lock timeouts. This imprecision does not apply to zero-second timeouts, which are always immediately reported. The lock wait interval does not apply to blocking checkpoints.

The database's managing subdaemon process checks every two seconds to see if there is a deadlock in the database among concurrent transactions. If a deadlock occurs, an error is returned to one of the transactions involved in the deadlock cycle. The transaction that receives the error must rollback in order to allow the other transactions involved in the deadlock to proceed.

Checkpoint operations

A checkpoint operation saves the in-memory image of a database to disk files, known as checkpoint files. By default, TimesTen performs background checkpoints at regular intervals. Checkpointing may generate a large amount of I/O activity and have a long execution time depending on the size of the database and the number of database changes since the most recent checkpoint.

Note: Applications can programmatically initiate checkpoint operations. See "Setting and managing checkpoints" on page 7-11 for more details.

Temporary databases do not initiate checkpointing. See "Database persistence" on page 8-2 for more information on temporary databases.

The following sections describe checkpoint operations and how you can manage them:

- Purpose of checkpoints
- Usage of checkpoint files
- Types of checkpoints
- Setting and managing checkpoints

Purpose of checkpoints

A checkpoint operation has two primary purposes.

- Decreases the amount of time required for database recovery, because it provides a more up-to-date database image on which recovery can begin.
- Makes a portion of the transaction log unneeded for any future database recovery operation, typically allowing one or more transaction 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 a transaction log needed to recover a database has a direct impact on the amount of downtime seen by an application after a system failure. The removal of unneeded transaction log files is important because it frees disk space that can be used for new transaction log files. In addition, the fewer transaction log files you have, the less time is required to load a database into memory. If these files were never removed, they would eventually consume all available space in the transaction log directory's file system, causing database operations to fail due to log space exhaustion.

Usage of checkpoint files

Each TimesTen database has two checkpoint files, named `*dsname*.ds0` and `*dsname*.ds1`, where *`dsname`* is the database path name and file name prefix specified in the database DSN. During a checkpoint operation, TimesTen determines which checkpoint file contains the most recent consistent image and then writes the next in-memory image of the database to the other file. Thus, the two files contain the two most recent database images.

TimesTen uses the most recent consistent checkpoint file and the transaction log to recover the database to its most recent transaction-consistent state after a database shutdown or system failure. If any errors occur during this process, or if the more recent checkpoint image is incomplete, then recovery restarts using the other checkpoint file.

When the database is created, TimesTen creates three transaction log files named *`dsname`*`.res0`, *`dsname`*`.res1`, and *`dsname`*`.res2`. These files contain pre-allocated space that serve as reserved transaction log space. Reserved transaction log space allows for a limited continuation of transaction logging if the file system that holds the transaction log files becomes full. If the file system becomes full, transactions are prevented from writing any new log records. Transactions that attempt to write new log records are forced to rollback.

Types of checkpoints

TimesTen supports two types of database checkpoint operations:

- Fuzzy or non-blocking checkpoints
- Transaction-consistent checkpoints

Fuzzy or non-blocking checkpoints

Fuzzy checkpoints, or non-blocking checkpoints, allow transactions to execute against the database while the checkpoint is in progress. Fuzzy checkpoints do not obtain locks of any kind, and therefore have a minimal impact on other database activity. Because transactions may modify the database while a checkpoint operation is in progress, the resulting checkpoint file may contain both committed and uncommitted transactions. 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 database image.

To recover the database when the checkpoint files were generated from fuzzy checkpoint operations, TimesTen requires the most recent consistent checkpoint file and the transaction log to bring the database into its most recent transaction-consistent state.

Transaction-consistent checkpoints

Transaction-consistent checkpoints, also known as blocking checkpoints, obtain an exclusive lock on the database for a portion of the checkpoint operation, blocking all access to the database during that time. The resulting checkpoint image contains all committed transactions prior to the time the checkpoint operations acquired the exclusive lock on the database. Because no transactions can be active while the database lock is held, no modifications made by in-progress transactions are included in the checkpoint image.

TimesTen uses the most recent consistent checkpoint file to recover the database to transaction-consistent state at the time of the last successful checkpoint operation completed. It uses the transaction log files to recover the database to its most recent transaction-consistent state after a database shutdown or system failure.

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 database for which both checkpoint files are already up to date then the checkpoint request is ignored.

Setting and managing checkpoints

The default behavior for TimesTen checkpoints is as follows:

- TimesTen performs periodic fuzzy checkpoints in the background. You can modify this behavior. See "Configuring or turning off background checkpointing" on page 7-12 for more information.
- TimesTen performs a transaction-consistent checkpoint operation of a database just before the database is unloaded from memory. See "Transaction-consistent checkpoints" on page 7-11.

You can manage and monitor checkpoints with the following connection attributes and built-in procedures:

- `CkptFrequency` attribute
- `CkptLogVolume` attribute
- `CkptRate` attribute
- `CkptReadThreads` attribute
- `ttCkpt` built-in procedure

- `ttCkptBlocking` built-in procedure
- `ttCkptConfig` built-in procedure
- `ttCkptHistory` built-in procedure

The following sections describe how to manage checkpointing:

- Programmatically performing a checkpoint
- Configuring or turning off background checkpointing
- Displaying checkpoint history and status
- Setting the checkpoint rate
- Setting the number of checkpoint file read threads

Programmatically performing a checkpoint

By default, TimesTen performs periodic fuzzy checkpoints in the background. Therefore, applications rarely need to issue manual checkpoints. However, if an application wishes to issue a manual checkpoint, it can call the `ttCkpt` built-in procedure to request a fuzzy checkpoint or the `ttCkptBlocking` built-in procedure to request a transaction-consistent checkpoint.

Configuring or turning off background checkpointing

Using attributes or built-in procedures, you can configure TimesTen to checkpoint either when the transaction log files contain a certain amount of data or at a specific frequency.

To configure checkpointing in TimesTen, do the following:

Configure the `CkptFrequency` and `CkptLogVolume` connection attributes as follows:

- The `CkptFrequency` connection attribute controls how often, in seconds, that TimesTen performs a background checkpoint. The default is 600 seconds. Set the `CkptFrequency` connection attribute to 0 if you want to control background checkpointing with the `CkptLogVolume` connection attribute.
- The `CkptLogVolume` connection attribute controls how much data, in megabytes, that collects in the transaction log file between background checkpoints. By increasing this amount, you can delay the frequency of the checkpoint. The default is 0. Set the `CkptFrequency` connection attribute to 0 if you want to control background checkpointing with the `CkptLogVolume` connection attribute.

To turn off background checkpointing, set both the `CkptFrequency` and `CkptLogVolume` connection attributes to 0.

Alternatively, you can configure background checkpointing or turn it off by calling the `ttCkptConfig` built-in procedure. The values set by `ttCkptConfig` take precedence over those set with the connection attributes.

Note: For information on default values and usage, see "CkptFrequency", "CkptLogVolume", and "ttCkptConfig" in the *Oracle TimesTen In-Memory Database Reference*.

Displaying checkpoint history and status

Call the `ttCkptHistory` built-in procedure to display the information on the last eight checkpoints. You can monitor the progress of a running checkpoint with the `Percent_Complete` column.

Setting the checkpoint rate

By default, there is no limit to the rate at which checkpoint data is written to disk. You can use the `CkptRate` attribute or the `ttCkptConfig` built-in procedure to set the maximum rate at which background checkpoint data is written to disk. Checkpoints taken during recovery and final checkpoints do not honor this rate; in those situations, the rate is unlimited.

Note: See "CkptRate" and "ttCkptConfig" in 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 transaction 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.

If a running checkpoint appears to be progressing too slowly when you evaluate the progress of this checkpoint with the `Percent_Complete` column of the `ttCkptHistory` result set, 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.

Perform the following to calculate the checkpoint rate:

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, you should consider the following:

- 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.
- The above method may underestimate the actual checkpoint rate, 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.

Setting the number of checkpoint file read threads

By default, TimesTen reads checkpoint files serially with a single thread. Use the `CkptReadThreads` connection attribute to set the number of threads that TimesTen uses to read the checkpoint files when loading the database into memory.

When using *`n`* number of threads, TimesTen divides the checkpoint file into *`n`* portions of equal size. Each thread concurrently reads a portion of the file into memory. Once all threads are done reading their portion of the checkpoint file successfully, TimesTen checks the database for consistency.

Note: For more information, see "Set CkptReadThreads" on page 10-30 in this book, and "CkptReadThreads" in the *Oracle TimesTen In-Memory Database Reference*.

Transaction logging

TimesTen creates one transaction log for each database, which is shared by all concurrent connections. A transaction log record is created for each database update, commit, and rollback. However, transaction log records are not generated for read-only transactions. Log records are first written to the transaction log buffer, which resides in the same shared memory segment as the database. The contents of the log buffer are then subsequently flushed to the latest transaction log file on disk.

The transaction log is used to track all updates made within a transaction, so that those updates can be undone if the transaction is rolled back.

Transaction logging enables recovery of transactions from checkpoint files and the transaction log, which were committed from the time of the last checkpoint operation after a system failure. If the transaction is non-durable, any committed transactions in the log buffer that have not been flushed to disk would be lost in the event of a system failure.

The following sections describe how to manage and monitor the transaction log buffers and file:

- Managing transaction log buffers and files
- Monitoring accumulation of transaction log files

Managing transaction log buffers and files

The following describes how to configure transaction log buffers and files:

- Transaction log buffers: There is one transaction log buffer for each database and the size of the transaction log buffer can be configured using the `LogBufMB` DSN attribute. Each transaction log buffer can have multiple strands. The number of transaction log buffer strands is configured with the `LogBufParallelism` attribute.
- Transaction log files: The maximum size for the transaction log files are configured with the `LogFileSize` DSN attribute. The transaction log files are created in the same directory as the checkpoint files unless the `LogDir` attribute specifies a different location. The transaction log file names have the form *`ds_name`*`.log`*`n`*. The *`ds_name`* is the database path name that is specified by the `DataStore` DSN attribute and is provided within the database's DSN. The suffix *`n`* is the transaction log file number, starting at zero.

Note: For best performance, TimesTen recommends that applications use the `LogDir` attribute to place the transaction log files in a different physical device from the checkpoint files. If separated, I/O operations for checkpoints do not block I/O operations to the transaction log and vice versa.

TimesTen writes a message to the support log if the transaction log files and checkpoint files for your databases are on the same device.

Monitoring accumulation of transaction log files

It is important to verify at frequent intervals that there are no transaction log holds that could result in an excessive accumulation of transaction log files. If too many transaction log files accumulate and fill up available disk space, new transactions in the TimesTen database cannot begin until the transaction log hold is advanced and transaction log files are purged by the next checkpoint operation.

The following sections describe transaction log operations, log holds, and accumulation of log files:

- Purging transaction log files
- Log holds by TimesTen components or operations
- Monitoring log holds and log file accumulation

Purging transaction log files

Any transaction log file is kept until TimesTen determines it can be purged, which can occur under the following conditions:

- Transactions writing log records to the file have been committed or rolled back. These can be either local database transactions or XA transactions.
- Changes recorded in the file have been written to both checkpoint files.
- Changes recorded in the file have been replicated, if replication is enabled.
- Changes recorded in the file have been propagated to the Oracle database, if TimesTen Cache is used and configured for that behavior.
- Changes recorded in the file have been reported to XLA, if XLA is used.

Under normal TimesTen operating conditions, unneeded transaction log files are purged each time a checkpoint is initiated. A checkpoint can be initiated either through a configurable time interval with the `CkptFrequency` connection attribute, a configurable log volume with the `CkptLogVolume` connection attribute, or by calling the `ttCkpt` built-in function, which can be called either manually or in a background checkpointing application thread.

If you are running out of disk space because of log files accumulating, use the `CkptLogVolume` connection attribute instead of the `CkptFrequency` connection attribute. In addition, if you execute the `ttLogHolds` build-in procedure frequently, you can tell if log reclamation is blocked.

Note: To improve performance, locate your log files on a separate disk partition from the one on which the checkpoint files are located. The `LogDir` connection attribute determines where log files are stored. For more information, see "Managing transaction log buffers and files" on page 7-14 in this book or "LogDir" in the *Oracle TimesTen In-Memory Database Reference*.

See "Checkpointing" in the *Oracle TimesTen Application-Tier Database Cache Introduction* for general information. See "Configuring or turning off background checkpointing" on page 7-12 for more details on `CkptFrequency` and `CkptLogVolume`. Also, see the sections for "CkptFrequency", "CkptLogVolume", "ttCkpt", and "ttLogHolds" in the *Oracle TimesTen In-Memory Database Reference*.

Log holds by TimesTen components or operations

Several TimesTen components or operations can cause transaction log holds. A transaction log hold prevents log files, beyond a certain point, from being purged until they are no longer needed. In normal circumstances, the log hold position is regularly advanced and log files are purged appropriately. However, if operations are not functioning properly and the hold position does not advance, there can be an excessive accumulation of log files beyond the hold position that can no longer be purged, which eventually fills available disk space.

These components and operations include the following:

- Replication: There is a transaction log hold until the transmitting replication agent confirms that the log files have been fully processed by the receiving host.

  Possible failure modes include the following:

  – The network is down or there is a standby crash and replication is unable to deliver data to one or more subscribers. If necessary, the application can direct that logs no longer be held, then duplicate the master database to the standby when normal operations resume. Criteria for when to do this includes the amount of time required to duplicate, the amount of available disk space on the master for log files, and the transaction log growth rate.

  – The overall database transaction rate exceeds the ability of replication to keep the active and standby databases synchronized. An application can reduce the application transaction rate or the number of replicated tables.

  For more information, see "Improving Replication Performance" in the *Oracle TimesTen In-Memory Database Replication Guide* and "Troubleshooting Replication" in the *Oracle TimesTen In-Memory Database Troubleshooting Guide*.

- XLA: There is a transaction log hold until the XLA bookmark advances.

  A possible failure mode occurs when the bookmark becomes stuck, which can occur if an XLA application terminates unexpectedly or if it disconnects without first deleting its bookmark or disabling change-tracking. If a bookmark gets too far behind, the application can delete it. If the XLA reader process is still active, it must first be terminated, so that another XLA process can connect and delete the bookmark.

- Active standby pairs that replicate AWT cache groups: There is a transaction log hold until the replication agent confirms that the transaction corresponding to the log hold has been committed on the Oracle Database. With an active standby pair, the active database typically receives the confirmation from the standby database.

If the standby database is down, the replication agent receives confirmation from Oracle Database directly.

Possible failure modes include the following:

– Oracle Database is down or there is a lock or resource contention.

– The network is down, slow, or saturated.

– With an active standby pair, replication to the standby database falls behind. Check log holds on the standby database.

– The transaction rate to TimesTen exceeds the maximum sustainable rate that TimesTen can propagate to Oracle Database.

For more information, see "Monitoring AWT cache groups" in the *Oracle TimesTen Application-Tier Database Cache User's Guide* and "Troubleshooting AWT Cache Groups" in the *Oracle TimesTen In-Memory Database Troubleshooting Guide*.

- Cache groups configured with `AUTOREFRESH`: There is a transaction log hold until the replication agent on the active database confirms the log files have been fully processed by the standby database.

  Possible failure modes include the following:

  – Replication from the active database to the standby database is impacted because the standby database falls behind due to large workloads resulting from `AUTOREFRESH` mode.

  – The standby database is down or recovering, but has not been marked as `FAILED` through a call, initiated by either the user application or Oracle Clusterware, to the `ttRepStateSave` built-in procedure. The active database does not take over propagation to the Oracle Database until the state of the standby database is marked as `FAILED`. While the standby database is down or recovering, transaction log files are held for the Oracle Database.

  For more information, see "Monitoring autorefresh cache groups" in the *Oracle TimesTen In-Memory Database Troubleshooting Guide*.

- Incremental TimesTen backup: There is a transaction log hold until the backup completes.

  A possible failure mode can occur if the incremental backup falls too far behind the most recent entries in the transaction log. For example, ensure that an unexpected burst of transaction activity cannot fill up available transaction log disk space due to the backup holding a log file that is too old. An application can perform another incremental backup to work around this situation.

- Long-running transaction or XA transaction: There is a transaction log hold until the transaction completes.

  A possible failure mode can occur if an application transaction does not commit or roll back for a long time, so that it becomes necessary for the application to terminate the long-running transaction.

  If necessary, you can roll back a transaction using the `ttXactAdmin` utility with the `-xactIdRollback` option. See "ttXactAdmin" in *Oracle TimesTen In-Memory Database Reference*.

Monitoring log holds and log file accumulation

Options for periodic monitoring of excessive transaction log accumulation include the following:

- Call the `ttLogHolds` built-in procedure, which returns a result set with details of all log holds. The information includes the following, as applicable:
  - Log file number, the offset of the hold position, and the type of hold, which can be checkpoint, replication, backup, XLA, long-running transaction, or long-running XA transaction
  - Name of the checkpoint file for a checkpoint hold
  - Name of the subscriber and the parallel track ID it uses for replication
  - Backup path for a backup hold
  - Name of the persistent subscription and process ID of the last process to open it for XLA
  - Transaction ID for a long-running transaction
  - XA XID for a long-running XA transaction

  For more information, see "ttLogHolds" in the *Oracle TimesTen In-Memory Database Reference*.
- Call the `ttCkptHistory` built-in procedure to check the last several checkpoints to confirm none of the returned rows has a status of `FAILED`.

  For more information, see "ttCkptHistory" in the *Oracle TimesTen In-Memory Database Reference*.
- Check the `SYS.SYSTEMSTATS` table for operational metrics. Each transaction log file has a unique sequence number, which starts at 0 for the first log file and increments by 1 for each subsequent log file. The number of the current log file is available in `SYS.SYSTEMSTATS.log.file.latest`. The number of the oldest log file not yet purged is available in `SYS.SYSTEMSTATS.log.file.earliest`. You should raise an error or warning if the difference in the sequence numbers exceeds an inappropriate threshold.

  For more information, see "SYS.SYSTEMSTATS" in the *Oracle TimesTen In-Memory Database System Tables and Views Reference*.
- For XLA, check the `SYS.TRANSACTION_LOG_API` table that provides bookmark information, such as the process ID of the connected application, which could help diagnose the reason why a bookmark may be stuck or lagging.

  For more information, see "SYS.TRANSACTION_LOG_API" in the *Oracle TimesTen In-Memory Database System Tables and Views Reference*.

Durable options for logging transactions

The following sections describe durability options for logging transactions:

- Guaranteed durability
- Delayed durability
- Durable commit performance enhancements

Guaranteed durability

Durability is implemented with a combination of checkpointing and logging.

- Checkpoint files: A checkpoint operation writes the current database image to a checkpoint file on disk, which has the effect of making all transactions that committed before the checkpoint durable.

- Transaction log files: For transactions that committed after the last checkpoint, TimesTen uses conventional logging techniques to make them durable. As each transaction progresses, it records its database modifications in an in-memory transaction log. At commit time, the relevant portion of the transaction log is flushed to disk. This log flush operation makes that transaction, and all previously-committed transactions, durable.

  Control returns to the application after the transaction log data has been durably written to disk. A durably committed transaction is not lost even in the event of a system failure.

To enable guaranteed durability, set the `DurableCommits` attribute to 1.

Any recovery uses the last checkpoint image together with the transaction log to reconstruct the latest transaction-consistent state of the database.

Note: Committing a transaction durably makes that transaction and all previous transactions durable. Any non-durable transactions are no longer subject to loss in the event of a database failure, just as if it had originally been committed durably.

If most of your transactions commit durably, you may want to set the `LogFlushMethod` first connect attribute to 2. This connection attribute configures how TimesTen writes and synchronizes log data to transaction log files. For more information, see "Use durable commits appropriately" on page 10-27.

Delayed durability

In delayed durability mode, as in guaranteed durability mode, each transaction enters records into the in-memory transaction log as it makes modifications to the database. However, when a transaction commits in delayed durability mode, it does not wait for the transaction log to be posted to disk before returning control to the application. Thus, a non-durable transaction may be lost in the event of a database failure. However, they execute considerably faster than durable transactions. Eventually, transactions are flushed to disk by the database's subdaemon process or when the in-memory log buffer is full.

Applications request delayed durability mode by setting the `DurableCommits` attribute to 0. This is the default and the recommended option. Connections that use delayed durability can coexist with connections that use guaranteed durability.

Applications that wish to take advantage of the performance benefits of delayed durability mode, but 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.

Durable commit performance enhancements

The performance cost for durable commits can be reduced with a group commit of multiple concurrently executing transactions. Many threads executing at the same time, if they are short transactions, may commit at almost the same time. Then, 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. Each connection spends more time waiting to commit than it spends using the CPU. Alternatively, applications that perform infrequent durable commits cause each connection to be very CPU-intensive for the TimesTen portion of its workload.

Applications that do not require optimal response time and can tolerate some transaction loss may elect to perform periodic durable commits. This maintains a smaller window of vulnerability to transaction loss as opposed to all transactions being committed non-durably. By committing only every *n*th transaction durably or performing a durable commit every *n* seconds, an application can achieve a quicker response time while maintaining a small window of vulnerability to transaction loss. A user can elect to perform a durable commit of a critical transaction, such as one that deals with financial exchange, that cannot be vulnerable to loss.

To enable periodic durable commits, an application does the following:

1. Connects with setting the attribute `DurableCommits=0`. This causes the transactions to commit non-durably.
2. When a durable commit is needed, the application can call the `ttDurableCommit` built-in procedure before committing. The `ttDurableCommit` built-in procedure does not actually commit the transaction; it merely causes the commit to be durable when it occurs.

Transaction reclaim operations

After a transaction is marked by TimesTen as committed, there is a *reclaim* phase of the commit during which database resources are reclaimed. This section discusses these reclaim operations, covering the following topics:

- About reclaim operations
- Configuring the commit buffer for reclaim operations

About reclaim operations

TimesTen resource cleanup occurs during the reclaim phase of a transaction commit. Consider a transaction with `DELETE` operations, for example. The SQL operation marks the deleted rows as deleted, but the space and resources occupied by these rows are not actually freed until the reclaim phase of the transaction commit.

During reclaim, TimesTen reexamines all the transaction log records starting from the beginning of the transaction to determine the reclaim operations that must be performed, then performs those operations.

To improve performance, a number of transaction log records can be cached to reduce the need to access the transaction log on disk. This cache is referred to as the *commit buffer* and its size is configurable, as described in the next section, "Configuring the commit buffer for reclaim operations".

Notes:

- Once the reclaim phase has begun, the transaction is considered to be committed and can no longer be rolled back.
- If a process is terminated during the reclaim phase, the cleanup operation will complete the reclaim.

Configuring the commit buffer for reclaim operations

The reclaim phase of a large transaction commit results in a large amount of processing and is very resource intensive. (For this reason, smaller transactions are generally recommended.) You can improve performance, however, by increasing the maximum size of the commit buffer, which is the cache of transaction log records used during reclaim operations.

You can use the TimesTen `CommitBufferSizeMax` connection attribute to specify the maximum size of the commit buffer, in megabytes. This setting has the scope of your current session. For efficiency, initial memory allocation will be significantly less than the maximum, but will automatically increase as needed in order to fit all the relevant log records into the commit buffer, until the allocation reaches the maximum. The allocation is then reduced back to the initial allocation after each reclaim phase. By default, the maximum is 128 KB with an initial allocation of 16 KB. (Also see "CommitBufferSizeMax" in *Oracle TimesTen In-Memory Database Reference*.)

Be aware that an increase in the maximum size of the commit buffer may result in a corresponding increase in temporary space consumption. There is no particular limit to the maximum size you can specify, aside from the maximum value of an integer, but exceeding the available temporary space will result in an error.

Note the following related features:

- During the course of a session, you can use `ALTER SESSION` to change the maximum size of the commit buffer as follows, where *n* is the desired maximum, in megabytes. (Also see "ALTER SESSION" in *Oracle TimesTen In-Memory Database SQL Reference*.)

  ```
  ALTER SESSION SET COMMIT_BUFFER_SIZE_MAX = n
  ```

- You can use the `ttCommitBufferStats` built-in procedure to gather statistics for your connection to help you tune the commit buffer maximum size. This built-in takes no parameters and returns the total number of commit buffer overflows and the highest amount of memory used by reclaim operations for transaction log records, in bytes. If there are buffer overflows, you may consider increasing the commit buffer maximum size. If there are no overflows and the highest amount of memory usage is well under the commit buffer maximum size, you may consider decreasing the maximum size.

  The `ttCommitBufferStatsReset` built-in procedure resets these statistics to 0 (zero). This is useful, for example, if you have set a new value for the commit buffer maximum size and want to restart the statistics.

  (Also see "ttCommitBufferStats" and "ttCommitBufferStatsReset" in *Oracle TimesTen In-Memory Database Reference*.)

- The system-wide number of commit buffer overflows is also recorded in the TimesTen statistic `txn.commits.buf.overflowed` in the `SYS.SYSTEMSTATS` table. (Also see "SYS.SYSTEMSTATS" in *Oracle TimesTen In-Memory Database System Tables and Views Reference*.)

- You can check the current setting of `CommitBufferSizeMax` by calling the `ttConfiguration` built-in procedure.

Recovery with checkpoint and transaction log files

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

During recovery, the latest checkpoint file is read into memory. All transactions that have been committed since the last checkpoint and whose log records are on disk are rolled forward from the appropriate transaction log files. Note that such transactions include all transactions that were committed durably as well as all transactions whose log records aged out of the in-memory log buffer. Uncommitted or rolled-back transactions are not recovered. For details on checkpoint and transaction log files, see "Checkpoint operations" on page 7-9 and "Transaction logging" on page 7-14.

8 Working with Data in a TimesTen Database

This chapter provides detailed information on the basic components in a TimesTen database 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 an application, see the appropriate TimesTen developer's guide.

This chapter includes the following topics:

- Database overview
- Understanding tables
- Understanding views
- Understanding materialized views
- Understanding indexes
- Understanding rows
- Understanding synonyms

Database overview

The following sections describe the main TimesTen database elements and features:

- Database components
- Database users and owners
- Database persistence

Database components

A TimesTen database has the following permanent components:

- **Tables**. The primary components of a TimesTen database are the tables that contain the application data. See "Understanding tables" on page 8-3.
- **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 8-12.
- **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 8-11.

- **Indexes**. Indexes on one or more columns of a table may be created for faster access to tables. See "Understanding indexes" on page 8-21.
- **Rows**. Every table consists of 0 or more rows. A row is a formatted list of values. See "Understanding rows" on page 8-30.
- **System tables**. System tables contain TimesTen metadata, such as a table of all tables. See "System Tables" in the *Oracle TimesTen In-Memory Database System Tables and Views Reference*.

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

Database users and owners

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 database. 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 databases 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.

Database persistence

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

Note: You can define database persistence by setting the value of the `Temporary` connection attribute. You cannot change the `Temporary` attribute on a database after it is created. For more information on the `Temporary` attribute, see "Temporary" in the *Oracle TimesTen In-Memory Database Reference*.

- **Permanent databases** are stored to disk automatically through a procedure called checkpointing. TimesTen automatically performs background checkpoints based on the settings of the connection attributes `CkptFrequency` and `CkptLogVolume`. TimesTen also checkpoints the database when the last application disconnects. Applications can also checkpoint a database directly to disk by calling the `ttCkptBlocking` built-in procedure. For more information, see "ttCkptBlocking" in the *Oracle TimesTen In-Memory Database Reference*.
- **Temporary databases** are not stored to disk. A temporary database is automatically destroyed when no applications are connected to it; that is, when the last connection disconnects or when there is a system or application failure. TimesTen removes all disk-based files when the last application disconnects.

  A temporary database cannot be backed up or replicated. Temporary databases are never fully checkpointed to disk, although Checkpoint operations can have significant overhead for permanent databases, depending on database size and activity, but have very little impact for temporary databases. Checkpoints are still necessary to remove transaction log files.

However, temporary databases do have a transaction log, which is periodically written to disk, so transactions can be rolled back. The amount of data written to the transaction log for temporary databases is less than that written for permanent databases, allowing better performance for temporary databases. Recovery is never performed for temporary databases.

You can increase your performance with temporary databases. If you do not need to save the database to disk, you can save checkpoint overhead by creating a temporary database.

Details for setting up a temporary database are described in "Setting up a temporary database" on page 1-17.

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
- Inline 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.

Inline 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`, `NVARCHAR2` and `VARCHAR2` column of a table as either *inline* or *out-of-line*.

- An inline column has a fixed length. All values of fixed-length columns of a table are stored row wise.
- A not inline column (also referred to as an out-of-line column) has a varying length. Some `VARCHAR2`, `NVARCHAR2` or `VARBINARY` data type columns are stored out-of-line. Out-of-line columns are not stored contiguously with the row but are allocated. By default, TimesTen stores `VARCHAR2`, `NVARCHAR2` and `VARBINARY` columns whose declared column length is > 128 bytes as out-of-line. In addition, all LOB data types are stored out-of-line. By default, TimesTen stores variable-length columns whose declared column length is <= 128 bytes as inline.

Most operations are slightly slower when performed on an out-of-line column instead of an inline column. There are several performance considerations when you use out-of-line columns instead of inline columns:

- Accessing data is slower because TimesTen does not store data from out-of-line columns contiguously with the row.
- Populating data is slower because TimesTen generates more logging operations.
- Deleting data is slower because TimesTen performs more reclaim and logging operations. If you are deleting a large number of rows (100,000 or more) consider using multiple smaller `DELETE FROM` statements, a `TRUNCATE TABLE` statement, or the `DELETE FIRST` clause. For more information, see "Avoid large DELETE statements" on page 10-29.
- Storing a column requires less overhead.

The maximum sizes of inline and out-of-line portions of a row are listed in "Using the ttIsql tablesize command" on page 6-11.

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 "CREATE TABLE" 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, TimesTen defines the owner as the user who created the table. Tables created by TimesTen, such as system tables, have the owner name `SYS`.

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, Namespace 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 10-12. You can use optimizer hints (statement or transaction level) to tune the TimesTen execution plan for a specific application. For more information on optimizer hints, see "Use optimizer hints to modify the execution plan" on page 9-14.

Primary keys, foreign keys and unique indexes

You can create a primary key on one or more columns 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 improve access speeds through the primary key. Once a row is inserted, its primary key columns cannot be modified, except to change a range index to a hash index.

Note: Indexes are discussed in "Understanding indexes" on page 8-21.

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 database contains system tables. System tables contain TimesTen metadata such as descriptions of all tables and indexes in the database, 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 "System Tables" chapter in the *Oracle TimesTen In-Memory Database System Tables and Views 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 8–1 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 8–2 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 8-5.

The `UNIQUE HASH ON custId PAGES` value indicates that there are 30 pages in the hash index. This means that the expected number of rows in the table is 30 * 256 = 7680. If the table ends up with significantly more rows than this, performance can be degraded, and the hash index should be resized. For more details on pages in a hash index, see information for `SET PAGES` in the "ALTER TABLE" section in the *Oracle TimesTen In-Memory Database SQL Reference*. For details on how to size pages in a hash table, see "Size hash indexes appropriately" on page 10-17.

```
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 8–3 Drop a table

The following example drops the table `NameID`.

```
DROP TABLE NameID;
```

Estimating table size

Increasing the size of a TimesTen database can be done on first connect. To avoid having to increase the size of a database, it is important not to underestimate the eventual database size. Use the `ttSize` utility to estimate table size.

The following example shows that the `ttSize` utility estimates the rows, inline row bytes, size of any indexes on the table, and the total size of the table:

```
ttSize -tbl Pat.tab1 MyDb

Rows = 2

Total in-line row bytes = 17524
Indexes:

 Bitmap index PAT.BITMAP_ID adds 6282 bytes
  Total index bytes = 6282

Total = 23806
```

You can also calculate the size of an existing table with the `ttIsql tablesize` command. For more information, see "Using the ttIsql tablesize command" on page 6-11.

Implementing aging in your tables

You can define an aging policy for one or more tables in your database. 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 database 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 database, 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 in a cache group" in the *Oracle TimesTen Application-Tier Database Cache User's Guide*.

This section includes the following topics:

- Usage-based aging
- Time-based aging
- Aging and foreign keys

- Scheduling when aging starts
- Aging and replication

Usage-based aging

Usage-based aging enables you to maintain the amount of memory used in a database 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`.

Call the `ttAgingLRUConfig` built-in procedure to specify the LRU aging attributes. The attribute values apply to all tables in the database 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 database `PermSize` at which LRU aging is deactivated.
HighUsageThreshhold	The percent of the database `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.

Call the `ttAgingScheduleNow` built-in procedure to schedule when aging starts. For more information, see "Scheduling when aging starts" on page 8-10.

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, also called 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.

Call the `ttAgingScheduleNow` built-in procedure to schedule when aging starts. For more information, see "Scheduling when aging starts" on page 8-10.

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 is recently accessed, then neither the parent row nor the child row is 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 are deleted.
- If a table has `ON DELETE CASCADE` enabled, the setting is ignored.

Scheduling when aging starts

Call 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 begins 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` starts or resets aging on all of the tables in the database that have aging defined.

Aging and replication

For active standby pairs, implement aging on the active master database. Deletes that occur as a result of aging are replicated to the standby master database and the read-only subscribers. If a failover to the standby master database occurs, aging is enabled on the database 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 *non-materialized 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 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 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

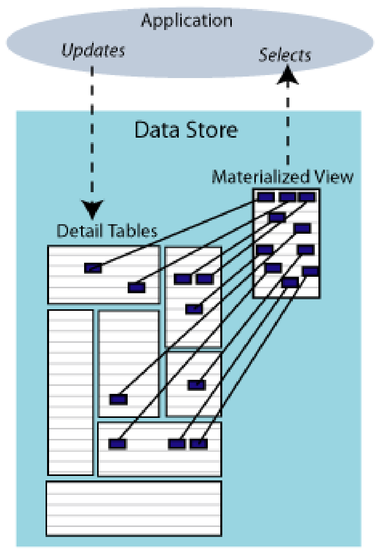
Overview of materialized views

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 8–1 shows a materialized view created from detail tables. An application updates the detail tables and can select data from the materialized view.

Figure 8–1 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 8-14.

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 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 acquire 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 are 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`.

Note: 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*.

If the owner has these privileges revoked for any of the detail tables on which the materialized view is created, the materialized view becomes invalid. See "Object privileges for materialized views" on page 4-15 for details.

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 8-5.

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`, 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 is refreshed at the specified interval. *`IntegerLiteral`* must be an integer. *`IntervalUnit`* must be one of the following values: `'DAY'`, `'HOUR'`, `'MINUTE'`, or `'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 an 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 some restrictions, which are described in "CREATE MATERIALIZED VIEW" in the *Oracle TimesTen In-Memory Database SQL Reference*.

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$_detailTableId`, where `detailTableId` is the table id of the table on which it was created. The table id and table name are both recorded in `SYS.TABLES`. For example, if the materialized view log file name is `MVLOG$_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 one of two built-in procedures for computing statistics: `ttOptUpdateStats` and `ttOptEstimateStats`.

  Note: For more details on updating table statistics and when it is appropriate to update statistics, see "Compute exact or estimated statistics" on page 10-18.

- 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, are used.
- Which indexes are present on the detail table and on the materialized view.
- How many materialized view rows are 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 performs poorly.

All update, insert, or delete statements on detail tables have execution plans, as described in "The TimesTen Query Optimizer" on page 9-1. For example, an update of a row in `T1` initiates the 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 its 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. You can use the Index Advisor to recommend indexes for a particular SQL workload. For more details, see "Using the Index Advisor to recommend indexes" on page 8-24.

Indexes 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 9-1.

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. For more information on null values, see "Null values" in the *Oracle TimesTen In-Memory Database SQL Reference*.

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
- Using the Index Advisor to recommend indexes

Overview of index types

TimesTen provides three types of indexes to enable fast access to tables.

- **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 does not expedite finding the answer, but a separate index will.

  See the "CREATE INDEX" section of the *Oracle TimesTen In-Memory Database SQL Reference* for more information on how to create range indexes.

- **Hash Indexes**. Hash indexes are useful for equality searches. A hash index is created with either of the following:

  – You can create a hash index or a unique hash index on one or more columns of a table or materialized view with the `CREATE INDEX` statement.

  – You can create a unique hash index on the primary key of a table during table creation with the `CREATE TABLE... UNIQUE HASH ON` statement.

  See the "CREATE INDEX" and "CREATE TABLE" sections of the *Oracle TimesTen In-Memory Database SQL Reference* for details on creating hash indexes. For an example of how to create a hash index, see Example 8–4. For details on how to size a hash table, see "Size hash indexes appropriately" on page 10-17.

  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.

  Range indexes are optimized for in-memory data management and provide efficient sorting by column value.

  TimesTen may create temporary hash and range indexes automatically during query processing to speed up query execution.

- **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. For more information, see "ROWID data type" in the *Oracle TimesTen In-Memory Database SQL Reference*.

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.

Note: You cannot create an index (range, hash, or bitmap) on LOB columns.

Example 8–4 Create an index

The following creates an index `IxID` over column `CustID` of table `NameID`.

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

The following creates a unique hash index on the `customer` table as part of the table creation:

```
CREATE TABLE customer
(cust_id NUMBER NOT NULL PRIMARY KEY,
cust_name CHAR(100) NOT NULL,
addr CHAR(100),
zip NUMBER,
region CHAR(10))
UNIQUE HASH ON (cust_id) PAGES = 30;
```

The following creates a nonunique hash index on the `customer` table over the customer name:

```
Command> CREATE HASH INDEX custname_idx ON customer(cust_name);
```

For more examples on the different indexes and how to create them, see "CREATE INDEX" and "ALTER TABLE" in the *Oracle TimesTen In-Memory Database SQL Reference*.

Altering an index

You can use the `ALTER TABLE` statement to add (or change) a primary key constraint to use either a range or hash index.

Note: You cannot alter an index to be transformed from a hash to a range index or from a range to a hash index if it was created with the `CREATE INDEX` statement.

You can change a primary key constraint to use a range index instead of a hash index with the `USE RANGE INDEX` clause of the `ALTER TABLE` statement; you can change a primary key constraint to use a hash index instead of a range index with the `USE HASH INDEX` of the `ALTER TABLE` statement. See "ALTER TABLE" in the *Oracle TimesTen In-Memory Database SQL Reference*.

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 8–5 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 database can be done on first connect. To avoid having to increase the size of a database, it is important not to underestimate the eventual database size. You can use the `ttSize` utility to estimate database size, including any indexes.

The following example shows that the `ttSize` utility estimates the rows, inline row bytes, size of any indexes on the table, and the total size of the table:

```
ttSize -tbl Pat.tab1 MyDb

Rows = 2

Total in-line row bytes = 17524
Indexes:

 Bitmap index PAT.BITMAP_ID adds 6282 bytes
  Total index bytes = 6282

Total = 23806
```

Using the Index Advisor to recommend indexes

The right set of indexes can make a difference in query performance. The Index Advisor can be used to recommend indexes for improving the performance of a specific SQL workload. The Index Advisor is intended for read-intensive complex queries. The use of the Index Advisor is not recommended for a write-intensive workload.

The Index Advisor evaluates a SQL workload and recommends indexes that can improve the performance for the following: joins, single table scans, and `ORDER BY` or

`GROUP BY` operations. The Index Advisor does not differentiate tables that are used for specific intentions, such as the base table for a materialized view or as a table within a cache group. As long as the table is used in queries in the SQL workload, the Index Advisor may recommend indexes on that table.

The Index Advisor generates the `CREATE` statement for each recommended index, which you can choose to execute. A database administrator should review each `CREATE` statement recommended for new indexes before they are applied since the Index Advisor may recommend the following:

- Indexes that are duplicates of existing indexes.
- Indexes for tables or columns of tables that are created and dropped during a SQL workload. However, you could add the `CREATE` statement for the recommended index in the SQL workload after the DDL that creates the tables or columns of tables and before they are dropped.
- Indexes that cannot be created, such as a unique index for a data set where the data is not unique. In this case, you should ignore this recommendation.
- Index creation options where you can create an index as either a `UNIQUE` or non-unique index. The Index Advisor suggests both index types. You can only create one of the indexes as both suggested indexes have the same index name. While the optimizer thinks that the `UNIQUE` index is better for the specified workload, you can choose to create the non-unique index. Consider creating the `UNIQUE` index if the column only contains unique values. Consider creating the non-unique index if the column contains non-unique value.

The Index Advisor does not cover the following:

- It does not optimize for memory use.
- It does not consider maintenance costs.
- It does not recommend that existing indexes be dropped if they are not useful.
- It does not recommend indexes for global temporary tables.

The recommended steps to use the Index Advisor are as follows:

- Prepare for executing the Index Advisor
- Capture the data used for generating index recommendations
- Retrieve index recommendations and data collection information
- Drop data collected for the index advisor and finalize results

Prepare for executing the Index Advisor

Before you execute the Index Advisor, you should optionally perform the following:

1. Since the Index Advisor relies on the query plan, set any relevant optimizer hints that you would use for the SQL workload before enabling the Index Advisor and running the workload. For more details on optimizer hints, see "Use optimizer hints to modify the execution plan" on page 9-14.
2. Update statistics for tables included in the SQL workload and force statements to be re-prepared during the capture. This provides the most up-to-date statistics for the data collection and causes the statements to be re-prepared based on the latest statistics.

Update statistics for tables included in the SQL workload with one of the following built-in procedures: `ttOptUpdateStats`, `ttOptEstimateStats`, or `ttOptSetTblStats`. In the built-in procedures, set the `invalidate` parameter to 1 to invalidate all commands that reference the indicated tables and force these commands to be automatically prepared again when re-executed. This ensures that statistics are up to date.

- The `ttOptUpdateStats` built-in procedure provides a full update of all statistics for the tables. However, it can be time consuming.
- The `ttOptEstimateStats` evaluates statistics based upon a small percentage of rows in the indicated tables.
- The `ttOptSetTblStats` sets the statistics to known values provided by you.

> **Note:** For more information on these built-in procedures, see "ttOptUpdateStats," "ttOptEstimateStats," and "ttOptSetTblStats" in the *Oracle TimesTen In-Memory Database Reference*.

The following example estimates statistics for all tables for the current user by evaluating a random sample of ten percent of the rows in these tables. It also invalidates all commands already prepared that reference these tables.

```
CALL ttOptEstimateStats ( '', 1, '10 PERCENT' );
```

Capture the data used for generating index recommendations

Call the `ttIndexAdviceCaptureStart` and `ttIndexAdviceCaptureEnd` built-in procedures to capture the information needed by the Index Advisor to generate index recommendations, as follows:

1. Call the `ttIndexAdviceCaptureStart` built-in procedure to start the process to collect index information.
2. Run the SQL workload.
3. Call the `ttIndexAdviceCaptureEnd` built-in procedure to end the index information collection process.

> **Note:** After the data collection process ends, you can retrieve the index recommendations as described in "Retrieve index recommendations and data collection information" on page 8-27.

When you call the `ttIndexAdviceCaptureStart` built-in procedure to initiate the data collection process, provide the following:

- In the `captureLevel` parameter, specify whether the index information is to be collected for the current connection or for the entire database. You can execute multiple connection-level captures concurrently for independent connections without conflict. A database-level capture can take place in parallel with a connection-level capture. Since there is no conflict between a database-level and a connection-level capture, any outstanding connection-level captures that are already in progress when a database-level capture is initiated completes as intended. However, an error is returned if you initiate a second request for a database-level capture while the first is still active; an error is also returned if a second request for a connection-level capture from the same connection is initiated while the first connection-level capture is still active.

  If you invoke `ttIndexAdviceCaptureStart` for a database-level capture, any outstanding connection-level captures that are already in progress complete.

- The `captureMode` parameter designates that you want the data collection performed on one of the following scenarios:
  - Perform the collection of index information using the current execution of the SQL workload.
  - Base the collection of index information not on a current execution of the SQL workload, but on existing computed statistics and query plan analysis. In this scenario, the SQL statements have been prepared, but not executed. This mode can only be executed with a connection-level capture.

To complete the capture, call the `ttIndexAdviceCaptureEnd` built-in procedure that ends either an active connection-level capture from the same connection or an active database-level capture. Completing a database-level capture requires the `ADMIN` privilege.

If a connection fails during a capture, the following occurs:

- If the capture is a connection-level capture, the capture ends and all associated resources are freed.
- If the capture is a database-level capture, the capture continues until another user with `ADMIN` privileges connects and invokes the `ttIndexAdviceCaptureEnd` built-in procedure to end a database-level capture.

If temporary space becomes full during a capture, an active capture ends and the data collected during the capture is saved.

Note: Execute `ttIndexAdviceCaptureDrop` to free the temporary space after a capture. See "Drop data collected for the index advisor and finalize results" on page 8-29 for more information on `ttIndexAdviceCaptureDrop`.

The following example starts a collection for the Index Advisor at the connection-level for the current execution of a SQL workload:

```
Call ttIndexAdviceCaptureStart(0,0);
```

The following example ends the collection for the connection-level capture:

```
Call ttIndexAdviceCaptureEnd(0)
```

Note: For more information on these built-in procedures, see "ttIndexAdviceCaptureStart" and "ttIndexAdviceCaptureEnd" in the *Oracle TimesTen In-Memory Database Reference*.

Retrieve index recommendations and data collection information

1. Call the `ttIndexAdviceCaptureInfoGet` built-in procedure to retrieve data collection overview information for the Index Advisor.
2. Call the `ttIndexAdviceCaptureOutput` built-in procedure to retrieve the recommended indexes.

Note: These built-in procedures retrieve the data collection overview and Index Advisor recommendations. Execute either or both for the data you want.

3. After a DBA has evaluated the recommended index creation statements, apply the desired index creation recommendations.

Retrieve data collection information with ttIndexAdviceCaptureInfoGet The `ttIndexAdviceCaptureInfoGet` built-in procedure retrieves information about the data collected for the Index Advisor. For both a connection-level capture and a database-level capture, only a single row is returned.

Note: The database-level capture row can only be returned to a user with `ADMIN` privileges.

The `ttIndexAdviceCaptureInfoGet` built-in procedure captures data if:

- The data capture was started and has not ended.
- A previous capture that was started and stopped, and the data was not deleted.

Note: If no capture is in progress or no data exists, then no rows are returned.

The rows returned include the following information:

- The capture state: Returns 0 if a capture is completed. Returns 1 if a capture is still in progress.
- The connection identifier, if appropriate.
- The capture level and mode set for this capture.
- The number of prepared and executed statements during the capture interval.
- The time that the capture was started and stopped.

The following shows capture information for a completed connection-level capture for 363 prepared statements and 369 executed statements:

```
Command> CALL ttIndexAdviceCaptureInfoGet();
< 0, 1, 0, 0, 363, 369, 2012-07-27 11:44:08.136833, 2012-07-27 12:07:35.410993 >
1 row found.
```

Note: For more details and syntax for this built-in procedure, see "ttIndexAdviceCaptureInfoGet" in the *Oracle TimesTen In-Memory Database Reference*.

Retrieve index recommendations with ttIndexAdviceCaptureOutput The `ttIndexAdviceCaptureOutput` built-in procedure retrieves the list of index recommendations from the last recorded capture at the specified level (connection or database-level). The list contains the `CREATE` statement for each recommended index.

To request index recommendations for a connection-level capture, execute `ttIndexAdviceCaptureOutput` with `captureLevel` set to 0 in the same connection that initiated the capture. For a database-level capture, execute `ttIndexAdviceCaptureOutput` with `captureLevel` set to 1 in a connection where the user has `ADMIN` privilege.

The returned row contains:

- `stmtCount` - The number of times the index would be useful to speed up the executed SQL workload.
- `createStmt` - The executable statement that can be used to create the recommended index. All database object names in these statements are fully qualified.

The following example provides the `CREATE` statement for an index called `PURCHASE_i1` on the `HR.PURCHASE` table, which would be useful 4 times for this SQL workload.

```
CALL ttIndexAdviceCaptureOutput();
< 4, create index PURCHASE_i1 on HR.PURCHASE(AMOUNT); >
1 row found.
```

Note: For more information and syntax for this built-in procedure, see "ttIndexAdviceCaptureOutput" in the *Oracle TimesTen In-Memory Database Reference*.

Drop data collected for the index advisor and finalize results

After you have applied the `CREATE` statements for the new indexes that have been approved by the DBA, you can drop the captured data collected for the Index Advisor. The `ttIndexAdviceCaptureDrop` built-in procedure drops the existing data collected for the specified `captureLevel`, which can either be a connection or database-level capture.

```
Call ttIndexAdviceCaptureDrop(0);
```

You must call this built-in procedure twice to drop both a connection-level and database-level capture. You may not invoke this built-in procedure while a capture at the same level is in progress.

Note: For more information and syntax for this built-in procedure, see "ttIndexAdviceCaptureDrop" in the *Oracle TimesTen In-Memory Database Reference*.

You can repeat the steps in "Prepare for executing the Index Advisor" on page 8-25 and "Retrieve index recommendations and data collection information" on page 8-27 until a SQL workload is executed with no more index recommendations. You can keep updating the statistics for the tables on which the new indexes were applied and re-execute the Index Advisor to see if any new indexes are now recommended.

Example using Index Advisor built-in procedures

The following shows the flow of a data collection for a SQL workload and the resulting index advice provided by the Index Advisor built-in procedures.

```
Command> CALL ttOptUpdateStats();

Command> CALL ttIndexAdviceCaptureStart();

Command> SELECT employee_id, first_name, last_name FROM employees;
< 100, Steven, King >
< 101, Neena, Kochhar >
< 102, Lex, De Haan >
< 103, Alexander, Hunold >
< 104, Bruce, Ernst >
...
```

```
< 204, Hermann, Baer >
< 205, Shelley, Higgins >
< 206, William, Gietz >
107 rows found.

Command> SELECT MAX(salary) AS MAX_SALARY
FROM employees
WHERE employees.hire_date > '2000-01-01 00:00:00';
< 10500 >
1 row found.

Command> SELECT employee_id, job_id FROM job_history
       > WHERE (employee_id, job_id) NOT IN (SELECT employee_id, job_id
       > FROM employees);
< 101, AC_ACCOUNT >
< 101, AC_MGR >
< 102, IT_PROG >
< 114, ST_CLERK >
< 122, ST_CLERK >
< 176, SA_MAN >
< 200, AC_ACCOUNT >
< 201, MK_REP >
8 rows found.

Command> WITH dept_costs AS (
       > SELECT department_name, SUM(salary) dept_total
       > FROM employees e, departments d
       > WHERE e.department_id = d.department_id
       > GROUP BY department_name),
       > avg_cost AS (
       > SELECT SUM(dept_total)/COUNT(*) avg
       > FROM dept_costs)
       > SELECT * FROM dept_costs
       > WHERE dept_total >
       > (SELECT avg FROM avg_cost)
       > ORDER BY department_name;
< Sales, 304500 >
< Shipping, 156400 >
2 rows found.

Command> call ttIndexAdviceCaptureEnd();

Command> call ttIndexAdviceCaptureInfoGet();
< 0, 1, 0, 0, 9, 6, 2012-07-27 11:44:08.136833, 2012-07-27 12:07:35.410993 >
1 row found.

Command> call ttIndexAdviceCaptureOutput();
< 1, create index EMPLOYEES_i1 on HR.EMPLOYEES(SALARY); >
< 1, create index EMPLOYEES_i2 on HR.EMPLOYEES(HIRE_DATE); >
2 rows found.

Command> call ttIndexAdviceCaptureDrop();
```

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:mi:ss [AM|am|PM|pm]`.
- Date represented as `yyyy-mm-dd`.
- Timestamp represented as `yyyy-mm-dd hh:mi:ss`.

The "Data Types" section 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 8–6 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 8–7 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%';
```

Understanding synonyms

A synonym is an alias for a database object. Synonyms are often used for security and convenience, because they can be used to mask object name and object owner. In addition, you can use a synonym to simplify SQL statements. Synonyms provide independence in that they permit applications to function without modification regardless of which object a synonym refers to. Synonyms can be used in DML statements and some DDL and TimesTen cache statements.

Synonyms are categorized into two classes:

- Private synonyms: A private synonym is owned by a specific user and exists in the schema of a specific user. A private synonym shares the same namespace as other object names, such as table names, view names, sequence names, and so on. Therefore, a private synonym cannot have the same name as a table name or a view name in the same schema.
- Public synonyms: A public synonym is owned by all users and every user in the database can access it. A public synonym is accessible for all users and it does not belong to any user schema. Therefore, a public synonym can have the same name as a private synonym name or a table name.

In order to create and use synonyms, the user must have the correct privileges, which are described in "Object privileges for synonyms" on page 4-17.

After synonyms are created, they can be viewed using the following views:

- `SYS.ALL_SYNONYMS`: describes the synonyms accessible to the current user. For more information, see "SYS.ALL_SYNONYMS" in the *Oracle TimesTen In-Memory Database System Tables and Views Reference*.
- `SYS.DBA_SYNONYMS`: describes all synonyms in the database. For more information, see "SYS.DBA_SYNONYMS" in the *Oracle TimesTen In-Memory Database System Tables and Views Reference*.
- `SYS.USER_SYNONYMS`: describes the synonyms owned by the current user. For more information, see "SYS.USER_SYNONYMS" in the *Oracle TimesTen In-Memory Database System Tables and Views Reference*.

Creating synonyms

Create the synonym with the `CREATE SYNONYM` statement. You can use the `CREATE OR REPLACE SYNONYM` statement to change the definition of an existing synonym without needing to drop it first. The `CREATE SYNONYM` and `CREATE OR REPLACE SYNONYM` statements specify the synonym name and the schema name in which the synonym is created. If the schema is omitted, the synonym is created in the user's schema. However, when creating public synonyms, do not provide the schema name as it is defined in the `PUBLIC` namespace.

In order to execute the `CREATE SYNONYM` or `CREATE OR REPLACE SYNONYM` statements, the user must have the appropriate privileges, as described in "Object privileges for synonyms" on page 4-17.

- *Object types for synonyms:* The `CREATE SYNONYM` and `CREATE OR REPLACE SYNONYM` statements define an alias for a particular object, which can be one of the following object types: table, view, synonym, sequence, PL/SQL stored procedure, PL/SQL function, PL/SQL package, materialized view, or cache group.

  Note: If you try to create a synonym for unsupported object types, you may not be able to use the synonym.

- *Naming considerations:* A private synonym shares the same namespace as all other object names, such as table names and so on. Therefore, a private synonym cannot have the same name as a table name or other objects in the same schema.

  A public synonym is accessible for all users and does not belong to any particular user schema. Therefore, a public synonym can have the same name as a private synonym name or other object name. However, you cannot create a public synonym that has the same name as any objects in the SYS schema.

In the following example, the user creates a private synonym of `synjobs` for the `jobs` table. Execute a `SELECT` statement on both the `jobs` table and the `synjobs` synonym to show that selecting from `synjobs` is the same as selecting from the `jobs` table. Finally, to display the private synonym, the example executes a `SELECT` statement on the `SYS.USER_SYNONYMS` table.

```
Command> CREATE SYNONYM synjobs FOR jobs;
Synonym created.

Command> SELECT FIRST 2 * FROM jobs;
< AC_ACCOUNT, Public Accountant, 4200, 9000 >
< AC_MGR, Accounting Manager, 8200, 16000 >
2 rows found.
Command> SELECT FIRST 2 * FROM synjobs;
< AC_ACCOUNT, Public Accountant, 4200, 9000 >
< AC_MGR, Accounting Manager, 8200, 16000 >
2 rows found.

Command> SELECT * FROM sys.user_synonyms;
< SYNJOBS, TTUSER, JOBS, <NULL> >
1 row found.
```

For full details, more examples, and rules on creating or replacing a synonym, see the "CREATE SYNONYM" section in the *Oracle TimesTen In-Memory Database SQL Reference*.

Dropping synonyms

Use the `DROP SYNONYM` statement to drop an existing synonym from the database. A user cannot be dropped unless all objects, including synonyms, owned by this user are dropped.

For example, the following drops the public synonym `pubemp`:

```
DROP PUBLIC SYNONYM pubemp;
Synonym dropped.
```

In order drop a public synonym or a private synonym in another user's schema, the user must have the appropriate privileges. For full details, more examples, and rules on creating or replacing a synonym, see the "DROP SYNONYM" section in the *Oracle TimesTen In-Memory Database SQL Reference*.

Synonyms may cause invalidation or recompilation of SQL queries

When a synonym or object is newly created or dropped, some SQL queries and DDL statements may be invalidated or recompiled. The following lists the invalidation and recompilation behavior for SQL queries and DDL statements:

1. All SQL queries that depend on a public synonym are invalidated if you create a private synonym with the same name for one of the following objects:
   - private synonym
   - table
   - view
   - sequence

- materialized view
- cache group
- PL/SQL object including procedures, functions, and packages

2. All SQL queries that depend on a private synonym or schema object are invalidated when a private synonym or schema object is dropped.

9 The TimesTen Query Optimizer

The TimesTen cost-based query optimizer uses information about an application's tables and their available indexes to choose a fast path to the data. Application developers can examine the plan chosen by the optimizer to check that indexes are used appropriately. If necessary, application developers can provide hints to influence the optimizer's behavior so that it considers a different plan.

This chapter includes the following topics:

- When optimization occurs
- Viewing SQL statements stored in the SQL command cache
- Viewing SQL query plans
- Modifying plan generation

When optimization occurs

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

A statement is automatically *invalidated* under the following circumstances:

- An object that the command uses is dropped
- An object that the command uses is altered
- An index on a table or view that the command references is dropped
- An index is created on a table or view that the command references

You can manually invalidate statements with either of the following methods:

- Use the `ttOptCmdCacheInvalidate` built-in procedure to invalidate statements in the SQL command cache. For more information, see "Control the invalidation of commands in the SQL command cache" on page 10-20.
- Set the `invalidation` option to 1 in the `ttOptUpdateStats` or the `ttOptEstimateStats` built-in procedures. These built-in procedures also update statistics for either a specified table or all of the current user's tables.

Note: For complete details on when to calculate statistics, see "Compute exact or estimated statistics" on page 10-18. In addition, see "ttOptUpdateStats," or "ttOptEstimateStats" in the *Oracle TimesTen In-Memory Database Reference*.

An invalid statement 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 statement may be prepared several times.

Note: When using JDBC, you must manually reprepare statement when a table has been altered.

A statement may have to be prepared manually if, for example, the table that the statement 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 statement 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.
`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 is 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 optimizer hints to influence the optimizer's behavior, these hints persist until the command is deleted. See "Modifying plan generation" on page 9-12 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 use those same hints.

Viewing SQL statements 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 9-9.

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 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 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.
- Detect 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 SQL 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.
- Number of fetch executions performed internally for this statement.
- The timestamp when the statement started.
- The maximum execution time in seconds for the statement.
- Last measured execution time in seconds for the statement.
- The minimum execution time in seconds for the statement.

At the end of the list of all commands, a status is printed of how many commands were in the cache.

The following examples show how to display all or a single command from the SQL command cache using the `ttSQLCmdCacheInfo` built-in utility:

- Displaying all commands in the SQL command cache
- Displaying a single SQL command

Example 9–1 Displaying all commands in the SQL command cache

This example executes within `ttIsql` the `ttSQLCmdCacheInfo` built-in procedure without arguments to show all cached commands. The commands are displayed in terse format. To display the information where each column is prepended with the column name, execute `vertical on` before calling the `ttSQLCmdCacheInfo` procedure.

```
Command> call ttSQLCmdCacheInfo;

< 110168040, 2048, 2, 2, 0, 1, 2792, CACHEUSER                      , create
writethrough cache group update_orders from ordertab
(orderid number(10) not null primary key,
custid number (6) not null), 0, 2013-01-04 13:13:47.614000, 4.908, 4.908, 0 >
< 35681280, 2048, 0, 8, 0, 1, 5256, SYS                            , select null
from sys.obj$ where obj#=:1 and type#=:2 and obj# not in (select p_obj# from
dependency$ where p_obj# = sys.obj$.obj#), 0, <NULL>, 0, 0, 0 >
< 110189568, 2048, 1, 1, 0, 1, 4304, PAT                            , call
ttstatsconfig('connsamplefactor',0), 0, 2013-01-08 08:33:18.969000, 0, 0, 0 >
< 110176976, 2048, 1, 1, 0, 1, 4328, PAT                            , call
ttstatsconfig('sqlcmdsamplefactor',0), 2, 2013-01-08 08:32:31.751000, .000664,
.000664, .000664 >
< 110193800, 2048, 1, 1, 0, 1, 4328, PAT                            , call
ttstatsconfig('connsamplefactor','1,0'), 2, 2013-01-08 08:33:29.578000, 0, 0, 0 >
< 35291992, 2048, 0, 1, 0, 1, 2568, SYS                             , select sys25
from sys.tables where cachegroup = ?, 0, <NULL>, 0, 0, 0 >
< 40815768, 2048, 2, 1, 0, 0, 3944, SYS                             , select
u.user#, u.password, u.identification, u.astatus from sys.user$ u where u.name =
:name and u.type# = 1, 2, 2013-01-04 13:03:29.262000, 0, 0, 0 >
< 110243304, 2048, 0, 2, 0, 1, 1208, PAT                            , call
ttcmdcacheinfo(), 0, <NULL>, 0, 0, 0 >
< 35287688, 2048, 1, 1, 0, 1, 4232, SYS                             , select
cgname, cgowner, cgid, refresh_mode, refresh_state, refresh_interval, sys10,
TBLCNT, CGATTRIBUTES from sys.cache_group, 1, 2013-01-04 13:04:33.549000, 0, 0, 0
>
< 110255264, 2048, 2, 2, 0, 1, 3048, SYS                             , select
order#,columns,types from sys.access$ where d_obj#=:1, 2, 2013-01-08
08:40:44.633000, 0, 0, 0 >
< 110185240, 2048, 1, 1, 0, 1, 4312, PAT                            , call
ttstatsconfig('connsamplefactor','1,5'), 2, 2013-01-08 08:33:09.505000, 0, 0, 0 >
< 35031664, 2048, 1, 1, 0, 1, 2232, CACHEUSER                       , call
ttCacheUidPwdSet('cacheuser','oracle'), 0, 2013-01-04 13:03:43.615000, .326, .326,
0 >
< 110229864, 2048, 2, 2, 0, 1, 4240, PAT                            , call
ttstatsconfig('sqlcmdsamplefactor'), 4, 2013-01-08 08:32:10.638000, .000082,
.000082, .000082 >
< 40819648, 2048, 1, 1, 0, 0, 3904, SYS                             , select 1
from sys.sysauth$ s where (s.grantee# = :userid or s.grantee# = 1) and
(s.privilege# = :priv or s.privilege# = 67), 1, 2013-01-04 13:03:29.262000, 0, 0,
0 >
< 35038832, 2048, 1, 2, 0, 1, 3336, CACHEUSER                       , create
writethrough cache group update_cust from active_customer
(custid number (6) not null primary key,
name varchar2(50),
addr varchar2(100),
zip varchar2(12)), 0, 2013-01-04 13:06:53.452000, 19.917, 19.917, 0 >
< 110270264, 2048, 4, 4, 0, 1, 3888, SYS                             , select
piece#,length,piece from sys.idl_ub2$ where obj#=:1 and part=:2 and version=:3
order by piece#, 34, 2013-01-08 08:40:44.633000, 0, 0, 0 >
< 110266280, 2048, 4, 4, 0, 1, 3888, SYS                             , select
piece#,length,piece from sys.idl_char$ where obj#=:1 and part=:2 and version=:3
order by piece#, 14, 2013-01-08 08:40:44.633000, 0, 0, 0 >
< 35034128, 2048, 1, 1, 0, 1, 1688, CACHEUSER                       , call
ttGridCreate('ttGrid'), 0, 2013-01-04 13:04:00.891000, 0, 0, 0 >
```

Example 9–2 Displaying a single SQL command

If you provide a command id as the input for the `ttSQLCmdCacheInfo`, the single command is displayed from within the SQL command cache. If no command id is provided to the `ttSQLCmdCacheInfo` built-in procedure, then it displays information about all current commands, where the command id is the first column of the output.

The following example displays the command identified by Command ID of `527973892`. It is displayed in terse format; to view with the column headings prepended, execute `vertical on` before calling 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 command in one of two ways: storing the latest query plan into the system `PLAN` table or viewing all cached 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 statement to be executed, 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 or 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 `ttOptUpdateStats` built-in procedure.
- The user invalidates commands with the `ttOptCmdCacheInvalidate` built-in procedure.

Note: For more information, see "Control the invalidation of commands in the SQL command cache" on page 10-20. For more information on the built-in procedures, see "ttOptUpdateStats" and "ttOptCmdCacheInvalidate" in the *Oracle TimesTen In-Memory Database Reference*.

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, enable the creation of entries in the `PLAN` table with the plan generation option as follows:

- For transaction level optimizer hints, call the built-in `ttOptSetFlag` procedure and enable the `GenPlan` flag.
- For statement level optimizer hints, set `TT_GENPLAN(1)`, which is only in effect for the statement. After the statement executes, the plan generation option takes on the value of the `GenPlan` transaction level optimizer hint.

Note: See "Use optimizer hints to modify the execution plan" on page 9-14 for details on statement level and transaction level optimizer hints.

This 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.

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 plan generation option set, it is also recompiled for plan generation. Thus, the plan is generated even though the plan for another query was found in the `SYS.PLAN` table.

You can use `showplan` in `ttIsql` to test the query and optimizer hints, which enables plan generation as well as shows the query plan for the statements in the transaction. Autocommit must be off.

```
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 Tables" in the *Oracle TimesTen In-Memory Database System Tables and Views Reference*.

Example 9–3 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	TblLkRangeScan	T1	IX1		
2	3	TblLkRangeScan	T2	IX2(D)		T2.B <> 0
3	2	MergeJoin			T1.A = -T2.A	
4	2	TblLkRangeScan	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 9-8.

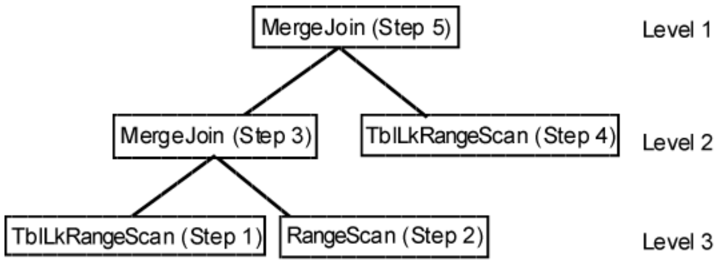
Describing the PLAN table columns

The `SYS.PLAN` table has seven columns.

Column 1 (Step) Indicates the order of operation, which always starts with one. Example 9–3 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 9–3, 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 Tables" in the *Oracle TimesTen In-Memory Database System Tables and Views 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 9–3, 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 9–3, 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 9-3.

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 command output, only the query plan for the specified command is displayed.

Note: If you want to display a query plan for a specific command, you must provide the command identifier that is displayed with the `ttSQLCmdCacheInfo` built-in procedure. See "Displaying commands stored in the SQL command cache" on page 9-3 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 9-7. 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 `ttSQLCmdQueryPlan` built-in process displays the query plan in a raw data format. Alternatively, you can execute the `ttIsql explain` command for a formatted version of this output. For more information, see "Display query plan for statement in SQL command cache" on page 6-35.

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 9–4 Displaying all SQL query plans

You can display all SQL query plans associated with commands stored in the command cache with the `ttSQLCmdQueryPlan` 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 calling 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, RowLkRangeScan                 , T2
, PAT                        , I2                             , , NOT(LIKE( tuf
,abc ,NULL ))  >
< 528079360, <NULL>, 2, 3, RowLkRangeScan                 , 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, RowLkRangeScan                 , T2
, PAT                         , I2                            , ,  >
< 528123000, <NULL>, 2, 6, RowLkRangeScan                 , 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 9–5   Displaying a single SQL query plan

You can display any query plan associated with a command by providing the command id of the command as the input for the `ttSQLCmdQueryPlan` built-in procedure. The single query plan is displayed from within the SQL command cache. If no command id is supplied, the `ttSQLCmdCacheInfo` built-in procedure displays information about all current commands in the TimesTen cache.

The following example displays the query plan of the command identified by command id of `528078576`. It is displayed in terse format; to view with the column headings prepended, execute `vertical on` before calling 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, RowLkRangeScan                 , T2
, PAT                         , I2                            , ,  >
< 528078576, <NULL>, 2, 7, RowLkRangeScan                 , T5
, PAT                         , I2                            , ,  >
< 528078576, <NULL>, 3, 6, NestedLoop                     ,
,                                 ,                               , ,  >
< 528078576, <NULL>, 4, 6, RowLkRangeScan                 , 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, RowLkRangeScan                , 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 9-6. 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?
- How hints can influence an execution plan
- Use optimizer hints to modify the execution plan

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.

How hints can influence an execution plan

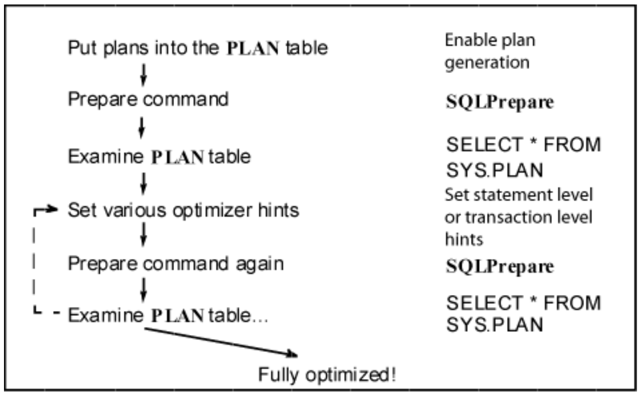
You can apply hints to pass instructions to the TimesTen query optimizer. The optimizer considers these hints when choosing the best execution plan for your query.

Transaction level hints are in effect for all calls to the ODBC `SQLPrepare` function or JDBC `PreparedStatement` objects in the transaction.

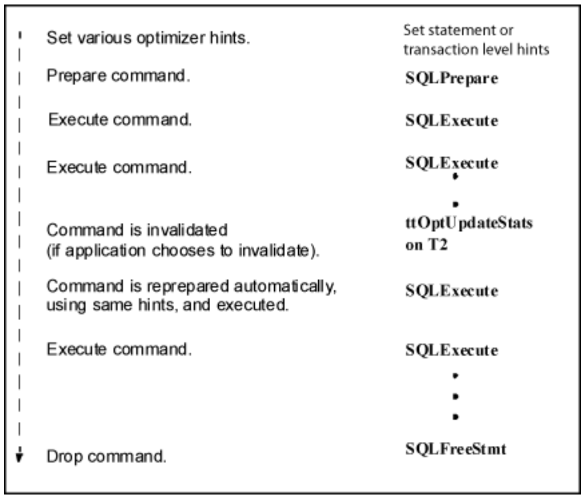
- If a command is prepared with certain hints in effect, those hints continue to apply if the command is reprepared automatically, even when this happens outside the initial prepare transaction. This can happen when a table is altered, or an index is dropped or created, or when statistics are modified, as described in "When optimization occurs" on page 9-1.
- If a command is prepared without hints, subsequent hints do not affect the command if it is reprepared automatically. An application must call the ODBC `SQLPrepare` function or JDBC `Connection.prepareStatement` method a second time so that hints have an effect.

Example 9–6 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.

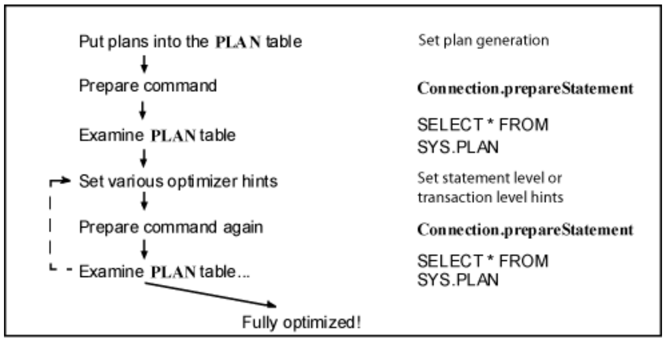


During execution, the application may then go through the steps shown in the following figure.

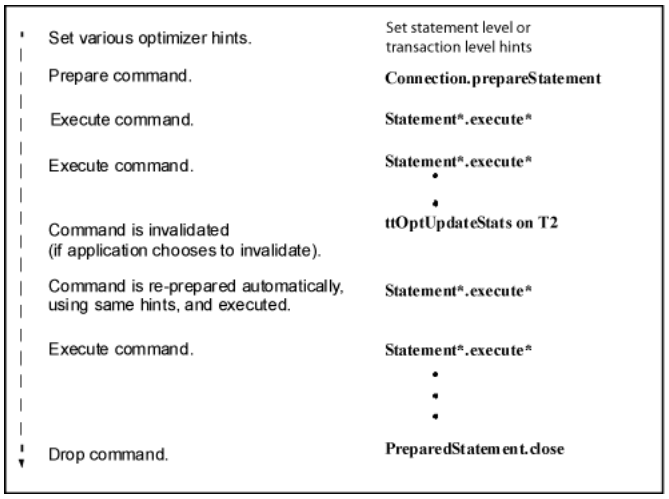

Set various optimizer hints.
Set statement or transaction level hints
Prepare command.
SQLPrepare
Execute command.
SQLExecute
Execute command.
SQLExecute
Command is invalidated (if application chooses to invalidate).
ttOptUpdateStats on T2
Command is reprepared automatically, using same hints, and executed.
SQLExecute
Execute command.
SQLExecute
Drop command.
SQLFreeStmt

Example 9–7 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.


Put plans into the PLAN table
Set plan generation
Prepare command
Connection.prepareStatement
Examine PLAN table
SELECT * FROM SYS.PLAN
Set various optimizer hints
Set statement level or transaction level hints
Prepare command again
Connection.prepareStatement
Examine PLAN table...
SELECT * FROM SYS.PLAN
Fully optimized!

During execution, the application may then go through the steps shown in the following figure.


Set various optimizer hints.
Set statement level or transaction level hints
Prepare command.
Connection.prepareStatement
Execute command.
Statement*.execute*
Execute command.
Statement*.execute*
Command is invalidated (if application chooses to invalidate).
ttOptUpdateStats on T2
Command is re-prepared automatically, using same hints, and executed.
Statement*.execute*
Execute command.
Statement*.execute*
Drop command.
PreparedStatement.close

Use optimizer hints to modify the execution plan

You can apply hints that to pass instructions to the TimesTen query optimizer as follows:

- To apply a hint only for a particular SQL statement, use a statement level optimizer hint.
- To apply a hint for an entire transaction, use a transaction level optimizer hint with the appropriate TimesTen built-in procedure.

Note: TimesTen concurrently processes read and write queries optimally. Your read queries can be optimized for read-only concurrency when you use transaction level optimizer hints such as `ttOptSetFlag ('tblLock',1)` or statement level optimizer hints such as `/*+ tt_tbllock(1) tt_rowlock(0) */`. Write queries that operate concurrently with read optimized queries may result in contention.

You can control read optimization during periods of concurrent write operations with the `ttDbWriteConcurrencyModeSet` built-in procedure. For more information, see "Control read optimization during concurrent write operations" on page 10-23.

Directions for applying hints are described in the following sections:

- Apply statement level optimizer hints for a SQL statement
- Apply transaction level optimizer hints for a transaction

Apply statement level optimizer hints for a SQL statement

A statement level optimizer hint is a specially formatted SQL comment containing instructions for the SQL optimizer. A statement level optimizer hint can be specified within the SQL statement that it is to be applied against with one of the following methods:

- /*+ */ The hints can be defined over multiple lines. The hints must be enclosed in the comment syntax. The plus sign (+) denotes the start of a hint.
- --+ The hint must be defined on a single line after the plus sign (+).

The statement level optimizer hint can be specified in the `SELECT`, `INSERT`, `UPDATE`, `MERGE`, `DELETE`, `CREATE TABLE AS SELECT`, or `INSERT ... SELECT` statements. You must specify the hint within comment syntax immediately following the `SQL VERB`. The placement, rules, and syntax on how to define statement level optimizer hints are fully described in "Statement level optimizer hints" in the *Oracle TimesTen In-Memory Database SQL Reference*.

Any hint defined in the SQL statement overrides any transaction level optimizer hint defined by a TimesTen built-in procedure, which are described in "Apply transaction level optimizer hints for a transaction" on page 9-15.

Note: If you specify any statement level optimizer hints incorrectly, TimesTen ignores these hints and does not provide an error. If you define conflicting hints, the rightmost hint overrides any conflicting hints for the statement.

Apply transaction level optimizer hints for a transaction

To change the query optimizer behavior for all statements in a transaction, an application calls one of the following built-in procedures using the ODBC procedure call interface:

Note: Make sure autocommit is off for transaction level optimizer hints. All optimizer flags are reset to their default values when the transaction has been committed or rolled back. If optimizer flags are set while autocommit is on, the optimizer flags are ignored because each statement is executed within its own transaction.

- `ttOptSetFlag`—Sets certain optimizer parameters. Provides the optimizer with transaction level optimizer hints with a recommendation on how to best optimize a particular query.
- `ttOptGetFlag`—View the existing transaction level hints set for a database.
- `ttOptSetOrder`—Enables an application to specify the table join order.
- `ttOptUseIndex`—Enables an application to specify that an index be used or to disable the use of certain indexes; that is, to specify which indexes should be considered for each correlation in a query.
- `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 transaction level optimizer hints. To view current transaction level optimizer hint settings, use the `optprofile` command.

- Example 9–8, "Using ttOptSetFlag in JDBC"
- Example 9–9, "Using ttOptSetFlag in ODBC"

Example 9–8 Using ttOptSetFlag in JDBC

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 9–9 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 */
...
```

10 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.

The following sections describe how to tune and identify performance issues:

Note: You can also identify performance issues by examining the `SYS.SYSTEMSTATS` table.

For information on tuning TimesTen Java applications, see "Java Application Tuning" in the *Oracle TimesTen In-Memory Database Java Developer's Guide*. For information on tuning TimesTen C applications, see "ODBC Application Tuning" in the *Oracle TimesTen In-Memory Database C Developer's Guide*.

System and database tuning

The following sections include tips for tuning your system and databases:

Provide enough memory

Performance impact: Large

Configure your system so that the entire database fits in main memory. The use of virtual memory substantially decreases performance. The database 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 tables in your database. For more information, see "ttSize" in the *Oracle TimesTen In-Memory Database Reference*.

Size your database correctly

Performance impact: Variable

When you create a database, you are required to specify a database size. Specifically, you specify sizes for the permanent and temporary memory regions of the database. For details on how to size the database and shared memory, see "Specifying the size of a database" on page 1-29.

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.

Note: When you enable PL/SQL (`PLSQL=1`), there is both a fixed and per connection overhead automatically allocated from the PL/SQL segment, even if you do not use PL/SQL.

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

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 MB on UNIX systems and 32 MB on Windows 32-bit systems. 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

Log buffer waits occur when application processes cannot insert transaction data to the log buffer and must stall to wait for log buffer space to be freed. The usual reason for this is that the log flusher thread has not cleared out data fast enough. This may indicate that log buffer space is insufficient, disk bandwidth is insufficient, writing to disk is taking too long, or the log flusher is CPU-bound.

Increasing the value of `LogBufMB` and `LogFileSize` can have a substantial positive performance impact. If `LOG_BUFFER_WAITS` is increasing, increase the value of `LogBufMB`. If log buffer waits still persist after increasing the `LogBufMb` and `LogFileSize` values, then review the other possible issues mentioned above.

The trade-off from increasing the value of `LogBufMb` 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.

Note: For more details, see "LogBufMB" and "LogFileSize" in the *Oracle TimesTen In-Memory Database Reference*.

Use temporary databases if appropriate

Performance impact: Variable

A TimesTen database may be permanent or temporary. A temporary database disappears when the last connection goes away or when there is a system or application failure. For information on temporary databases, see "Database overview" on page 8-1.

If you do not need to save the database to disk, you can save checkpoint overhead by creating a temporary database.

Temporary databases are never fully checkpointed to disk, although the transaction log is periodically written to disk. The amount of data written to the transaction log for temporary databases is less than that written for permanent databases, allowing better performance for temporary databases. Checkpoint operations can have significant overhead for permanent databases, depending on database size and activity, but have very little impact for temporary databases. Checkpoints are still necessary to remove transaction log files.

Avoid connection overhead

Performance impact: Large

By default, TimesTen loads an idle database, which is a database with no connections, into memory when a first connection is made to it. When the final application disconnects from a database, a delay occurs when the database is written to disk. If applications are continually connecting and disconnecting from a database, the database 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 database you may want to pre-load the database into memory before any applications attempt to use it.

To avoid the latency of loading a database into memory, you can change the RAM policy of the database to allow databases to always remain in memory. The trade-off is that since the database is never unloaded from memory, a final disconnect checkpoint never occurs. So, applications should checkpoint the database explicitly to reduce the disk space taken up by transaction log files.

Alternatively, you can specify that the database remain in memory for a specified interval of time and accept new connections. If no new connections occur in this interval, TimesTen unloads the database from memory and checkpoints it. You can also specify a setting to enable a system administrator to load and unload the database from memory manually.

To change the RAM policy of a database, use the `ttAdmin` utility. For more details on the RAM policy and the `ttAdmin` utility, see "Specifying a RAM policy" on page 1-25 in this book and the "ttAdmin" section in the *Oracle TimesTen In-Memory Database Reference*.

Load the database into RAM when duplicating

Performance impact: Large

When you duplicate a database, use the `-ramLoad` option of the `ttAdmin` utility. This places the database in memory, available for connections, instead of unloading it with a blocking checkpoint. For more information, see "Avoid connection overhead" on page 10-4 in this book and the "ttAdmin" section in the *Oracle TimesTen In-Memory Database Reference*.

Prevent reloading of the database after automatic recovery fails

Performance impact: Large

When TimesTen recovers after a database is invalidated, a new database is reloaded. However, the invalidated database is only unloaded after all connections to this database are closed. Thus, both the invalidated database and the recovered database could exist in RAM at the same time.

Reloading a large database into memory when an invalidated database still exists in memory can fill up available RAM. See "Preventing an automatic reload of the database after failure" on page 1-27 on how to stop automatic reloading of the database.

Reduce contention

Performance impact: Large

Database 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 10-9.
- Distribute data strategically in multiple tables or databases.

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.locks_granted.immediate`, `lock.locks_granted.wait` and `lock.timeouts` columns in the `SYS.SYSTEMSTATS` table provide some information on lock contention:

- `lock.locks_granted.immediate` counts how often a lock was available and was immediately granted at lock request time.
- `lock.locks_granted.wait` counts how often a lock request was granted after the requestor had to wait for the lock to become available.
- `lock.timeouts` 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 reduce the number of threads or processes making API (JDBC, ODBC, or OCI) calls. Using fewer threads 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 database.

Avoid operating system 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 databases 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 AIX, you can avoid paging by configuring large pages, which locks the shared segment into memory so it cannot be paged. See "Large pages (AIX)" in the *Oracle TimesTen In-Memory Database Installation Guide* for details on how to configure large pages on AIX.

On Linux, you can avoid paging by configuring large pages, which locks the shared segment into memory so it cannot be paged. See "Large pages (Linux)" in the *Oracle TimesTen In-Memory Database Installation Guide* for details on how to configure large pages on Linux.

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 database-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 database-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.

For more details, see "ttBulkCp" and "ttMigrate" in the *Oracle TimesTen In-Memory Database Reference*.

Check your driver

Performance impact: Large

There are two versions of the TimesTen Data Manager driver for each platform: a debug and production version. Unless you are debugging, use the production version. The debug library can be significantly slower. See "Specify the Data Manager DSN" on page 1-10 and "Specify the ODBC driver" on page 1-9 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 applications are linked with the correct TimesTen driver. For direct connect applications, use the TimesTen

Data Manager 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.

Use metrics to evaluate performance

Performance impact: Variable

You can use the `ttStats` utility to collect and display database metrics. The `ttStats` utility can perform the following functions.

- Monitor and display database performance metrics in real-time, calculating rates of change during each preceding interval.
- Collect and store snapshots of metrics to the database then produce reports with values and rates of change from specified pairs of snapshots. (These functions are performed through calls to the `TT_STATS` PL/SQL package.)

The `ttStats` utility gathers metrics from TimesTen system tables, views, and built-in procedures. In reports, this includes information such as a summary of memory usage, connections, and load profile, followed by metrics (as applicable) for SQL statements, transactions, PL/SQL memory, replication, logs and log holds, checkpoints, cache groups, cache grid, latches, locks, XLA, and TimesTen connection attributes.

For details on the `ttStats` utility, see "ttStats" in the *Oracle TimesTen In-Memory Database Reference*. For more details on the `TT_STATS` PL/SQL package, see "TT_STATS" in the *Oracle TimesTen In-Memory Database PL/SQL Packages Reference*.

Investigate alternative JVMs

Performance impact: Variable

JRockit and IBM provide JVMs that may perform better than the Sun JVM.

Migrate data with character set conversions

Performance impact: Variable

If character set conversion is requested when migrating databases, performance may be slower than if character set conversion is not requested.

Client/Server tuning

The following sections include tips for Client/Server tuning:

- Diagnose Client/Server performance
- 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 system
- Enable TT_PREFETCH_CLOSE for Serializable transactions
- Use a connection handle when calling SQLTransact

Diagnose Client/Server performance

Performance impact: Variable

You can analyze your Client/Server performance with the following statistics that are tracked in the `SYS.SYSTEMSTATS` table that can also be viewed with either the `ttStats` utility or the `TT_STATS` PL/SQL package:

Note: For more details on the `SYS.SYSTEMSTATS` table, see "SYS.SYSTEMSTATS" in the *Oracle TimesTen In-Memory Database System Tables and Views Reference*. For details on the `ttStats` utility, see "ttStats" in the *Oracle TimesTen In-Memory Database Reference*. For more details on the `TT_STATS` PL/SQL package, see "TT_STATS" in the *Oracle TimesTen In-Memory Database PL/SQL Packages Reference*.

- Total number of executions from a Client/Server application.
- Total number of `INSERT`, `UPDATE`, `DELETE`, `SELECT`, `MERGE`, `ALTER`, `CREATE`, `DROP` statements executed by the server.
- Total number of transactions committed or rolled back by the server.
- Total number of table rows inserted, updated, or deleted by the server.
- Total number of Client/Server roundtrips.
- Total number of bytes transmitted or received by the server.
- Total number of Client/Server disconnects.

Work locally when possible

Performance impact: Large

Using TimesTen Client to access databases on a remote server system 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

Performance impact: Variable

By default, connections wait 10 seconds to acquire a lock. To change the timeout interval for locks, use the `ttLockWait` built-in procedure.

For more details, see "ttLockWait" in the *Oracle TimesTen In-Memory Database Reference*.

Choose the best method of locking

Performance impact: Variable

When multiple connections access a database simultaneously, TimesTen uses locks to ensure that the various transactions operate in apparent isolation. TimesTen supports the isolation levels described in Chapter 7, "Transaction Management". It also supports the locking levels: database-level locking, table-level locking and row-level locking. You can use the `LockLevel` connection attribute to indicate whether database-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-level locking.

Note: For more information, see "LockLevel" and "ttOptSetFlag" in the *Oracle TimesTen In-Memory Database Reference*.

Choose an appropriate lock level

If there is very little contention on the database, use table-level locking. It provides better performance and deadlocks are less likely. There is generally little contention on the database 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.

Database-level locking completely restricts database access to a single transaction, and it is not recommended for ordinary operations. A long-running transaction using database-level locking blocks all other access to the database, affecting even the various background tasks needed to monitor and maintain the database.

Row-level locking is generally preferable when there are many concurrent transactions that are not likely to need access to the same row. On modern systems with a sufficient number of processors using high-concurrency, for example, multiple `ttBulkCp` processes, row-level locking generally outperforms database-level locking.

Note: For more information on the `ttBulkCp` utility, see "ttBulkCp" in the *Oracle TimesTen In-Memory Database Reference*.

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.

- 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.

- Any transaction updating a row blocks out writers to that row until the transaction commits. A reader of that row receives the previously committed version of the row.
- 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 `lock.deadlocks` columns of the `SYS.SYSTEMSTATS` table.

For more details on isolation levels, see "Transaction isolation levels" on page 7-6.

Use shared memory segment as IPC when client and server are on the same system

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 system, 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 options" on page 3-9.

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 2-17.

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 system 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 2-17. 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 example shows 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

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 9, "The TimesTen Query Optimizer" describes how to view the plan that TimesTen uses to execute a statement. Alternatively, you can use the `ttIsql showplan` command to view the plan. View the plan for each frequently executed statement in the application. If indexes are not used to evaluate predicates, consider creating new indexes or rewriting the statement or query so that indexes can be used. For example, indexes can only be used to evaluate `WHERE` clauses when single columns appear on one side of a comparison predicate (equalities and inequalities), or in a `BETWEEN` predicate.

If this comparison predicate is evaluated often, it would therefore make sense to rewrite

```
WHERE c1+10 < c2+20
```

to

```
WHERE c1 < c2+10
```

and create an index on `c1`.

The presence of indexes does slow down write operations such as `UPDATE`, `INSERT`, `DELETE` and `CREATE VIEW`. If an application does few reads but many writes to a table, an index on that table may hurt overall performance rather than help it.

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 8-9.

Collect and evaluate sampling of execution times for SQL statements

Performance impact: Variable

TimesTen provides built-in procedures that measure the execution time of SQL operations to determine the performance of SQL statements. Instead of tracing, the built-in procedures sample the execution time of SQL statements during execution. The built-in procedures measure the execution time of SQL statements by timing the execution within the `SQLExecute` API.

You can configure the sampling rate and how the execution times are collected with the `ttStatsConfig` built-in procedure and the following name-value pairs:

Note: For full details, see "ttStatsConfig" in the *Oracle TimesTen In-Memory Database Reference*.

Table 10–1 ttStatsConfig parameter and value descriptions

Parameter	Description
`SQLCmdSampleFactor`	Configures how often a SQL statement execution timing sample is taken. The default is 0, which means that the sampling is turned off. For example, when set to 10, TimesTen captures the wall clock time of the SQL statement execution for every 10th statement.
`ConnSampleFactor`	Configures how often a SQL statement sample is taken for an individual connection. The value includes two parameters separated by a comma within quotes, so that it appears as a single value. The first number is the connection ID; the second is the same as the `SQLCmdSampleFactor` as a number that designates how often the command sample is taken. By default, sampling is turned off (set to zero) for individual connections.
`SQLCmdHistogramReset`	When set to a nonzero value, clears the SQL execution time histogram data.
`StatsLevel`	Sets the level of statistics to be taken. Values can be set to either `NONE`, `BASIC`, `TYPICAL`, or `ALL`. The default is `TYPICAL`. Setting the level to `ALL` could negatively impact your performance.

The following are examples of how to set the name-value pairs with the `ttStatsConfig` built-in procedure:

Note: You can achieve the best results by choosing representative connections with the `ConnSampleFactor` parameter in the `ttStatsConfig` built-in procedure, rather than sampling all transactions. Sampling all transactions with a small sample factor can affect your performance negatively.

For meaningful results, the database should remain in memory since unloading and re-loading the database empties the SQL command cache.

Sample every 5th statement on connection 1.

```
Command> call ttStatsConfig('ConnSampleFactor', '1,5');
< CONNSAMPLEFACTOR, 1,5 >
1 row found.
```

Turn off sampling on connection 1.

```
Command> call ttStatsConfig('ConnSampleFactor', '1,0');
< CONNSAMPLEFACTOR, 1,0 >
1 row found.
```

Sample every command:

```
Command> call ttStatsConfig('SqlCmdSampleFactor',1);
< SQLCMDSAMPLEFACTOR, 1 >
1 row found.
```

Check whether sampling:

```
Command> call ttStatsConfig('SqlCmdSampleFactor');
< SQLCMDSAMPLEFACTOR, 1 >
1 row found.
```

Check the current database statistics collection level.

```
Command> call ttStatsConfig('StatsLevel');
< STATSLEVEL, TYPICAL >
1 row found.
```

Turn off database statistics collection by setting to `NONE`.

```
Command> call ttStatsConfig('StatsLevel','None');
< STATSLEVEL, NONE >
1 row found.
```

Once you have configured the statistics that you want collected, the collected statistics are displayed with the `ttSQLCmdCacheInfo` built-in procedure. To display the execution time histogram at either the command or database levels, use the `ttSQLExecutionTimeHistogram` built-in procedure.

The `ttSQLCmdCacheInfo` built-in procedure displays the following information relating to SQL execution time statistics:

- Number of fetch executions performed internally for this statement.
- The timestamp when the statement started.
- The maximum wall clock execute time in seconds of this statement.
- Last measured execution time in seconds of the statement.

- The minimum execute time in seconds of the statement.

In the following example, the display shows these statistics as the last five values:

```
Command> vertical call ttSQLCmdCacheInfo(135680792);

  SQLCMDID:                         146250096
  PRIVATE_COMMAND_CONNECTION_ID:    2048
  EXECUTIONS:                       40
  PREPARES:                         20
  REPREPARES:                       1
  FREEABLE:                         1
  SIZE:                             3880
  OWNER:                            ORATT
  QUERYTEXT:                        select min(unique2) from big1
  FETCHCOUNT:                       40
  STARTTIME:                        2012-06-18 13:10:46.808000
  MAXEXECUTETIME:                   .001319
  LASTEXECUTETIME:                  .000018
  MINEXECUTETIME:                   .000017
1 row found.
```

For more information on the `ttSQLCmdCacheInfo` built-in procedure, see "ttSQLCmdCacheInfo" in the *Oracle TimesTen In-Memory Database Reference*.

The `ttSQLExecutionTimeHistogram` built-in procedure displays a histogram of SQL execution times for either a single SQL command or all SQL commands in the command cache, assuming that sampling is enabled where `SQLCmdSampleFactor` is greater than zero.

The histogram displays a single row for each bucket of the histogram. Each row includes the following information:

- The number of SQL statement execution time operations that have been measured since either the TimesTen database was started or after the `ttStatsConfig` built-in procedure was used to reset statistics.
- Accumulated wall clock execution time.
- The execution time limit that denotes each time frame.
- The last row shows the number of SQL statements that executed in a particular time frame.

The following example shows the output for the `ttSQLExecutionTimeHistogram` built-in procedure:

The following example of the `ttSQLExecutionTimeHistogram` built-in procedure shows that a total of 1919 statements executed. The total time for all 1919 statements to execute was 1.090751 seconds. This example shows that SQL statements ran in the following time frames:

- 278 statements executed in a time frame that was less than or equal to .00001562 seconds.
- 1484 statements executed in a time frame that was greater than .00001562 seconds and less than or equal to .000125 seconds.
- 35 statements executed in a time frame that was greater than .000125 seconds and less than or equal to .001 seconds.
- 62 statements executed in a time frame that was greater than .001 seconds and less than or equal to .008 seconds.

- 60 statements executed in a time frame that was greater than .008 seconds and less than or equal to .064 seconds.

```
Command> call ttSQLExecutionTimeHistogram;
< 1919, 1.090751, .00001562, 278 >
< 1919, 1.090751, .000125, 1484 >
< 1919, 1.090751, .001, 35 >
< 1919, 1.090751, .008, 62 >
< 1919, 1.090751, .064, 60 >
< 1919, 1.090751, .512, 0 >
< 1919, 1.090751, 4.096, 0 >
< 1919, 1.090751, 32.768, 0 >
< 1919, 1.090751, 262.144, 0 >
< 1919, 1.090751, 9.999999999E+125, 0 >
10 rows found.
```

Select hash, range, or bitmap indexes appropriately

Performance impact: Variable

The TimesTen database supports hash, range, and bitmap indexes. The following details when it is appropriate to use each type of index.

Hash indexes are useful for finding rows with an exact match on one or more columns. Hash indexes are useful for doing equality searches. A hash index is created with either of the following:

- You can create a hash index or a unique hash index with the `CREATE [UNIQUE] HASH INDEX` statement.
- You can create a unique hash index when creating your table with the `CREATE TABLE... UNIQUE HASH ON` statement. The unique hash index is specified over the primary key columns of the table.

Range indexes are created by default with the `CREATE TABLE` statement or created with the `CREATE [UNIQUE] HASH 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 are optimized for in-memory data management.

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` is not used for the following queries, because the prefix property is not satisfied:

```
SELECT * FROM T WHERE i2=12;
```

There is no equality condition for `i1`.

The index `IXT` is used, but matching only occurs on the first two columns for queries like the following:

```
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 nonunique 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 as both primary key constraints and when specified as part of the `CREATE INDEX` statement. The size of the hash index is determined by the `PAGES` parameter specified in the `UNIQUE HASH ON` clause of the `CREATE TABLE` and `CREATE INDEX` statements. The value for `PAGES` should be the expected number of rows in the table divided by 256; for example, a 256,000 row table should have `PAGES = 1000`. A smaller value may result in a greater number of hash 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 rows in the table varies dramatically, and if performance is the primary consideration, it is best to create a large index. If the size of a table cannot be accurately predicted, consider using a range 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. See information about `SET PAGES` in the "ALTER TABLE" section in the *Oracle TimesTen In-Memory Database SQL Reference*.

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.

Compute exact or estimated statistics

Performance impact: Large

If statistics are available on the data in the database, 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.

Note: See Chapter 9, "The TimesTen Query Optimizer" for more information.

You should compute statistics before preparing your statements, since the information is likely to result in a more efficient query optimizer plan. When gathering statistics, you need to determine when and how often to gather new statistics as performance is affected by the statistics collection process. The frequency of collection should balance the task of providing accurate statistics for the optimizer against the processing overhead incurred by the statistics collection process.

Since computing statistics is a time-consuming operation, you should compute statistics with the following guidelines:

- Update statistics after loading your database or after major application upgrades.
- Do not update statistics during a heavy transaction load.
- Update statistics when there is substantial creation or alteration on tables, columns, or PL/SQL objects.

  If you have created or altered a substantial number of tables, columns, or PL/SQL objects in your database, you should update the data dictionary optimizer statistics for the following system tables: `SYS.TABLES`, `SYS.COLUMNS`, and `SYS.OBJ$`.
- When you substantially modify tables in batch operations, such as a bulk load or bulk delete, you can gather statistics on these tables as part of the batch operation.
- Update statistics infrequently, such as once a week or once a month, when tables are only incrementally modified.
- Update statistics as part of a regularly executed script or batch job during low transaction load times.
- When updating the statistics for multiple large tables, see "Update table statistics for large tables in parallel" on page 10-19.

Note: For performance reasons, TimesTen does not hold a lock on tables or rows when computing statistics.

Use the following for computing statistics: `ttIsql statsupdate` command, `ttOptUpdateStats`, or `ttOptEstimateStats`. Providing an empty string as the table name updates statistics for all tables in the current user's schema.

- The `statsupdate` command within `ttIsql` evaluates every row of the table(s) in question and computes exact statistics.
- 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. This can be faster, but may result in less accurate statistics. 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.

Note: For more details on `ttIsql` or the built-in procedures, see "ttIsql" and "Built-In Procedures" in the *Oracle TimesTen In-Memory Database Replication Guide*.

Update table statistics for large tables in parallel

Performance impact: Large

It is important to keep table statistics up to date for all TimesTen tables. However, this process can be time-consuming and performance intensive when used on large tables. Consider calling the `ttOptUpdateStats` built-in procedure in parallel when updating the statistics for multiple large tables.

Note: A TimesTen table is considered a small table when it contains less than 1 million rows. A TimesTen table is considered a large table when it contains over 100 million rows.

Call the `ttOptUpdateStats` built-in procedure for all of the large tables where you want to update table statistics. Make sure to call each `ttOptUpdateStats` built-in procedure in parallel. For more information on the `ttOptUpdateStats` built-in procedure, see "ttOptUpdateStats" in the *Oracle TimesTen In-Memory Database Reference*.

```
Command> call ttOptUpdateStats('table1',0,0);
Command> call ttOptUpdateStats('table2',0,0);
...
...
Command> call ttOptUpdateStats('finaltable',0,0);
```

Once the `ttOptUpdateStats` built-in procedure calls have completed, determine how many transactions are accessing the large TimesTen tables for which you updated table statistics. During low transaction load times execute the `ttOptCmdCacheInvalidate('',1)` built-in procedure. For more information on the `ttOptCmdCacheInvalidate` built-in procedure, see "ttOptCmdCacheInvalidate" in the *Oracle TimesTen In-Memory Database Reference*. During high transaction load times execute the following built-in procedures and make sure to call each `ttOptCmdCacheInvalidate` built-in procedure in parallel:

```
Command> call ttOptCmdCacheInvalidate('table1',1);
Command> call ttOptCmdCacheInvalidate('table2',1);
...
...
Command> call ttOptCmdCacheInvalidate('finaltable',1);
```

The table statistics of your tables are now up to date and compiled commands in the SQL command cache are invalidated.

Create script to regenerate current table statistics

Performance impact: Variable

You can generate a SQL script with the `ttOptStatsExport` built-in procedure from which you can restore the table statistics to the current state. When you apply these statements, you re-create the same environment. Recreating the table statistics could be used for diagnosing SQL performance.

Call the `ttOptStatsExport` built-in procedure to return the set of statements required to restore the table statistics to the current state. If no table is specified, `ttOptStatsExport` returns the set of statements required to restore the table statistics for all user tables that the calling user has permission to access.

Note: For more information and syntax for this built-in procedure, see "ttOptStatsExport" in the *Oracle TimesTen In-Memory Database Reference*.

The following example returns a set of built-in procedure commands that would be required to be executed to restore the statistics for the `employees` table:

```
Command> CALL ttOptStatsExport('HR.employees');

< call ttoptsetcolIntvlstats('HR.EMPLOYEES', 'EMPLOYEE_ID', 0, (6, 0, 107, 107,
 (20, 20, 1 ,100, 120, 101), (20, 20, 1 ,121, 141, 122), (20, 20, 1 ,142, 162,
143), (20, 20, 1 ,163, 183, 164), (20, 20, 1 ,184, 204, 185), (1, 1, 1 ,205, 206,
205))); >
< call ttoptsetcolIntvlstats('HR.EMPLOYEES', 'FIRST_NAME', 0, (1, 0, 89, 107,
(89, 107, 0, 'Adam', 'Winston', 'Adam'))); >
< call ttoptsetcolIntvlstats('HR.EMPLOYEES', 'LAST_NAME', 0, (1, 0, 97, 107, (97,
107, 0, 'Abel', 'Zlotkey', 'Abel'))); >
< call ttoptsetcolIntvlstats('HR.EMPLOYEES', 'EMAIL', 0, (6, 0, 107, 107, (20,
20, 1, 'ABANDA', 'DGREENE', 'ABULL'), (20, 20, 1, 'DLEE', 'JKING', 'DLORENTZ'),
(20, 20, 1, 'JLANDRY', 'LOZER', 'JLIVINGS'), (20, 20, 1, 'LPOPP', 'RMATOS',
'LSMITH'), (20, 20, 1, 'RPERKINS', 'WGIETZ', 'SANDE'), (1, 1, 1, 'WSMITH',
'WTAYLOR', 'WSMITH'))); >
< call ttoptsetcolIntvlstats('HR.EMPLOYEES', 'PHONE_NUMBER', 0, (1, 0, 103, 107,
(103, 107, 0, '011.44.1343.329268', '650.509.4876', '011.44.1343.329268'))); >
< call ttoptsetcolIntvlstats('HR.EMPLOYEES', 'HIRE_DATE', 0, (1, 0, 90, 107, (90,
107, 0 ,'1987-06-17 00:00:00', '2000-04-21 00:00:00', '1987-06-17 00:00:00'))); >
< call ttoptsetcolIntvlstats('HR.EMPLOYEES', 'JOB_ID', 0, (4, 0, 19, 107, (11,
16, 5, 'AC_ACCOUNT', 'PR_REP', 'FI_ACCOUNT'), (3, 11, 30, 'PU_CLERK', 'SA_REP',
'SA_REP'), (1, 20, 20, 'SH_CLERK', 'ST_CLERK', 'ST_CLERK'), (0, 0, 5, 'ST_MAN',
'ST_MAN', 'ST_MAN'))); >
< call ttoptsetcolIntvlstats('HR.EMPLOYEES', 'SALARY', 0, (1, 0, 57, 107, (57,
107, 0 ,2100, 24000, 2100))); >
< call ttoptsetcolIntvlstats('HR.EMPLOYEES', 'COMMISSION_PCT', 0, (1, 72, 7, 107,
(7, 35, 0 ,0.1, 0.4, 0.1))); >
< call ttoptsetcolIntvlstats('HR.EMPLOYEES', 'MANAGER_ID', 0, (1, 1, 18, 107,
(18, 106, 0 ,100, 205, 100))); >
< call ttoptsetcolIntvlstats('HR.EMPLOYEES', 'DEPARTMENT_ID', 0, (3, 1, 11, 107,
(4, 10, 45 ,10, 50, 50), (2, 6, 34 ,60, 80, 80), (2, 5, 6 ,90, 110, 100))); >
< call ttoptsettblstats('HR.EMPLOYEES', 107, 0); >
12 rows found.
```

Control the invalidation of commands in the SQL command cache

Performance impact: Variable

TimesTen caches compiled commands in the SQL command cache. These commands can be invalidated. An invalidated command is usually reprepared automatically just before it is re-executed. A single command may be prepared several times.

Note: See "When optimization occurs" on page 9-1 for more information on how commands are automatically invalidated.

When you compute statistics, the process of updating and compiling commands may compete for the same locks on certain tables. If statistics are collected in multiple transactions and commands are invalidated after each statistics update, the following issues may occur:

- A join query that references multiple tables might be invalidated and recompiled more than once.
- Locks needed for recompilation could interfere with updating statistics, which could result in a deadlock.

You can avoid these issues by controlling when commands are invalidated in the SQL command cache. In addition, you may want to hold off invalidation of all commands if you know that the table and index cardinalities will be changing significantly.

You can control invalidation of the commands, as follows:

1. Compute statistics without invalidating the commands in the SQL command cache. Set the `invalidate` option to 0 in either the `ttIsql statsupdate` command, the `ttOptUpdateStats` built-in procedure, or the `ttOptEstimateStats` built-in procedure
2. Manually invalidate the commands in the SQL command cache once all statistics have been compiled with the `ttOptCmdCacheInvalidate` built-in procedure.

The `ttOptCmdCacheInvalidate` built-in procedure can invalidate commands associated solely with a table or all commands within the SQL command cache. In addition, you can specify whether the invalidated commands are to be recompiled or marked as unusable.

Note: For complete details on when to optimally calculate statistics, see "Compute exact or estimated statistics" on page 10-18. In addition, see "ttIsql," "ttOptUpdateStats," "ttOptEstimateStats, " or "ttOptCmdCacheInvalidate" in the *Oracle TimesTen In-Memory Database Reference*.

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 `VARCHAR2` 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 `VARCHAR2` and `VARBINARY` values in the column to be dropped.

Avoid nested queries

Performance impact: Variable

If you can, it is recommended that you should rewrite your query to avoid nested queries that need materialization of many rows.

The following are examples of nested queries that may need to be materialized and result in multiple rows:

- Aggregate nested query with `GROUP BY`
- Nested queries that reference `ROWNUM`
- Union, intersect, or minus nested queries
- Nested queries with `ORDER BY`

For example, the following aggregate nested query results in an expensive performance impact:

```
select * from (select sum(x1) sum1 from t1 group by y1),
 (select sum(x2) sum2 from t2 group by y2) where sum1=sum2;
```

The following is an example of a nested query that references `ROWNUM`:

```
select * from (select rownum rc, x1 from t1 where x1>100),
 (select rownum rc, x2 from t2 where x2>100) where x1=x2;
```

The following is an example of a union nested query:

```
select * from (select x1 from t1 union select x2 from t2),
 (select x3 from t3 group by x3) where x1=x3;
```

For more information on subqueries, see "Subqueries" in the *Oracle TimesTen In-Memory Database SQL Reference*.

Prepare statements in advance

Performance impact: Variable

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

Performance impact: Large

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.

Store data efficiently with table compression

Performance impact: Large

TimesTen provides the ability to compress tables at the column level, which stores the data more efficiently. This mechanism provides space reduction for tables by eliminating the redundant storage of duplicate values within columns.

When compressing columns of a TimesTen table, consider the following:

- Compress a column if values are repeated throughout such as the name of countries or states.
- Compress a column group if you often access multiple columns together.
- Do not compress columns that contain data types that require a small amount of storage such as `TT_TINYINT`.
- TimesTen does not compress `NULL` values.

For more information on columnar compression, see "In-memory columnar compression of tables" in the *Oracle TimesTen In-Memory Database SQL Reference*.

Compressed column groups can be added at the time of table creation or added later using `ALTER TABLE`. You can drop the entire compressed column group with the `ALTER TABLE` statement. For more information, see "ALTER TABLE" and "CREATE TABLE" in the *Oracle TimesTen In-Memory Database SQL Reference*.

You can call the `ttSize` built-in procedure to review the level of compression that TimesTen achieved on your compressed table. For more information on the `ttSize` built-in procedure, see "ttSize" in the *Oracle TimesTen In-Memory Database Reference*.

Control read optimization during concurrent write operations

Performance impact: Variable

TimesTen concurrently processes read and write queries optimally. Your read queries can be optimized for read-only concurrency when you use transaction level optimizer hints such as `ttOptSetFlag ('tblLock',1)` or statement level optimizer hints such as `/*+ tt_tbllock(1) tt_rowlock(0) */`. Write queries that operate concurrently with read optimized queries may result in contention.

You can control read optimization during periods of concurrent write operations with the `ttDbWriteConcurrencyModeSet` built-in procedure. This built-in procedure enables you to switch between a standard mode and an enhanced write concurrent mode. In the standard mode, the optimizer respects read optimization hints. In the enhanced write concurrent mode, the optimizer ignores read optimization hints and does not use shared read table locks or write table locks.

Note: For more information about table locking, see "Locking granularities" on page 7-7.

For more information about statement level optimizer hints, see "Statement level optimizer hints" in the *Oracle TimesTen In-Memory Database SQL Reference*.

For more information about transaction level optimizer hints, see "ttOptSetFlag" in the *Oracle TimesTen In-Memory Database Reference*.

Set the *`mode`* of the `ttDbWriteConcurrencyModeSet` built-in procedure to `1` to enable the enhanced write concurrent mode and disable read optimization. Set the *`mode`* to `0` to disable the enhanced write concurrent mode and re-enable read optimization.

When the *`mode`* is set to `1`, all transaction and statement table lock optimizer hints are ignored. This affects the following:

- Shared read table-level locks for `SELECT` query and subqueries that are triggered by optimizer hints.
- Write table locks for DML statements that are triggered by optimizer hints.

Regardless of the *`mode`* setting, table locks that are not triggered by optimizer hints are not affected.

Set the *`wait`* of the `ttDbWriteConcurrencyModeSet` built-in procedure to `0` to perform a mode switch without notifications. Set the *`wait`* of the `ttDbWriteConcurrencyModeSet` built-in procedure to `1` to force the built-in procedure to wait until the mode transition is complete.

Execution of certain SQL statements causes the mode of the `ttdbWriteConcurrencyModeSet` built-in procedure to remain in transition. Such SQL statements must match the following two conditions:

- Affected by the write concurrency mode.
- Compiled in a different write concurrency mode.

The mode of the `ttdbWriteConcurrencyModeSet` built-in procedure remains in transition until all such SQL statements complete. The `ttDbWriteConcurrencyModeSet` built-in procedure uses lock acquisition to wait during the mode transition. An error is returned if the `ttDbWriteConcurrencyModeSet` built-in procedure is not granted a lock within the timeout interval of the current connection.

Note: For more information about the `ttDbWriteConcurrencyModeSet`, `ttLockWait`, `ttDbWriteConcurrencyModeGet` built-in procedures, see "ttDbWriteConcurrencyModeSet", "ttLockWait", and "ttDbWriteConcurrencyModeGet" in the *Oracle TimesTen In-Memory Database Reference*.

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
- Avoid large DELETE statements
- Increase the commit buffer cache size

Size transactions appropriately

Performance impact: Large

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" on page 10-27.

Long transactions perform fewer disk writes per unit of time than short transactions. However, long transactions also can reduce concurrency, as discussed in Chapter 7, "Transaction Management".

- If only one connection is active on a database, 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 the natural length, as determined by requirements for atomicity and durability.

Use durable commits appropriately

Performance impact: Large

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 drivers only write data into cache memory or write to disk some time after the operating system receives the write completion notice. 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 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 could 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 `txn.commits.durable` column of the `SYS.SYSTEMSTATS` table indicates the number of transactions that were durably committed.

Avoid frequent checkpoints

Performance impact: Large

Applications that are connected to a database for a long period of time occasionally need to call the `ttCkpt` built-in procedure to checkpoint the database 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 database.

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 database 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 `ckpt.completed` column of the `SYS.SYSTEMSTATS` table indicates how often checkpoints have successfully completed.

Turn off autocommit mode

Performance impact: Large

`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 `txn.commits.count` column of the `SYS.SYSTEMSTATS` 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

Performance impact: Large

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 `txn.rollbacks` column of the `SYS.SYSTEMSTATS` table indicates the number of transactions that were rolled back.

Avoid large DELETE statements

Performance impact: Large

Consider the following ways to avoid large delete statements:

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

Avoid DELETE FROM statements

If you attempt to delete a large number of rows (100,000 or more) from a table with a single SQL statement the operation can take a long time. TimesTen logs each row deleted, in case the operation needs to be rolled back, and writing all of those log records can be very time-consuming because it is a disk-bound operation.

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

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

For more information about the `DELETE` SQL statement, see "DELETE" in the *Oracle TimesTen In-Memory Database SQL Reference*.

Prefer the TRUNCATE TABLE statement

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

For more information about the `TRUNCATE TABLE` SQL statement, see "TRUNCATE TABLE" in the *Oracle TimesTen In-Memory Database SQL Reference*.

Consider using the DELETE FIRST clause

If you want to delete a large number of rows and `TRUNCATE TABLE` is not appropriate, consider using the `DELETE FIRST` *NumRows* clause to delete rows from a table in batches. The `DELETE FIRST` *NumRows* syntax allows you to change "`DELETE FROM` *`TableName`* `WHERE` . . ." into a sequence of "`DELETE FIRST` 10000 `FROM` *`TableName`* `WHERE` . . ." operations.

By splitting a large `DELETE` operation into a batch of smaller operations, the rows will be deleted much faster, and the overall concurrency of the system and replication will not be affected.

For more information about the `DELETE FIRST` clause, see "DELETE" in the *Oracle TimesTen In-Memory Database SQL Reference*.

Increase the commit buffer cache size

Performance impact: Large

TimesTen resource cleanup occurs during the reclaim phase of a transaction commit. During reclaim, TimesTen reexamines all the transaction log records starting from the beginning of the transaction to determine the reclaim operations that must be performed.

The reclaim phase of a large transaction commit results in a large amount of processing and is very resource intensive. You can improve performance, however, by increasing the maximum size of the commit buffer, which is the cache of transaction log records used during reclaim operations.

You can use the TimesTen `CommitBufferSizeMax` connection attribute to specify the maximum size of the commit buffer, in megabytes. This setting has the scope of your current session.

See "Configuring the commit buffer for reclaim operations" on page 7-21 for information.

Recovery tuning

The following sections include tips for improving performance of database recovery after database shutdown or system failure:

- Set RecoveryThreads
- Set CkptReadThreads

Set RecoveryThreads

Performance impact: Large

Set the `RecoveryThreads` attribute to the number of indexes or CPUs to improve recovery performance.

Set CkptReadThreads

Performance impact: Large

When a database has large checkpoint files (hundreds of gigabytes), first connection or recovery operations may not perform well and, in extreme cases, may take hours to complete. To improve recovery performance when you have large checkpoint files, use the `CkptReadThreads` connection attribute to increase the number of concurrent threads used for reading the checkpoint files during the loading of the database into memory.

For more information on the `CkptReadThreads` connection attribute, see "CkptReadThreads" in the *Oracle TimesTen In-Memory Database Reference*.

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

- 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 enables you to control how it executes, including options to vary the number of processes that execute TimesTen operations and the transaction mix of `SELECT`s, `UPDATE`s, and `INSERT`s, 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 the 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 greatly inhibits 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.locks_granted.wait` fields in the `SYS.SYSTEMSTATS` 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 executes well, but an 8-processor application does not perform well. The contention between the active threads is 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 system 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 `stmt.prepares.count` and `stmt.reprepares.count` columns of the `SYS.SYSTEMSTATS` 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 `connections.established.count` column in the `SYS.SYSTEMSTATS` table. If the field has a high value, modify your application to do connection pooling, so that connects and disconnects are rare events.

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

Performance impact: Large

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.

Cache and replication tuning

For recommendations on improving performance for when using a replication scheme, see "Improving replication performance" in the *Oracle TimesTen In-Memory Database Replication Guide*.

For recommendations on improving performance when using cache groups, see "Cache Performance" in the *Oracle TimesTen Application-Tier Database Cache User's Guide*.

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 database.

backup instance

A set of files containing backup information for a given database, residing at a given backup path. See also "backup path", "full backup" and "incremental backup".

backup path

The location of a database, 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 system, 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 database 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 system that refers to a server DSN on a server system.

data source name, server

A system data source name (system DSN) defined on a server system. 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 system.

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 database backup procedure in which a complete copy of a database is created. Typically, the first backup of a database 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".

inline column

A column whose values are physically stored together with the other column values of a row.

incremental backup

A database backup procedure in which an existing backup is augmented with all the transaction 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 to be loaded 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 TimesTen Cache to send table or row modifications from a TimesTen 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 database 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 database 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 database. 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