Oracle® TimesTen In-Memory Database

Operations Guide

Release 22.1
F35391-05
June 2024

ORACLE

Oracle TimesTen In-Memory Database Operations Guide, Release 22.1

F35391-05

Copyright © 1996, 2024, Oracle and/or its affiliates.

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

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

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

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

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

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

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

Contents

What's New

1 Connecting to a TimesTen Database

2 Managing TimesTen Databases

3 Working with the TimesTen Client and Server

4 Working with the TimesTen Daemon

6 Using the ttIsql Utility

7 Transaction Management

8 Working with Data in a TimesTen Database

9 The TimesTen Query Optimizer

10 TimesTen Database Performance Tuning

About This Content

This guide provides background information to help you understand how TimesTen works, as well as step-by-step instructions 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).

Documentation Accessibility

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

Access to Oracle Support

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

Conventions

The following text conventions are used in this document.

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

What's New

This section summarizes the new features and functionality of Oracle TimesTen In-Memory Database Release 22.1 that are documented in this guide, providing links into the guide for more information.

New features in Release 22.1.1.17.0

- Previously, you could only provide credentials when opening a connection to the TimesTen database by providing the user name and password individually in a client DSN or using connection attributes. Now, you can provide user credentials within an Oracle Wallet where the wallet location is provided when opening a connection. The preferred method is storing credentials in an Oracle Wallet.

  See Providing User Credentials When Connecting.

New features in Release 22.1.1.3.0

- If an application process wants to use both direct and client/server TimesTen drivers, then it should link with an ODBC driver manager. In this release, a TimesTen-specific driver manager is provided. The TimesTen driver manager is a transparent, low overhead option designed specifically for this use case and supports all TimesTen functionality. See Connecting Using an ODBC Driver Manager.

New features in Release 22.1.1.1.0

- Executing a `SELECT` statement returns a result set (if there is one). The client driver stores a maximum number of rows from the result set that fits into a result set buffer. You can improve performance by adjusting the maximum size of the result set buffer or the maximum number of rows that can be stored into a result set buffer. See "Sizing the Client Result Set Buffer" for details.
- There are now two LRU aging policies for TimesTen Classic:
  - LRU aging based on set thresholds for the amount of permanent memory in use.
  - LRU aging based on row thresholds for a specified root tables of your cache groups.

  See "Usage-Based Aging" and "Defining LRU Aging Based on Row Thresholds for Tables" for details.
- The `ttPageLevelTableInfo` built-in procedure shows the page allocation for each table to determine when TimesTen is reusing empty slots and freeing empty pages or if new pages are allocated to store new rows. See "Determining the Effectiveness of Aging" for details.
- There is a new RAM policy of enduring. In addition to being able to manually load and unload data from the database (the same as the manual RAM policy), you can also attach, detach, and free the shared memory segment. When you detach, theshared memory segment still exists and can be reused when you attach to it. See "Specifying a RAM Policy" and "Detaching, Attaching, and Freeing the Shared Memory Segment" for details.

- For TimesTen Classic, the root user can set up systemd to use for automatic management (including starting and stopping) of the TimesTen daemon. See "Automatically Starting the TimesTen Daemon" and "Using systemd to Automatically Manage the TimesTen Daemon" for details.

1

Connecting to a TimesTen Database

A TimesTen database is a collection of objects such as tables, views, and sequences. You can access and manipulate the TimesTen database through SQL statements after connecting to the database.

In TimesTen Classic, if your database does not exist, then TimesTen Classic creates the database with the specified connection attributes when the instance administrator connects to the database.

In TimesTen Scaleout, you create and open the database before applications can connect to the database. See Managing a Database in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

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

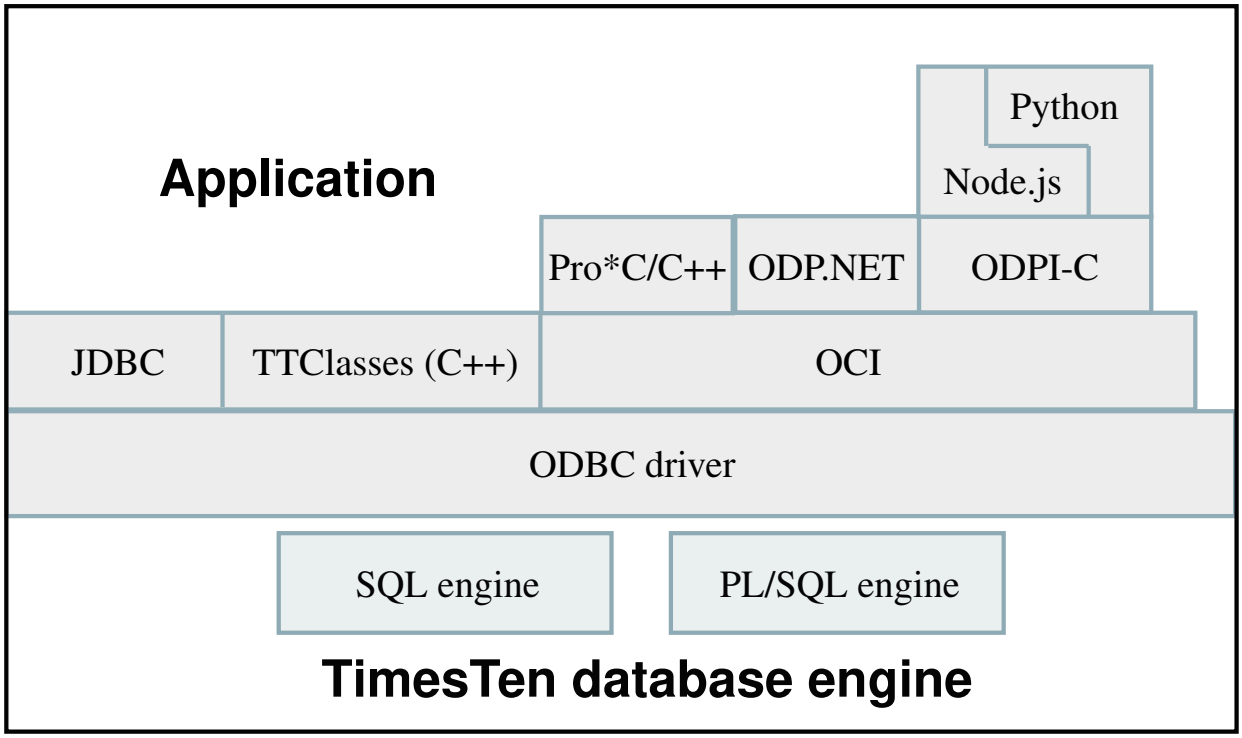
- Run an application that uses the database.
- Use the `ttIsql` utility to connect to the database and run a SQL file or start an interactive SQL session. See Using ttIsql in Batch Mode or Interactive Mode.

This chapter describes how to configure to connect to the TimesTen database. The connection attributes include management configuration details for your TimesTen database.

Connecting to TimesTen with ODBC and JDBC Drivers

An application can use different interfaces and drivers to access the TimesTen database.

Figure 1-1 Application Access to TimesTen Database Diagram



- Open source languages interact with TimesTen through the Oracle Database Programming Interface for C (ODPI-C). The languages currently supported are Python and Node.js. See Getting Started in the *Oracle TimesTen In-Memory Database Open Source Languages Support Guide*.
- C applications interact with TimesTen by linking with a TimesTen direct or client/server driver, by linking with the TimesTen driver manager, by linking with a generic driver manager, or by using the OCI or Pro*C/C++ APIs that indirectly access the TimesTen ODBC driver.
- Java applications interact with TimesTen by using the TimesTen JDBC driver.
- C++ applications can interact with TimesTen directly through ODBC, through a TimesTen-provided set of classes called TTClasses or by using the OCI or Pro*C/C++ interfaces that indirectly access the TimesTen ODBC drivers.
- C# applications interact with TimesTen through Oracle Data Provider for .NET support for the TimesTen database.

All of these interfaces ultimately use the TimesTen ODBC driver to access a TimesTen database. These interfaces can use the TimesTen ODBC direct driver, the TimesTen ODBC client driver, the TimesTen driver manager, or a generic ODBC driver manager. See TimesTen Connection Options in the *Oracle TimesTen In-Memory Database Introduction*.

The following sections describe how to define TimesTen databases:

- Connecting Using TimesTen ODBC Drivers
- Connecting Using an ODBC Driver Manager
- Connecting Using the TimesTen JDBC Driver and Driver Manager

Connecting Using TimesTen ODBC Drivers

Your application can connect to the TimesTen database using TimesTen ODBC drivers.

TimesTen includes the following TimesTen 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.

An application that links directly with a TimesTen 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 a TimesTen ODBC driver can connect to multiple databases at the same time. The TimesTen direct and client drivers support multiple connections to multiple TimesTen databases.

> **Note:**
>
> This option offers less flexibility but better performance than linking with a driver manager. However, an application should use a driver manager when connecting with both direct and client-server connections. If you are going to use a driver manager, use the TimesTen driver manager over a generic ODBC driver manager to maximize performance and functionality.

TimesTen includes the following two versions of the TimesTen Data Manager driver (direct 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 Linux and UNIX, the TimesTen debug libraries are compiled with the `-g` option to display additional debug information.

For TimesTen Classic support on Windows, you can install the "TimesTen Client 22.1" driver after choosing either the Typical or Custom setup.

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

Table 1-1 lists the TimesTen ODBC drivers for Linux and UNIX platforms.

Table 1-1 ODBC Drivers Provided for Linux and UNIX Platforms

Platform	Version	Location and name
Linux Solaris x86 Solaris Sparc	Production	*timesten_home*/install/lib/libtten.so TimesTen Data Manager 22.1 driver.

Table 1-1 (Cont.) ODBC Drivers Provided for Linux and UNIX Platforms

Platform	Version	Location and name
Linux Solaris x86 Soliaris Sparc	Debug	*timesten_home*/install/lib/libttenD.so TimesTen Data Manager 22.1 Debug driver.
Linux Solaris x86 Soliaris Sparc	Client	*timesten_home*/install/lib/libttclient.so TimesTen Client 22.1 driver.
AIX	Production	*timesten_home*/install/lib/libtten.a TimesTen Data Manager 22.1 driver.
AIX	Debug	*timesten_home*/install/lib/libttenD.a TimesTen Data Manager 22.1 Debug driver.
AIX	Client	*timesten_home*/install/lib/libttclient.a TimesTen Client 22.1 driver.
macOS	Client	*timesten_home*/install/lib/libttclient.dylib TimesTen Client 22.1 driver.

Connecting Using an ODBC Driver Manager

If an application process wants to use both direct and client/server TimesTen drivers, then it should link with an ODBC driver manager.

- The TimesTen driver manager is a transparent, low overhead option designed specifically for this use case. The TimesTen driver manager supports all TimesTen functionality. See Introduction to the TimesTen Driver Manager in the *Oracle TimesTen In-Memory Database C Developer's Guide*.
- Alternatively, your application can be linked with a generic ODBC driver manager. A generic ODBC driver manager typically has a significant performance impact and inhibits the use of some TimesTen functionality (such as XLA, the Routing API, the Utility API and TimesTen ODBC extensions).

Both the TimesTen driver manager and generic ODBC driver manager dynamically load an ODBC driver at runtime.

Connecting Using the TimesTen JDBC Driver and Driver Manager

The TimesTen JDBC driver 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.

As shown in Figure 1-1, the TimesTen JDBC driver uses the ODBC driver to access TimesTen databases. For each JDBC method, the driver runs 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. Java provides a driver manager that can support multiple drivers connecting to separate databases. The Java `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, an application can load both a TimesTen client JDBC driver and an Oracle database JDBC driver. Then, Java applications can access either the TimesTen or Oracle databases.

For a list of the Java functions supported by TimesTen, see Support for Interfaces in the java.sql Package in *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.

In TimesTen Classic, 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. See Setting Up a Default DSN in TimesTen Classic.

> **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.
  - For TimesTen Classic on Linux and 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.

    TimesTen Classic supports data sources for both TimesTen Data Manager driver and the TimesTen Client driver in the `.odbc.ini` file.
  - For TimesTen Scaleout, define user DSNs within the appropriate database definitions and connectables. See Managing a Database in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

  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.
- 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.
  - For TimesTen Classic on Linux and 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 an installation, the file is located in *`timesten_home/conf/sys.odbc.ini`*.
    - If not found in any of these locations, TimesTen looks on the system for the `/etc/odbc.ini` file.
  - For TimesTen Scaleout, create system DSNs within the appropriate database definitions and connectables. See Managing a Database in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

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 on a Linux or UNIX host. You can use either the production version or 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 `/disk1/databases/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 `/disk1/databases/AdminDS.ds0`.

A Data Manager DSN that refers to a given TimesTen database must be defined on the same system on which the database resides. 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.

- *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 Linux and 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.

See Working with the TimesTen Client and Server for more information about Client DSNs and Server DSNs.

Connection Attributes for Data Manager DSNs or Server DSNs

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

Common types of connection attributes:

> **Note:**
>
> For a complete description of all attributes, see List of Connection Attributes in *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. Placing the transaction log files and checkpoint files on different file systems can improve system performance.
  - `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 objects. TimesTen only writes the permanent memory region to the file system 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 running statements.

  > **Note:**
  >
  > Your system must have sufficient main memory to accommodate the entire database. For more details on setting region sizes, see Specifying the Memory Region Sizes of a Database.

- **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:

> **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.

  - `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 a user name. When you do not specify a 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.
  - `PWDCrypt`: Specifies the encrypted password that corresponds with the specified `UID`.
  - `PwdWallet`: TimesTen enables you to store user names and associated passwords in an Oracle Wallet. This is the most secure and preferred method of providing credentials for connecting to a TimesTen database. The `PwdWallet` connection attribute is the path to the location of the wallet, from which TimesTen retrieves the password for the specified user name.
  - > **Note:**
    >
    > For more information, see UID and PWD, PWDCrypt, and PwdWallet in *Oracle TimesTen In-Memory Database Reference*. See Authentication in TimesTen in *Oracle TimesTen In-Memory Database Security Guide*.

- **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 for a description of the connection attributes that can be used with the TimesTen client ODBC driver.

Creating a DSN on Linux and UNIX for TimesTen Classic

You can create a DSN in TimesTen Classic in an odbc.ini file for Linux or UNIX platforms.

This section includes the following topics for creating a DSN for TimesTen Classic. In TimesTen Scaleout, you would create the DSN within the appropriate database definitions and connectables. See Managing a Database in the *Oracle TimesTen In-Memory Database Scaleout User's Guide* for details.

- 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 for TimesTen Classic.

Create a User or System odbc.ini File

User and system DSNs are defined in one of several `odbc.ini` file types.

On Linux and 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 *`timesten_home`*`/conf/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. 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 TimesTen ODBC driver.

   > **Note:**
   >
   > JDBC users need to specify the TimesTen ODBC driver to be used by the JDBC driver, as described in Connecting Using the TimesTen JDBC Driver and Driver Manager.

   To set the TimesTen ODBC 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, where `/disk1/timesten` is the *`timesten_home`*:

   ```
   [AdminDS]
   Driver=/disk1/timesten/lib/libtten.so
   ```

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

3. Specify the database directory path and prefix in the `odbc.ini` file. The following example defines `/disk1/databases` as the database directory path and `FixedDs` as the prefix for the database files:

   ```
   DataStore=/disk1/databases/FixedDs
   ```

> **Note:**
>
> For more information, see Specifying Data Source Names to Identify TimesTen Databases.

The database directory path can use environment variables, as discussed in Using Environment Variables in Database Path Names.

4. Choose a database character set. The following example defines the database character set in the `odbc.ini` file as `AL32UTF8`:

```
DatabaseCharacterSet=AL32UTF8
```

> **Note:**
>
> See Choosing a Database Character Set.

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 for TimesTen Classic.

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 `sys.odbc.ini` file.

Defining Client and Server DSNs

There are specific methods to define client or server DSNs for each platform.

For directions on how to define client or server DSNs for each platform, see Defining Server DSNs for TimesTen Server on a Linux or UNIX System and Defining Client DSNs on a TimesTen Client System.

Resolution Path for a DSN

TimesTen uses certain rules when resolving a DSN.

> **Note:**
>
> - If a user DSN and a system DSN with the same name exist, TimesTen retrieves the user DSN.
> - On Linux and 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 Linux and UNIX, the file is located in *`timesten_home`*`/conf/sys.odbc.ini`.

DSN Examples for TimesTen Classic

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

- Creating Multiple DSNs to a Single TimesTen Classic Database
- Specifying PL/SQL Connection Attributes in a DSN
- Setting Up a Default DSN in TimesTen Classic

Creating Multiple DSNs to a Single TimesTen Classic Database

You can create two or more DSNs that refer to the same TimesTen Classic 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 `AL32UTF8` 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`.

The text in square brackets is the data source name.

AdminDSN is created with the AL32UTF8 database character set and AL32UTF8 as the connection character set. GlobalDSN is created with the AL32UTF8 database character and AL32UTF8 as the connection character set. Note that /disk1/timesten is the *timesten_home*.

```
[AdminDSN]
Driver=/disk1/timesten/lib/libtten.so
Datastore=/disk1/databases/TTDS
DatabaseCharacterSet=AL32UTF8
ConnectionCharacterSet=AL32UTF8
CacheAdminWallet=1

[GlobalDSN]
Driver=/disk1/timesten/lib/libtten.so
DataStore=/disk1/databases/TTDS
DatabaseCharacterSet=AL32UTF8
ConnectionCharacterSet=AL32UTF8
CacheAdminWallet=1
```

Specifying PL/SQL Connection Attributes in a DSN

For TimesTen Classic, PL/SQL connection attributes exist both as first and general connection attributes.

- PLSQL_MEMORY_ADDRESS - A first connection attribute that 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 - A first connection attribute that determines the size, in megabytes, of the PL/SQL shared memory segment.
- PLSCOPE_SETTINGS - A general connection attribute that controls whether the PL/SQL compiler generates cross-reference information.
- PLSQL_OPTIMIZE_LEVEL - A general connection attribute that sets the optimization level that is used to compile PL/SQL library units.

This example creates the PLdsn DSN and sets the PL/SQL shared memory segment size to 32 MB.

```
[PLdsn]
Datastore=/disk1/databases/PLdsn
PermSize=128
DatabaseCharacterSet=AL32UTF8
ConnectionCharacterSet=AL32UTF8
PLSQL_MEMORY_SIZE=32
CacheAdminWallet=1
```

For more examples and a complete list of PL/SQL connection attributes see PL/SQL Connection Attributes in *Oracle TimesTen In-Memory Database PL/SQL Developer's Guide*.

Setting Up a Default DSN in TimesTen Classic

You can add an optional default data source definition in TimesTen Classic.

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 Classic 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.

When connecting, TimesTen Classic 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 Classic 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, 2024 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 2021

Daemon pid 13623 port 16138 instance myhost
TimesTen Server pid 13632 started on port 16140
------------------------------------------------------------------------
Data store /timesten/install/sample_db/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. Note that `/disk1/timesten` is the *`timesten_home`*.

```
[ODBC Data Sources]
database1=TimesTen 22.1 Driver

[default]
Driver=/disk1/timesten/lib/libtten.so
DataStore=/disk1/timesten/sample_db/DemoDataStore/default
PermSize=128
TempSize=64
DatabaseCharacterSet=AL32UTF8
CacheAdminWallet=1

[database1]
Driver=/disk1/timesten/lib/libtten.so
DataStore=/disk1/timesten/sample_db/DemoDataStore/database1
PermSize=128
TempSize=64
DatabaseCharacterSet=AL32UTF8
CacheAdminWallet=1
```

odbc.ini File Entry Descriptions

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

> **Note:**
>
> In TimesTen Scaleout, use the `ttGridAdmin gridClientExport` command to create a system or user `odbc.ini` file to be used by the TimesTen Client instance. See Establishing Client Connections from a TimesTen Client in *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

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 could have the format shown in Table 1-2.

Table 1-2 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 driver that is linked with the data source.
`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. See Choosing a Database Character Set.
`CacheAdminWallet=1`	Designates that when TimesTen registers the cache administration user credentials, the credentials are saved in an internally managed Oracle Wallet rather than in memory. Setting the `CacheAdminWallet` connection attribute to 1 is recommended when using cache, but optional. See CacheAdminWallet in *Oracle TimesTen In-Memory Database Reference*
Optional attributes	See Connection Attributes in *Oracle TimesTen In-Memory Database Reference*.

For example, the `database1` DSN could have a specification that includes the following. Note that `/disk1/timesten` is the *timesten_home*.

```
[database1]
Driver=/disk1/timesten/lib/libtten.so
DataStore=/disk1/timesten/sample_db/DemoDataStore/database1
CacheAdminWallet=1
```

The database specification for TimesTen client DSN could have the format shown in Table 1-3.

> **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 for TimesTen Server on a Linux or UNIX System and Defining Client DSNs on a TimesTen Client System.

Table 1-3 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 driver attributes are ignored for TimesTen client DSNs.

For example, the client/server data source `database1CS` that connects to `database1` on the TimesTen server `ttserver` could have a data source specification that includes the following:

```
[database1CS]
TTC_Server=ttserver
TTC_SERVER_DSN=database1
TTC_Timeout=30
```

odbc.ini File Example

The following example shows portions of a Linux or UNIX `.odbc.ini` file. Note that `/disk1/timesten` is the *timesten_home*.

```
...
[ODBC Data Sources]
database1=TimesTen 22.1 Driver
...

[database1]
Driver=/disk1/timesten/lib/libtten.so
DataStore=/disk1/timesten/sample_db/DemoDataStore/database1
PermSize=128
TempSize=64
DatabaseCharacterSet=AL32UTF8
CacheAdminWallet=1
...

##########################################################################
# This following sample definitions should be in the .odbc.ini file
# that is used for the TimesTen 22.1 Client.
# The Server Name is set in the TTC_SERVER attribute.
# The Server DSN is set in the TTC_SERVER_DSN attribute.
##########################################################################

[ODBC Data Sources]
database1CS=TimesTen 22.1 Client Driver
...
```

```
[database1CS]
TTC_SERVER=localhost
TTC_SERVER_DSN=database1
...
```

Providing User Credentials When Connecting

User names and their respective passwords are required when connecting to the TimesTen database.

Supply the user credentials in the connection string either by:

- Providing them from within an Oracle Wallet. This method requires you to first save the credentials in an Oracle Wallet with the `ttUser` utility. After creating the wallet, the particular wallet is identified by `UID` and `PWDWallet` connection attributes on the connection string. This is the preferred method as it is more secure. See Connect Using an Oracle Wallet with Credentials.
- Providing the user name and password on the connection string. Specify the user name in the `UID` connection attribute and specify the user password in the `PWD` or `PWDCrypt` connection attributes. See Connect Using Connection Attributes for Credentials.

Connect Using an Oracle Wallet with Credentials

You can provide credentials by saving them in an Oracle Wallet, which then can be used for connecting to the database. This is the preferred method as it is more secure.

The following creates the Oracle Wallet `mywallet` in the `/home/terry/wallets` directory. Assuming that `/home/terry/wallets/mywallet` does not exist, run the `ttUser -setPwd` command to save credentials for the user `terry` into the Oracle Wallet `/home/terry/wallets/mywallet`.

```
% ttUser -setPwd -wallet /home/terry/wallets/mywallet -uid terry
Enter password:
```

After the credentials are saved within an Oracle Wallet, you specify the wallet from which to retrieve credentials for your connection specifying the user name with the `UID` connection attribute and the location and name of the Oracle Wallet with the `PwdWallet` connection attribute.

```
connect "dsn=mydb;uid=terry;PwdWallet=/home/terry/wallets/mywallet";
```

When the `PwdWallet` connection attribute is provided, the credentials are retrieved from the wallet specified. For client/server connections, the wallet must exist on the client.

See Providing Both Cache Administration Users and Passwords in *Oracle TimesTen In-Memory Database Security Guide* for more information on providing cache credentials. See PwdWallet and ttUser in *Oracle TimesTen In-Memory Database Reference*.

Connect Using Connection Attributes for Credentials

You can set user names and passwords by providing them with individual connection attributes, which can be used for connecting to a database. However, please note that the preferred and most secure method is to provide credentials from within an Oracle Wallet.

In the connection string, specify the user name in the `UID` connection attribute. Specify the password in the `PWD` connection attribute.

```
% ttIsql "DSN=mydb;UID=sampleuser;PWD=samplepwd"
```

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.

The following sections describe how to use a connection string instead of a DSN:

- Using a Connection String in TimesTen Scaleout
- Using a Connection String in TimesTen Classic

Using a Connection String in TimesTen Scaleout

In TimesTen Scaleout, you can connect to a specific element by specifying in the connection string the address of the host associated with that element, the database name, and a database user with at least `CREATE SESSION` privileges.

See Using a Connection String to Establish a Client Connection in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

Using a Connection String in TimesTen Classic

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 `DataStore`, `Driver` 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 Client Driver.
  - On UNIX systems, the value of the `Driver` attribute should be the pathname of the TimesTen ODBC Driver shared library file (as described in Table 1-1). The file resides in the *`timesten_home`*`/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:

Providing the connection attributes on the connect string from a Linux/UNIX client. Note that `/disk1/timesten` is the *timesten_home*.

```
% ttIsql

Command> connect "Driver=/disk1/timesten/lib/libtten.so;
DataStore=/disk1/databases/database1;DatabaseCharacterSet=AL32UTF8";
```

Providing the connection attributes on the connect string from a Windows client:

```
C:\ ttIsql
Copyright (c) 1996, 2016, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

Command> connect "Driver=TimesTen Client 22.1;
DataStore=/disk1/databases/database1;DatabaseCharacterSet=AL32UTF8";
```

2

Managing TimesTen Databases

Managing a TimesTen database includes the following:

- Opening and Closing the Database for User Connections
- Specifying the Type of Memory for Shared Memory Segment
- Loading and Unloading the Database from Memory
- Detaching, Attaching, and Freeing the Shared Memory Segment
- Specifying a RAM Policy
- Disconnecting from a Database
- Specifying the Memory Region Sizes of a Database
- Storage Provisioning for TimesTen
- Bulk Copy Data Using the ttBulkCp Utility
- Thread Programming with TimesTen
- Defragmenting TimesTen Databases

Opening and Closing the Database for User Connections

For an application to be able to connect to a database, the database needs to be open. When a database is closed, any new user connection attempts fail.

- By default in TimesTen Classic, the database is automatically opened and user connections can connect.
- By default in TimesTen Scaleout, the database is closed until manually opened. See Open the Database for User Connections in *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

You can close the database to reject any new user connections to a database. The instance administrator can still connect to the database.

- TimesTen Classic: Use the `ttAdmin -close` command to close the database to any new user connections.

  Since the database can be automatically loaded or unloaded as set by the RAM policy, the status of a database (open or closed) is not aligned with whether the database is currently loaded into memory. Thus, the database could be in an open state as well as unloaded or in a closed state when loaded. See Loading a Database into Memory for TimesTen Classic for an example of the RAM policy.

  You should close a database before manually unloading a database. When loading a closed database into memory, the database cannot be opened until the load operation completes. You can re-open the database with the `ttAdmin -open` command. See Unloading a Database from Memory for TimesTen Classic for an example.
- TimesTen Scaleout: Before unloading a database from memory, you must manually close the database. See Unloading a Database from Memory in *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

Use the following to see the status of your database:

- TimesTen Classic: Use the `ttStatus` or `ttAdmin -query` utilities to see the status of the database. See ttStatus or ttAdmin in *Oracle TimesTen In-Memory Database Reference*.
- TimesTen Scaleout: Use the `ttGridAdmin dbStatus` command to see the status of the database. See Monitor the Status of a Database (dbStatus) in *Oracle TimesTen In-Memory Database Reference*.

Specifying the Type of Memory for Shared Memory Segment

In TimesTen Classic, the entire database resides in a single shared memory segment.

The behavior for the shared memory segment depends on:

- SysV shared memory segment: You need to configure the size of the shared memory segment to include the entire database. See Configure shmmax and shmall in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.
- RAM policy setting: The RAM policy setting designates how to load or unload the database (from the checkpoint files) for all shared memory types. When you unload the database, it destroys the shared memory segment. When you load the database, it creates a new shared memory segment and loads from the checkpoint files. If you specify the RAM policy of enduring, you can alternatively attach, detach, and free the shared memory segment. When you detach, the shared memory segment still exists and can be reused when you attach to it. See Detaching, Attaching, and Freeing the Shared Memory Segment.

Loading and Unloading the Database from Memory

TimesTen is an in-memory database. As such, a database must first be loaded into memory from the file system to be available for connections.

When a database is loaded into memory, the contents of the permanent memory region are read from checkpoint files stored on the file system. The temporary memory region is created when a database is loaded into memory and is destroyed when it is unloaded. See Specifying the Memory Region Sizes of a Database for more details on permanent and temporary memory.

- In TimesTen Scaleout: The grid administrator controls how to load and unload the database using the `ttGridAdmin` utility. See ttGridAdmin in the *Oracle TimesTen In-Memory Database Reference* for details.
- In TimesTen Classic: RAM policies specify how and when a database is loaded into memory, including whether to automatically reload the database into memory if the database is unloaded unexpectedly. See Specifying a RAM Policy for full details on the different RAM policies.

  Only the instance administrator can load a database manually. By default, TimesTen automatically loads an idle database (which is a database with no connections) into memory when a first connection is made to it. See Loading a Database into Memory for TimesTen Classic.

  After a database loads into memory, you may need to explicitly start the cache and replication agents for the database, depending on the functionality you are using and on which cache and replication policies you set with the `ttAdmin` utility.

> **Note:**
>
> Instead of loading and unloading the database, you can detach or attach the shared memory segment leaving the shared memory segment in memory. This is significantly faster than loading and unloading a database. See Detaching, Attaching, and Freeing the Shared Memory Segment.

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

- Loading a Database into Memory for TimesTen Classic
- Unloading a Database from Memory for TimesTen Classic

Loading a Database into Memory for TimesTen Classic

Loading the database into memory for TimesTen Classic can be done automatically when a certain ram policy is set or manually with the `ttAdmin` utility.

1. Before you try to load the database into memory for TimesTen Classic, confirm that the TimesTen daemon is running with the `ttStatus` utility. The following output shows that the TimesTen daemon is not running.

   ```
   % ttStatus
   ttStatus: Could not connect to the TimesTen daemon.
   If the TimesTen daemon is not running, please start it by running "ttDaemonAdmin -
   start".
   ```

2. Start the TimesTen daemon, if necessary.

   ```
   ttDaemonAdmin -start
   ```

3. The RAM policy setting is important as it specifies if, how, and when the database is loaded or unloaded from memory. The default RAM policy for a TimesTen database is `inUse`.

   You can set the RAM policy before loading the database into memory. See Specifying a RAM Policy.

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

   ```
   % ttAdmin -ramPolicy manual database1

   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
   Database state                  : open
   ```

4. Use the `ttAdmin` utility to load (or reload) the database into memory, or unload the database from memory.

   If the RAM policy is `manual` for the database `database1`, 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` or `enduring` RAM policies:

   ```
   % ttAdmin -ramLoad database1
   ```

```
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
Database state                  : open
```

If the RAM policy is `manual`, you can change it to `always` to specify that the database is always reloaded.

```
ttAdmin -ramPolicy always database1
```

If the RAM policy is `inUse`, then you want the grace period to be greater than 0, so that the database will be kept in memory for that time period when idle:

```
% ttAdmin -ramPolicy inUse -ramGrace 200 database1

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
Database state                  : open
```

If the RAM policy is `manual` for the database `database1` and the database was previously closed to incoming connections, then you can both load and open the TimesTen database into memory with the `ttAdmin -ramload -open` command.

```
% ttAdmin -ramLoad -open database1

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
Database state                  : open
```

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

   To start the replication agent:

   ```
   ttAdmin -repStart database1
   ```

   To start the cache agent:

   ```
   ttAdmin -cacheStart database1
   ```

See ttAdmin and ttDaemonAdmin in the *Oracle TimesTen In-Memory Database Reference*.

Unloading a Database from Memory for TimesTen Classic

In TimesTen Classic, a database remains loaded in shared memory if any applications or TimesTen agents, such as the cache agent or replication agent, are connected to it. In

TimesTen Classic, a database may also be kept in shared memory for particular RAM policy setting, even when no applications or agents are connected.

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

> **Note:**
>
> The following steps use examples where `database1` is the database that is to be unloaded. It is assumed that it is the active master in a replication scheme and has been configured with cache. Note that a database can have both replication and cache configured, and a RAM policy other than manual.

1. Close the database to reject any new requests to connect to the database.

   ```
   ttAdmin -close database1
   ```

2. Disconnect all applications from the database.

   To close all active connections to the database, run the `ttAdmin -disconnect` command. See Disconnecting from a Database in this book and ttAdmin in the *Oracle TimesTen In-Memory Database Reference*.

3. If the replication agent is running on the database, set the replication state to `pause` and stop the replication agent. The example sets the replication state from the active master `database1` to the standby master `standbydb` to `pause`, then stops the replication agent on the active master `database1`.

   ```
   ttRepAdmin -receiver -name database1 -state pause standbydb
   ttAdmin -repStop database1
   ```

4. If the cache agent is running on the database, stop the cache agent.

   ```
   ttAdmin -cacheStop database1
   ```

5. Ensure that the RAM policy is set to either `manual` or `inUse`. Then unload the database from memory. See Specifying a RAM Policy.

   If the RAM policy is set to `always`, change it to `manual` and then unload the database from memory with the `ttAdmin -ramPolicy -ramUnload` utility options.

   ```
   ttAdmin -ramPolicy manual -ramUnload database1
   ```

   If the RAM policy is set to `manual`, unload the database with the `ttAdmin -ramUnload` utility:

   ```
   ttAdmin -ramUnload database1
   ```

   If the RAM policy is set to `inUse` and a grace period is set, set the grace period to 0 or wait for the grace period to elapse. This results in the database being unloaded. TimesTen unloads a database with an `inUse` RAM policy from memory once you close all active connections.

   ```
   ttAdmin -ramGrace 0 database1
   ```

6. Run the `ttStatus` utility to verify that the database has been unloaded from memory and the database is closed. The database is unloaded if there are no processes. The database is closed when the output shows `Closed for user connections`.

See ttStatus in *Oracle TimesTen In-Memory Database Reference*.

7. Optionally, stop the TimesTen daemon.

```
ttDaemonAdmin -stop
```

See ttAdmin in the *Oracle TimesTen In-Memory Database Reference*.

Detaching, Attaching, and Freeing the Shared Memory Segment

With the enduring RAM policy, you can detach from the shared memory segment leaving the shared memory segment in memory. And when you are ready to open the database, you can attach to this same shared memory segment.

> **Note:**
>
> This should only be used when performing a fast upgrade. See About Performing a Fast Patch Upgrade in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.

Attaching and detaching a shared memory segment is significantly faster than loading and unloading a database. When you unload the database, it destroys the shared memory segment. When you load the database, it creates a new shared memory segment and loads from the checkpoint files. If you specify the RAM policy of enduring, you can attach, detach, and free the shared memory segment. When you detach, the shared memory segment still exists and can be reused when you attach to it.

Use the following ttAdmin commands to attach to, detach from, and free the shared memory segment. You can only use these features if the RAM policy is set to enduring.

- `ttAdmin -shmDetach [-ckpt | -noCkpt]`: Detaches the shared memory segment and stops the managing subdaemon. The shared memory segment remains in memory after the subdaemon exits.

  Prior to disconnecting from the shared memory segment, you can specify whether the subdaemon performs a static checkpoint or not. By default, a final checkpoint is performed with the `-ckpt` option. This final checkpoint ensures that a database start will not need to go through a recovery process. If you specify the `-noCkpt` option, then you avoid the final checkpoint and increase the performance of the detach and attach processes. However, if the database cannot attach to the previously detached shared memory segment and needs to be loaded from the checkpoint files, then not performing this checkpoint necessitates going through the recovery process during the load to avoid data loss.

- `ttAdmin -shmAttach`: Attaches to an existing shared memory segment that was successfully detached with a `ttAdmin -shmDetach` operation. Creates a subdaemon to manage this shared memory segment for the database.

- `ttAdmin -shmFree`: Explicitly destroys a shared memory segment that remains in memory. You can free a shared memory segment after a successful detach, after an unsuccessful detach, or after a database crash. If an unsuccessful detach operation or a database crash occurs, the shared memory segment needs to be freed before you can recover the database. To reload and recover the database, run the `ttAdmin -ramLoad` command, which uses the checkpoint files to recover.

The following examples demonstrates how to use `ttAdmin -shmDetach` to detach a shared memory segment so that it can remain in memory. Then, uses the `ttAdmin -shmAttach` command to attach to the existing shared memory segment.

1. Close database1 before detaching the shared memory segment with the `ttAdmin -shmDetach` command:

```
$ ttAdmin -close database1
RAM Residence Policy            : enduring
Manually Loaded In RAM          : True
Replication Agent Policy        : manual
Replication Manually Started    : False
Cache Agent Policy              : manual
Cache Agent Manually Started    : False
Database State                  : Closed
```

2. Detach the shared memory segment and shutdown the managing subdaemon for database1. You can see that the shared memory segment is detached as the "Manually loaded in RAM" output displays as False. The `ttStatus` utility states that the subdaemon is detached from the database.

```
$ ttAdmin -shmDetach database1
RAM Residence Policy            : enduring
Manually Loaded In RAM          : False
Replication Agent Policy        : manual
Replication Manually Started    : False
Cache Agent Policy              : manual
Cache Agent Manually Started    : False
Database State                  : Closed

$ ttStatus
TimesTen status report as of Wed Aug 18 01:45:55 2021

Daemon pid 27561 port 6626 instance instance1
TimesTen server pid 27568 started on port 6628
------------------------------------------------------------------------
------------------------------------------------------------------------
Data store /tmp/databases/database1
Daemon pid 27561 port 6626 instance instance1
TimesTen server pid 27568 started on port 6628
There are no connections to the data store
…
Closed to user connections
RAM residence policy: Enduring
Subdaemon is manually detached from data store (Shared Memory Key 0x07108a02 ID
745504801)
Replication policy  : Manual
Cache Agent policy  : Manual
PL/SQL enabled.
------------------------------------------------------------------------
Accessible by group g900
End of report
```

3. When ready, run the `ttAdmin -shmAttach` command to create the managing subdaemon, start the managing subdaemon for the database1 database and attach to the shared memory segment. The `Manually loaded in RAM` output displays as True to denote that the shared segment is reattached. The `ttStatus` utility shows that the database is loaded into RAM.

```
$ ttAdmin -shmAttach database1
RAM Residence Policy            : enduring
Manually Loaded In RAM          : True
```

```
Replication Agent Policy        : manual
Replication Manually Started    : False
Cache Agent Policy              : manual
Cache Agent Manually Started    : False
Database State                  : Closed

$ ttStatus
TimesTen status report as of Wed Aug 18 01:46:08 2021

Daemon pid 27561 port 6626 instance instance1
TimesTen server pid 27568 started on port 6628
------------------------------------------------------------------------
------------------------------------------------------------------------
Data store /tmp/databases/ database1
Daemon pid 27561 port 6626 instance instance1
TimesTen server pid 27568 started on port 6628
There are 12 connections to the data store
…
Closed to user connections
RAM residence policy: Enduring
Data store is manually loaded into RAM
Replication policy  : Manual
Cache Agent policy  : Manual
PL/SQL enabled.
------------------------------------------------------------------------
Accessible by group g900
End of report
```

4. Open the database.

```
$ ttAdmin -open sampledb
RAM Residence Policy            : enduring
Manually Loaded In RAM          : True
Replication Agent Policy        : manual
Replication Manually Started    : False
Cache Agent Policy              : manual
Cache Agent Manually Started    : False
Database State                  : Open
```

> **Note:**
>
> Being able to detach and attach to the shared memory segment provides a method for a faster upgrade. See Upgrades in TimesTen Classic in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.

If there is a failure (such as a subdaemon crash or an assertion failure) that causes an abrupt shutdown of the database, then the shared memory segment may not be transactionally consistent and needs to be destroyed before you can reload the database and create a new shared memory segment.

The following example frees the shared memory segment with the `ttAdmin -shmFree` command. After which, the example executes the `ttAdmin -ramLoad` to recover the database from the checkpoint files and creates a new shared memory segment.

1. Free the shared memory segment after a database crash.

```
$ ttAdmin -shmFree database1
RAM Residence Policy            : enduring
Manually Loaded In RAM          : False
```

```
Replication Agent Policy        : manual
Replication Manually Started    : False
Cache Agent Policy              : manual
Cache Agent Manually Started    : False
Database State                  : Closed

$ ttStatus
TimesTen status report as of Wed Aug 18 02:12:27 2021

Daemon pid 6168 port 6626 instance instance1
TimesTen server pid 6175 started on port 6628
------------------------------------------------------------------------
------------------------------------------------------------------------
Data store /tmp/databases/database1
Daemon pid 27561 port 6626 instance instance1
TimesTen server pid 27568 started on port 6628
There are no connections to the data store
Closed to user connections
RAM residence policy: Enduring
Data store is manually unloaded from RAM
Replication policy  : Manual
Cache Agent policy  : Manual
PL/SQL enabled.
------------------------------------------------------------------------
Accessible by group g900
End of report
```

2. Load and open the `database1` database with a new shared memory segment that recovers from the checkpoint files:

```
$ ttAdmin -ramLoad -open database1
RAM Residence Policy            : enduring
Manually Loaded In RAM          : True
Replication Agent Policy        : manual
Replication Manually Started    : False
Cache Agent Policy              : manual
Cache Agent Manually Started    : False
Database State                  : Open
```

Specifying a RAM Policy

TimesTen Classic enables you to specify a RAM policy that determines when TimesTen Classic databases are loaded and unloaded from main memory. There is also a RAM policy that enables you to detach and leave the shared memory segment in memory instead of unloading it.

For each TimesTen Classic database, you can have a different RAM policy.

> **Note:**
>
> TimesTen Scaleout supports the manually loading and unloading of the database through the `ttGridAdmin` utility by system administrators.

The RAM policy options are as follows:

- *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 (using `ttAdmin -ramUnload`) or unless an unrecoverable error condition occurs. The database can only be explicitly loaded into system RAM by the administrator (using the `ttAdmin -ramLoad` option). This is the recommended RAM policy, because it avoids unnecessary database loading or unloading.

See Perform RAM Operations in *Oracle TimesTen In-Memory Database Reference* for more details on `ttAdmin -ramLoad` and `-ramUnload` options. For more details on database error recovery, see Changes to RAM Policy After Automatic Recovery Fails.

- *enduring*: This option should be used when you want to perform a fast upgrade.

  The database can be manually loaded and unloaded by system administrators just as you would with the manual RAM policy.

  In addition, the enduring RAM policy enables you to detach from the shared memory segment before closing the database. And when you are ready to open the database, you can attach to this same shared memory segment and start up the main subdaemon to restart the database. In addition, if a database fails because of an error, you can free the shared memory segment and recover the database from the checkpoint files. See Detaching, Attaching, and Freeing the Shared Memory Segment.

  This is useful when performing an upgrade.

  See Perform RAM Operations in *Oracle TimesTen In-Memory Database Reference* for more details on `ttAdmin -ramLoad` and `-ramUnload` options. See Changes to RAM Policy After Automatic Recovery Fails for more details on database error recovery.

- *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.

  > **Note:**
  >
  > Set the RAM policy to *inUse* or *inUse with RamGrace* if the application requires that the TimesTen database is automatically loaded with the first connection and automatically unloaded with the last disconnection. However, setting RAM policy to *inUse* for production systems with large databases may cause performance issues having the database unload and reload unexpectedly.

- *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.

  The *always* RAM policy should be used with caution. When failures occur, it may not be beneficial to have your database automatically reload. In addition, it may affect system startup performance if all databases load at the same time when your system boots. See Changes to RAM Policy After Automatic Recovery Fails on error recovery and Preventing an Automatic Reload of the Database After Failure for what could occur when trying to reload the database.

A system administrator can set the RAM policy or manually load or unload a database in TimesTen Classic with either the `ttAdmin` utility or the C API RAM policy utilities. See ttAdmin in *Oracle TimesTen In-Memory Database Reference* or the TimesTen Utility API chapter in *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 Classic 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.

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

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

```
% ttAdmin -ramPolicy manual ttdata
RAM Residence Policy            : manual
Replication Agent Policy        : manual
Replication Manually Started    : False
Cache Agent Policy              : manual
Cache Agent Manually Started    : False
Database state                  : open
```

The following example sets the RAM policy of a TimesTen database to enduring:

```
$ ttAdmin -ramPolicy enduring database1

RAM Residence Policy            : enduring
Manually Loaded In RAM          : False
Replication Agent Policy        : manual
Replication Manually Started    : False
Cache Agent Policy              : manual
Cache Agent Manually Started    : False
Database state                  : open
```

Changes to RAM Policy After Automatic Recovery Fails

Certain procedures automatically occur by default when a fatal error invalidates the database and the automatic database recovery performed by TimesTen Classic is unsuccessful.

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

Preventing an Automatic Reload of the Database After Failure

After a fatal error that causes the database to be invalidated, TimesTen Classic 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 TimesTen Classic in the *Oracle TimesTen In-Memory Database Cache 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*.

> **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*.

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

- The TimesTen daemon restarts.

- A process connects successfully.
- The administrator runs 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"
```

Disconnecting from a Database

You can shut down or unload the database by first disconnecting applications in an orderly fashion.

The forced disconnect option asynchronously disconnects all connected applications from the database, including those that are idle or unresponsive.

- Reliably disconnects and detaches from the shared memory segment for a database.
- Successfully disconnects any idle or unresponsive connections.

The following sections describe how to disconnect connections from a TimesTen database:

- Disconnecting from a Database in TimesTen Scaleout
- Disconnecting from a Database in TimesTen Classic

Disconnecting from a Database in TimesTen Scaleout

If you are unable to individually disconnect every application from the TimesTen Scaleout database, use the `ttGridAdmin dbDisconnect` command to disconnect all user connections from the database.

See Unloading a Database from Memory in *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

Disconnecting from a Database in TimesTen Classic

You can disconnect all connections to a TimesTen Classic database with the `ttAdmin -disconnect` command.

However, you must first enable the capability for forced disconnect by setting the `ForceDisconnectEnabled` connection attribute to 1 in the DSN definition within the `sys.odbc.ini` file. See ForceDisconnectEnabled in the *Oracle TimesTen In-Memory Database Reference*.

While control returns to the command prompt, the force disconnect operation may take multiple seconds (or minutes) to complete. Verify the status of the force disconnect operation with the `ttStatus` utility.

While the forced disconnect operation is in process, any new connection request is rejected by the main daemon. Once the force disconnect operation completes, new connections are accepted.

You can specify how urgently you need connections to be forced to disconnect with the urgency level:

- The `-transactional` option waits for any open transactions to be committed or rolled back before disconnecting. Does not affect idle connections.

- The `-immediate` option rolls back any open transactions before immediately disconnecting. This option also disconnects idle connections.
- The `-abort` option terminates all direct mode application processes and client/server processes (`ttcserver`) in order to disconnect.

Most of the time, you should use the `-transactional` and `-immediate` options. A recommended practice is to run the `-disconnect` command twice, as necessary. First use the transactional urgency level. Then, after allowing some time, use `ttStatus` to confirm whether connections have been closed. If not all connections have been closed yet, then use the immediate urgency level.

The `-abort` option should only be used on the rare occasion when both the transactional and immediate urgency levels fail to successfully disconnect all specified connections. The `-abort` option could result in lost transactions, as this operation abruptly causes every user and `ttcserver` process connected to the database to terminate.

You can specify which type of connections to disconnect with the granularity level.

- The `-users` option (default) disconnects every user connection to the database. For example, use this granularity level when preparing to perform database maintenance.
- The `-unload` option disconnects every connection to the database, including subdaemon connections. For example, use this granularity level when attempting to unload the database.

> **Note:**
>
> The `always` RAM policy conflicts with the `unload` granularity level. Using these simultaneously returns an error.

For example, the following script disconnects all connections and unloads the TimesTen Classic database by first running `ttAdmin -disconnect` with the transactional urgency level. Then, the script waits a short time to evaluate if the connections disconnected before trying the immediate urgency level.

```
#!/bin/sh

# disconnect users and unload the database with the transactional urgency level
ttAdmin -disconnect -transactional -unload database1

# wait 10 seconds for the forced disconnect to finish
COUNT = 0
while [ ttStatus | grep "pending disconnection" ] || [ $COUNT -ne 10 ]
do
  sleep 1
  COUNT=$((COUNT+1))
done

# increase the urgency level to immediate
if [ ttStatus | grep "pending disconnection" ]; then
  ttAdmin -disconnect -immediate -unload database1
fi
```

Use the `ttStatus` utility to check progress. During a forced disconnect operation, the output indicates the pending disconnections:

```
TimesTen status report

Daemon pid 10457 port 6627 instance user1
TimesTen server pid 10464 started on port 6629
------------------------------------------------------------------------
------------------------------------------------------------------------
Data store /disk1/databases/database1
Daemon pid 10457 port 6627 instance user1
TimesTen server pid 10464 started on port 6629
There are 14 connections to the data store, ***14 pending disconnection***
Shared Memory KEY 0x0210679b ID 949092358
PL/SQL Memory KEY 0x0310679b ID 949125127 Address 0x5000000000
Type            PID     Context             Connection Name              ConnID
Process         10484   0x00007f3ddfeb4010  tt_181                            1
...
```

Specifying the Memory Region Sizes 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 the file system. The permanent memory region is written to the file system during checkpoint operations. TimesTen stores all data in RAM to achieve exceptional performance. The database throws an error if there is no space left for a new piece of data. `PermSize` can be increased with a database restart but it cannot be decreased.
- 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 PermSize and TempSize in the *Oracle TimesTen In-Memory Database Reference*.

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. See Loading and Unloading the Database from Memory.

Managing the database size is described in these sections:

- Estimating and Modifying the Memory Region Sizes for the Database
- Monitoring PermSize and TempSize Attributes
- Reducing Database Size for TimesTen Classic

- Receiving Out-of-Memory Warnings

Estimating and Modifying the Memory Region Sizes for the Database

Database operations cannot complete successfully without allocation of sufficient memory. First, determine appropriate sizes for the TimesTen permanent and temporary memory regions and the transaction log buffer.

Use the `ttShmSize` utility or run the application until you can make reasonable estimates, then set these TimesTen connection attributes:

- `DSN`: It is specified 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, `[Sampledb]`.
- `PermSize`: Size of the permanent memory region, in MB, for the database where the actual data is stored. Make sure `PermSize` is sufficient to hold all the data. You can increase this value, but not decrease it for this database.

  In TimesTen Classic, you can decrease the permanent memory region by re-creating the database with a smaller size. See Reducing Database Size for TimesTen Classic.

  > **Note:**
  >
  > See Determining the Value of the PermSize Attribute in the *Oracle TimesTen In-Memory Database Scaleout User's Guide* for how to evaluate an appropriate value for the `PermSize` connection attribute for a TimesTen Scaleout database.

- `TempSize`: For TimesTen Classic, `TempSize` indicates the total amount of memory in MB allocated to the temporary region for the database. For TimesTen Scaleout, `TempSize` indicates the total amount of memory in MB allocated to the temporary region for an element. Related database operations may fail if `TempSize` is insufficient. You can change this size with a database restart.
- `LogBufMB`: Size of the internal transaction log buffer, in MB. See Configure Log Buffer and Log File Size Parameters.
- `Connections`: Maximum number of connections that you expect your database will use.

See PermSize, TempSize, LogBufMB and LogFileSize in the *Oracle TimesTen In-Memory Database Reference*.

Next, ensure that the maximum shared memory segment size of your system is large enough to contain the database. Use the maximum number of connections that you expect your database to use. All of the values are in MB (megabytes). Make it larger than the following:

```
$ ttShmSize -connstr
"DSN=sampledb;PermSize=512;TempSize=128;LogBufMB=256;Connections=2048"
The required shared memory size is 1178028616 bytes.
```

> **Note:**
>
> If a TimesTen Classic 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.

If there are multiple TimesTen databases on your system, each using its own shared memory segment, the maximum shared memory segment size must be large enough to accommodate the largest database.

Next, determine the total shared memory allocation you will need (converted to appropriate units). If there are multiple TimesTen databases on your system, the total shared memory allocation must be large enough to accommodate all of them, using the above equation for each database. (Then, on Linux for example, divide this value by the page size, typically 4096 bytes, to get total memory allocation in pages.)

> **Note:**
>
> In TimesTen Classic, additional shared segments may be created either for PL/SQL with the `PLSQL_MEMORY_SIZE` connection attribute or for client/server with the `-server_shmsize` configuration option (in `timesten.conf`). Refer to PLSQL_MEMORY_SIZE in *Oracle TimesTen In-Memory Database Reference*. If you use the default values or similarly small sizes, there should be enough unused space in shared memory to accommodate these segments.

Finally, if you want to allow for database invalidation, there must be at least twice as much physical memory as the size of the largest TimesTen database. If you do not allow for this, but an invalidation does occur, you cannot reload the database into memory until all processes and connections that used the database have been found and terminated.

See Linux Prerequisites in *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.

Monitoring PermSize and TempSize Attributes

The `SYS.V$MONITOR` and `SYS.GV$MONITOR` system views 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.

For example, you could evaluate the temporary space usage by running a full workload and watching the high water mark (`TEMP_IN_USE_HIGH_WATER`) of the temporary space usage. The high water mark can be reset using the `ttMonitorHighWaterReset` built-in procedure. And if necessary, you can change `TempSize` to a value of the observed `TEMP_IN_USE_HIGH_WATER` value and add 10%.

> **Note:**
>
> You can also use the ttIsql `dssize` command to provide this information. See Using the ttIsql dssize Command and ttIsql in the *Oracle TimesTen In-Memory Database Reference*.

You can monitor block-level fragmentation in the database with the `SYS.V$BLOCK_INFO` or `SYS.GV$BLOCK_INFO` system tables or by calling the `ttBlockInfo` built-in procedure.

See SYS.GV$MONITOR, SYS.V$MONITOR, SYS.GV$BLOCK_INFO or SYS.V$BLOCK_INFO in the *Oracle TimesTen In-Memory Database System Tables and Views Reference* for details on these views. See ttBlockInfo in the *Oracle TimesTen In-Memory Database Reference*.

Reducing Database Size for TimesTen Classic

Once a TimesTen Classic database has been defined with a particular size for the permanent region (indicated by the `PermSize` DSN attribute), it cannot be reduced to a smaller size, even if tables or rows are deleted.

To reduce the allocated size of the permanent region of a TimesTen Classic database, run the `ttMigrate` utility to save a copy of the database and then re-create the database with a smaller permanent region size and restore the data.

Perform these steps to reduce the permanent region size of a TimesTen Classic database:

1. Disconnect applications from the database with the `ttAdmin -disconnect` command. See Disconnecting from a Database in TimesTen Classic in this book and ttAdmin in the *Oracle TimesTen In-Memory Database Reference.*
2. Use the `ttMigrate -c` option to create a data file for the database.

   ```
   ttMigrate -c database1 /tmp/database1
   ```

3. Unload the TimesTen Classic database from memory. See Unloading a Database from Memory for TimesTen Classic.
4. Create a new DSN definition for the new copy of the database with a smaller `PermSize` value. To modify the original DSN rather than create a new one, you must destroy the original TimesTen Classic database using the `ttDestroy` utility before restoring from the backup.
5. Recreate the TimesTen Classic database by using ttIsql with `AutoCreate=1`.

   ```
   ttIsql -connstr "dsn=database1;AutoCreate=1" -e "quit"
   ```

   The database is empty at this point.
6. Restore the backup by using the `ttMigrate -r` and `-relaxedUpgrade` options.

   ```
   ttMigrate -r -relaxedUpgrade database1 /tmp/database1
   ```

> **Note:**
>
> - The permanent region size of a TimesTen Classic database cannot be reduced below the size that is required by the data currently stored in the database. This value can be determined by querying the `perm_in_use_size` column of the `v$monitor` system view.
> - You can also use this procedure to compact the TimesTen Classic database in order to reduce fragmentation caused by partially full table pages or fragmentation of the heap buffers that store index nodes and out-of-line values.

Receiving Out-of-Memory Warnings

Applications must call the `ttWarnOnLowMemory` built-in procedure to receive out-of memory warnings.

TimesTen Classic also provides two general connection attributes that determine when a low memory warning should be issued: `PermWarnThreshold` and `TempWarnThreshold`. Both attributes take a percentage value.

Storage Provisioning for TimesTen

Storage provisioning is the process of allocating server storage space.

- TimesTen installation storage allocation: A TimesTen installation includes all of the software.

  Plan to allocate at least 1.5 GB for each TimesTen installation. If you run multiple instances on the same system, they can all share a single TimesTen installation. In order to incorporate software version upgrades, you should provision double that amount of space, plus some additional space.

- TimesTen instance storage allocation: Each TimesTen instance runs its own set of daemon processes (with associated daemon log files).

  Plan to allocate at least 256 MB for each TimesTen instance.

- Checkpoint files: Each TimesTen database requires disk space for two checkpoint files, each of which is stored in the directory that is specified with the `DataStore` attribute. As each checkpoint file grows on the file system, 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. Thus, you should plan as though each checkpoint file occupies approximately the same amount of space as defined for permanent memory (as defined with the `PermSize` connection attribute).

  Plan to allocate twice the total permanent space of your databases plus 30%.

  > **Note:**
  >
  > For TimesTen Classic, you can set the `Preallocate` connection attribute to `1` to have TimesTen reserve file system space at connect time for checkpoint files. This is useful for large databases, which ensures that the file system always has room for the checkpoint files as data is added to the database. For more information, see Preallocate in the *Oracle TimesTen In-Memory Database Reference*.

- Transaction log files: Transaction log files are the most difficult space requirement to estimate. See Storage Provisioning for Transaction Log Files for how to plan for space provisioning for transaction log files.

- Local space to facilitate backup or export files for repositories in TimesTen Scaleout.

  When you use an SCP-attached repository, any operations are first saved locally by the target instances before copied to the repository. The target data instances initially store the backup (or export) files to the $*timesten_home*`/grid/admin/temp` directory. You must ensure that you provision enough storage in this directory to accommodate the maximum size for backup or export files that would end up in a repository.

Plan to allocate space of `PermSize` times the number of replica sets.

- Repository storage provisioning in TimesTen Scaleout.

  A single backup file contains one element from each replica set. Thus, the number of backup files needed to contain all of the rows of the tables in the database corresponds to the number of replica sets in your database.

  > **Note:**
  >
  > For hash distributed tables, only a portion of the rows of the tables are included in each backup file. The rows of hash distributed tables are distributed across the number of replica sets.
  >
  > For duplicated tables, each backup file contains all rows in each table.

  Plan to allocate space of `PermSize` times the number of replica sets.

Storage Provisioning for Transaction Log Files

There is a process for estimating the necessary file system space allocation for transaction log files

> **Note:**
>
> You can generate a more accurate estimate of the transaction log volume within a test environment.

Consider the following when estimating the necessary file system space allocation for transaction log files:

- TimesTen keeps enough transaction log files to support recovering from either checkpoint file. Transaction log files may accumulate depending on the settings for background checkpoint operations (defined with the `CkptLogVolume` and `CkptFrequency` first connection attributes). In addition, TimesTen preallocates space for three additional transaction log files.
- TimesTen uses fuzzy checkpoint operations for background operations. Transaction log files can accumulate during fuzzy checkpoint operations. The amount of time taken to create a checkpoint scales with the value assigned to the `PermSize` connection attribute.
- Transaction log files may accumulate between incremental backups.
- In TimesTen Scaleout, each element in each replica set has its own transaction log files to store transaction log records. If a transaction modifies data in a replica set, then the changes are usually recorded in both elements of the replica set. However, if one of the elements in the replica set is down, then the changes are only recorded in the transaction log files for the active element. Transaction log files can accumulate if one element in a replica set is down for a very long time.
- In TimesTen Classic, transaction log files may accumulate on the master during temporary replication outages or if the subscriber is down.

> **Note:**
>
> You can limit the number of transaction log files retained with the `FAILTHRESHOLD` clause on the `CREATE REPLICATION` and `ALTER REPLICATION` statements.

See CkptLogVolume, CkptFrequency, and PermSize in the *Oracle TimesTen In-Memory Database Reference* for more information on these connection attributes. See CREATE REPLICATION and ALTER REPLICATION in the *Oracle TimesTen In-Memory Database SQL Reference* for more information on these SQL statements.

The estimate of the transaction log volume depends on the peak update transaction rate and the average complexity of each transaction that modifies the database.

Consider:

- **B** represents the transaction log volume in bytes per transaction.
- **C** represents the number of columns updated. The minimum transaction update (which is an update of a single number column) generates 400 bytes of transaction log data. Each additional number column update generates another 250 bytes.
- **V** represents the average size of larger data inserted or updated (columns of type `CHAR`, `VARCHAR2`, `BINARY`, `VARBINARY` or `LOB`) for each write transaction.

To estimate the transaction log volume for each transaction, use this formula:

B = 400 + ((**C-1**) * 250) + **V**

Note that the first column is just 150 bytes more than all subsequent columns, as follows:

B = (**C** * 250) + (400-250) + **V**

which simplifies to:

B =(**C** * 250) + 150 + **V**

Multiply the estimated transaction log volume value (**B**) by the expected peak transaction rate to find the expected peak transaction log rate.

Where:

- **L** = total storage space to provision for transaction log files in bytes.
- **C** = average number of columns inserted or updated per write transaction.
- **V** = average bytes of large data (`CHAR`, `VARCHAR2`, `BINARY`, or `VARBINARY` columns) inserted or updated for each write transaction.
- **S** = time in seconds of required transaction log retention.
- **T** = peak transaction rate per second averaged over intervals of **S** seconds.
- **f** = fraction of transactions that include insert, update, or delete operations.
- Lastly, increase the estimated file system space by an additional 30% for contingencies.

Taking all these factors into account, estimate provisioned transaction log file space in bytes according to this formula:

L = ((**C** * 250) + 150 + **V**) * **S** * **T** * **f** * 1.3

For example, a workload consists of 35% update transactions. Each transaction updates four columns, including two character columns each averaging 30 bytes updated total. The

workload runs at 1 million transactions per second and needs enough transaction log space to hold one hour's worth of transactions.

- **C** = 4 columns
- **V** = 30
- **S** = 3600
- **T** = 1,000,000
- **f** = 35% = 0.35

Thus, you would estimate that the storage provisioning workload requires:

L = ((4 * 250) + 150 + 30) * 3,600 * 1,000,000 * 0.35 * 1.3 = 1.9 TB

Thus, provision file system space of 1.9 TB for the transaction log files.

Bulk Copy Data Using the ttBulkCp Utility

The `ttBulkCp` utility enables you to copy data between TimesTen tables and ASCII files.

You can manage certain aspects of existing tables in the database with the `ttBulkCp` utility. With the `ttBulkCp` utility, you can add rows of data to an existing table, save data to an ASCII file, and 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`.

- Copying Data from a TimesTen Table to an ASCII File
- Copying Data from an ASCII File into a TimesTen Table

Copying Data from a TimesTen Table to an ASCII File

Run the `ttBulkCp` utility with the `-o` option to copy data from a TimesTen table to an ASCII file.

> **Note:**
>
> Ensure that your TimesTen user has `SELECT` privilege on the tables it copies information from.

This example copies the data from the `hr.employees` table of the `database1` database to the `employees.dmp` file using the `ttBulkCp -o` mode.

```
ttBulkCp -o -connstr "DSN=database1;UID=HR;PWD=hr" hr.employees > employees.dmp
```

See ttBulkCp in the *Oracle TimesTen In-Memory Database Reference*.

Copying Data from an ASCII File into a TimesTen Table

The `ttBulkCp` utility enables you to copy data from an ASCII file into a database table. The `ttBulkCp` utility does not copy duplicate rows into a table.

- Running ttBulkCp with the -i Option
- Running ttBulkCp with the -directLoad Option on TimesTen Classic

Running ttBulkCp with the -i Option

The `ttBulkCp` utility with the `-i` option enables you to load data from a file.

This option uses standard `INSERT` SQL statements to load data into a specific table of a TimesTen database.

On TimesTen Scaleout, the `ttBulkCp` utility inserts each row into its corresponding element based on the distribution scheme of the table. For TimesTen Scaleout, you can populate a table from a single location or from several locations. See Bulk Loading Data into a Database in *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

> **Note:**
>
> Ensure that your TimesTen user has `INSERT` privilege on the tables it copies information into.

This example copies the data from the `employees.dmp` file into the `hr.employees` table of the `database1` database using the `ttBulkCp -i` mode.

```
% ttBulkCp -i -connstr "DSN=database1;UID=HR;PWD=hr" hr.employees employees.dmp

employees.dmp:
    107 rows inserted
    0 duplicate rows not inserted
    107 rows total
```

See ttBulkCp in *Oracle TimesTen In-Memory Database Reference*.

Running ttBulkCp with the -directLoad Option on TimesTen Classic

Run the `ttBulkCp` utility with the `-directLoad` option to copy data from an ASCII file into a TimesTen Classic database table.

The `-directLoad` option loads data with standard `INSERT` SQL statements. The `ttBulkCp -directLoad` option can only be used by applications using direct connections, which avoids some of the overhead required when using client/server connections resulting in better performance than the `-i` option.

For improved performance, consider dropping indexes before loading data with the `-directLoad` option. Use the `ttSchema` utility to view the definition of all the indexes that are created on the tables of a TimesTen Classic database. Once the load operation is complete, manually re-create the indexes on your table. See ttSchema in the *Oracle TimesTen In-Memory Database Reference*.

> **Note:**
>
> Ensure that your TimesTen user has `INSERT` privilege on the tables it copies information into.

This example copies the data from the `employees.dmp` file into the `hr.employees` table of the `database1` database using the `ttBulkCp -directLoad` option.

```
% ttBulkCp -directLoad -connstr "DSN=database1;UID=HR;PWD=hr" hr.employees employees.dmp

employees.dmp:
    107 rows inserted
    0 duplicate rows not inserted
    107 rows total
```

See ttBulkCp in the *Oracle TimesTen In-Memory Database Reference*.

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 enables 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 enables 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 the different types of database fragmentation:

- Offline Defragmentation of a TimesTen Scaleout Database
- Offline Defragmentation of a TimesTen Classic Database
- Online Defragmentation of TimesTen Classic Databases in an Active Standby Pair Replication Scheme

- Online Defragmentation of TimesTen Classic Databases in a Non Active Standby Pair Replication Scheme

Offline Defragmentation of a TimesTen Scaleout Database

A TimesTen Scaleout database is defragmented as part of the export and import process.

Thus, to defragment a TimesTen Scaleout database, use the `ttGridAdmin dbExport` and `dbImport` commands. See Exporting and Importing a Database in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

Offline Defragmentation of a TimesTen Classic Database

To defragment a TimesTen Classic database, use the `ttMigrate` utility.

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:**
>
> You can achieve maximum table defragmentation with the `ttMigrate -r -relaxedUpgrade` command. The `-relaxedUpgrade` option also condenses table partitions. If you do not want to condense table partitions, remove the `-relaxedUpgrade` option from the `ttMigrate -r -relaxedUpgrade` 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*. See, Avoid ALTER TABLE for performance considerations.

See ttMigrate in the *Oracle TimesTen In-Memory Database Reference*.

Online Defragmentation of TimesTen Classic Databases in an Active Standby Pair Replication Scheme

Use a combination of the `ttMigrate -relaxedUpgrade` and `ttRepAdmin -duplicate` utilities to defragment TimesTen Classic 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:**
>
> If your TimesTen Classic database uses an active standby pair replication scheme, then you can only defragment these databases if the active standby pair replication scheme either does not contain any cache groups or contains only `READONLY` cache groups.

The following sections describe how to defragment TimesTen Classic 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 a 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`.

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

Migrate and Rebuild the Standby Database

This section 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, run `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.

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

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

   ```
   ttMigrate -c 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 in ttIsql, drop all cache groups:

```
DROP CACHE GROUP t_cg;
```

Destroy the standby database:

```
ttDestroy ttdb2
```

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

```
ttIsql ttdb2
```

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

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

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

```
ALTER CACHE GROUP t_cg SET AUTOREFRESH STATE PAUSED;
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: SALES.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 >
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 with the `ttAdmin -disconnect` command. Any query only processing can be moved to work at the `ttdb2` TimesTen database. See Disconnecting from a Database in this book and ttAdmin in the *Oracle TimesTen In-Memory Database Reference.*
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*.

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

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

   ```
   call ttRepStop;
   ```

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

   ```
   call ttRepDeactivate;
   ```

   Call the `ttRepStateGet` built-in procedure to show the database state.

   ```
   Command> call ttRepStateGet;
   < IDLE >
   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.

   ```
   call ttRepStateSet('ACTIVE');

   Command> call ttRepStateGet;
   < ACTIVE >
   1 row found.
   ```

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

   ```
   ttAdmin -repStop ttdb1
   ```

7. Run `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.

   ```
   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 a New Standby

You can destroy and recreate a new standby by duplicating the new active with the `ttRepAdmin -duplicate` command.

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 using ttIsql:

```
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 TimesTen Classic Databases in a Non Active Standby Pair Replication Scheme

Use a combination of the `ttMigrate -relaxedUpgrade` and `ttRepAdmin -duplicate` utilities to defragment TimesTen Classic 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:**
>
> 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*.

The following sections describe how to defragment TimesTen Classic 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 Classic 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 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 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 with the `ttAdmin -disconnect` command. Any processing can be moved to work at the `ttdb2` TimesTen database. For more information, see Disconnecting from a Database in this book and ttAdmin in the *Oracle TimesTen In-Memory Database Reference*.
2. Call the `ttRepSubscriberWait` built-in procedure at the 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.
>
> See ttRepSubscriberWait in the *Oracle TimesTen In-Memory Database Reference*.

Using ttIsql on `ttdb1`:

```
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:

   Using ttIsql on `ttdb1`:

   ```
   DROP REPLICATION r1;
   ```

   Using ttIsql on `ttdb2`:

   ```
   DROP REPLICATION r1;
   ```

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

   Using ttIsql on `ttdb1`:

   ```
   DROP REPLICATION r1;
   ```

   Using ttIsql on `ttdb2`:

   ```
   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 Classic database (`ttdb1`) by duplicating the previously defragmented TimesTen Classic 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 of the TimesTen Classic databases involved in either a unidirectional or bidirectional replication scheme with only a few seconds of service interruption.

3

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 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 Linux and UNIX

Overview of the TimesTen Client/Server

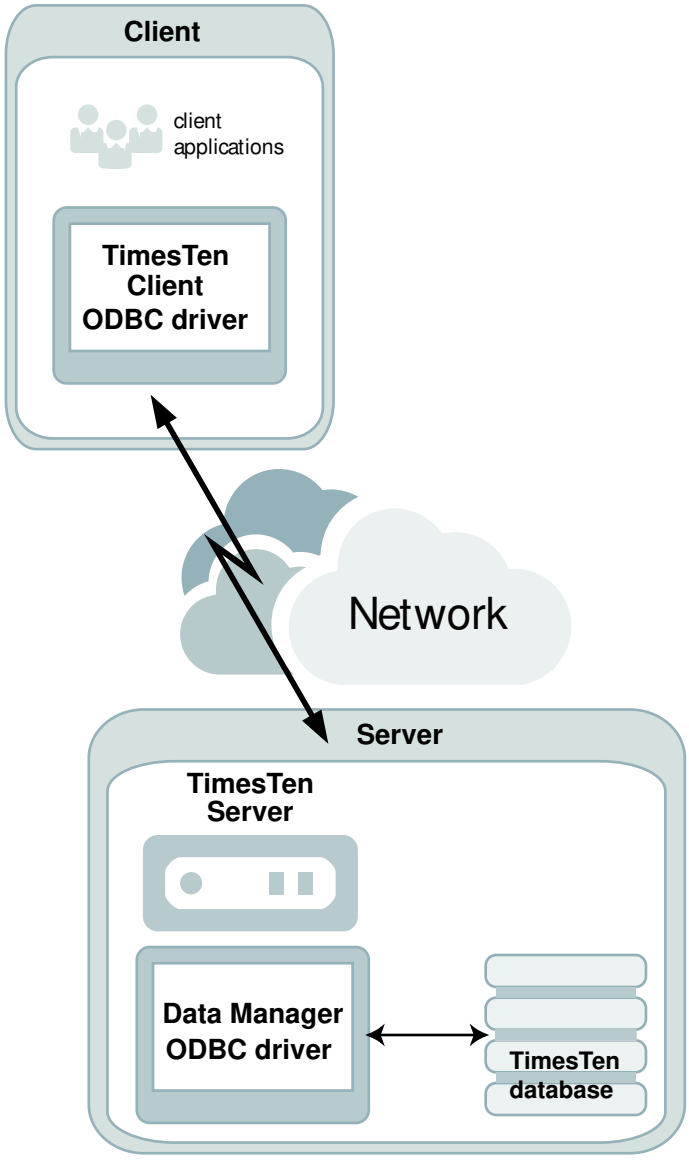
The TimesTen server is a process that runs on a server system that takes network requests from TimesTen clients and translates them into operations on databases on the server system. This enables clients to connect to databases that are located on different systems, potentially running a different platform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

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.

> **Note:**
>
> You can create a client/server connection between any combination of platforms that TimesTen supports.
>
> TimesTen does not support IP-level failover or load-balancing with client/server.

Figure 3-1 demonstrates how the TimesTen client and TimesTen server communicate using their respective drivers.

Figure 3-1 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 ODBC driver, applications can connect transparently to TimesTen databases on a remote or local system that has TimesTen installed.

  You can link a client application directly with the TimesTen client ODBC driver or with the Windows ODBC driver manager to access the TimesTen database. TimesTen supplies a driver manager for Linux, 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

- TimesTen server: On the server system, the TimesTen server uses the TimesTen Data Manager driver. The server's responsibility is to listen for requests from a client application, process the request through the TimesTen Data Manager 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 Setting the Classpath for Java Development 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

On Linux and UNIX, some TimesTen utilities only work over direct connections, such as `ttAdmin`, `ttRepAdmin`, and `ttBackup`.

The utilities that can be run over a client/server connection on the Linux and UNIX platforms are named with a suffix of `CS`, such as `ttIsqlCS`, `ttBulkCpCS`, `ttMigrateCS` and `ttSchemaCS`.

The following are the restrictions on client/server communication:

- 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 Utilities in the *Oracle TimesTen In-Memory Database Reference* provides the name of the client/server version of that utility, if there is one.
- XLA cannot be used over a client/server connection and is only supported within TimesTen Classic.
- 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. See Overview of TimesTen Users in the *Oracle TimesTen In-Memory Database Security Guide* for details on internal and external users.
- You cannot create or alter internal users over a client/server connection. Thus, you can only issue 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 a Database User and Privileges to Connect to the Database in the *Oracle TimesTen In-Memory Database Security Guide*.

- On Linux and 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

The following sections describe the communication protocols that the TimesTen client can use with the TimesTen server.

- TCP/IP Communication
- Shared Memory Segment Used for 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 Segment Used for Communication

Client/server connections use shared memory for communication.

A small shared memory segment is allocated (one for each client/server connection) for the duration of each client/server connection. This shared memory segment is created at connect time and is destroyed when the client/server connection is disconnected from the TimesTen database.

The `SHMMNI` value controls the number of shared memory segments that the host can create simultaneously. You must configure the `SHMMNI` parameter setting to account for the number of client/server connections. Set `SHMMNI` to a value that is greater than number of expected client/server connections. See Set the SHMMNI Parameter in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.

UNIX Domain Socket Communication

On Linux and 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 enables 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.

Configuring TimesTen Client and Server

You can connect an application to a TimesTen database using a client and server.

> **Note:**
>
> Before configuring the TimesTen client and server, see Connecting to TimesTen with ODBC and JDBC Drivers and Specifying Data Source Names to Identify TimesTen Databases.

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 of the Same TimesTen Release
- Installing and Configuring Cross-Release TimesTen Client/Server
- Defining Server DSNs for TimesTen Server on a Linux or UNIX System
- Defining a Logical Server Name
- Defining Client DSNs on a TimesTen Client System
- Sizing the Client Result Set Buffer
- Using Automatic Client Failover
- Configuring TCP Keep-Alive Parameters

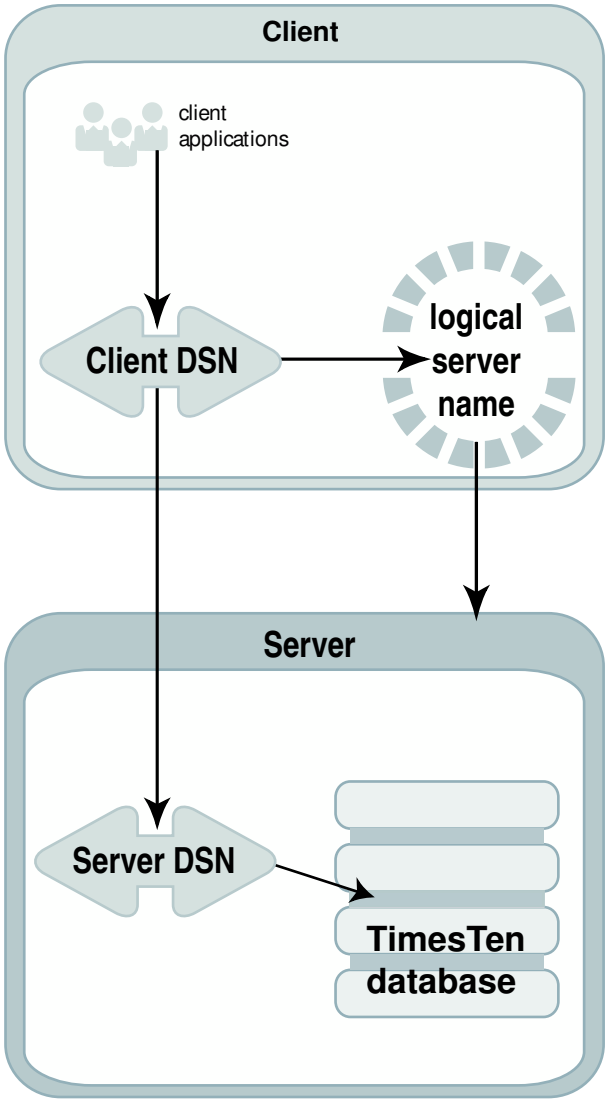
You can optionally configure and use TLS for encrypted communication between clients and the server. See Transport Layer Security for TimesTen Client/Server in *Oracle TimesTen In-Memory Database Security Guide*.

Overview of TimesTen Client/Server Configuration

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

See Figure 3-2.

Figure 3-2 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 odbc.ini file or the system `sys.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 `sys.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 of the Same TimesTen Release

There are methods for installing and configuring 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 Overview of the Installation Process in TimesTen Classic in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.

2. Create and configure a server DSN corresponding to the TimesTen database. See Defining Server DSNs for TimesTen Server on a Linux or UNIX System. 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

Install and configure the TimesTen client 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 Overview of the Installation Process in TimesTen Classic in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade 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*.

3. Create and configure a Client DSN corresponding to the Server DSN. See Creating and Configuring Client DSNs on Linux and UNIX and Creating and Configuring Client DSNs on Windows.

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*.

Installing and Configuring Cross-Release TimesTen Client/Server

Most TimesTen clients can connect to a TimesTen server from a different release.

A TimesTen client 11.2.2 or later release may connect to a TimesTen server 11.2.2 or later release. However, due to changes in ODBC 3.5 functionality, TimesTen clients of release 18.1 or later can no longer connect to older TimesTen servers in certain circumstances. For more information about client/server cross-release restrictions, see Client/Server Cross-Release Restrictions With ODBC 3.5 in the *Oracle TimesTen In-Memory Database C Developer's Guide*.

If you are configuring for a cross-release TimesTen client/server connection, install and configure as directed in Installing and Configuring for Client/Server of the Same TimesTen Release.

The TimesTen server loads a driver and a TimesTen database of its own release when a TimesTen client application connects to the TimesTen database. The TimesTen Data Manager driver is automatically installed on the TimesTen 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 Linux or UNIX. The client must be running on the same host as the server and thus must be on the same platform. The release level for the client and server hosts must be the same.

Defining Server DSNs for TimesTen Server on a Linux or UNIX System

Server DSNs identify TimesTen databases that are accessed by a client/server connection.

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 can be configured in the TimesTen daemon options file (`timesten.conf`). 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 Attributes. For a complete description of the TimesTen Server connection attributes, see Connection Attributes in the *Oracle TimesTen In-Memory Database Reference*.

A server DSN must be defined as a system DSN and has the same configuration format and attributes as a Data Manager DSN.

- For TimesTen Scaleout: A DSN is defined within the appropriate database definition and connectable. A connectable enables all data instances to accept connection requests from other data instances in the grid.

However, to establish a client connection from a TimesTen client that is not part of the grid, create a client DSN in the user `odbc.ini` file or the system `sys.odbc.ini` file with the `ttGridAdmin gridClientExport` command. This command exports every client/server connectable available for the grid into a file that is formatted to replace the `odbc.ini` file used by the TimesTen client. See Connect to a Database Using ODBC and JDBC Drivers in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

- With TimesTen Classic, you create each system DSN by creating or editing the user `odbc.ini` file or the system `sys.odbc.ini` file. You can add or configure a server DSN while the TimesTen server is running. See Creating a DSN on Linux and UNIX for TimesTen Classic.

By default, TimesTen creates only one connection to a server per child process. However, you can specify multiple client/server connections to a single TimesTen server with the connection attributes (described below) or by setting the TimesTen daemon attributes in the `timesten.conf` file as described in Specifying Multiple Connections to the TimesTen Server. If you have set both the server connection attributes and the related `timesten.conf` daemon attributes, the value of the server connection attributes takes precedence.

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

- `Connections` attribute: Sets the upper limit of the number of concurrent connections to the database. TimesTen allocates one semaphore for each expected connection, therefore the `SEMMNS` kernel parameter has to be set properly. Make sure the connections attribute is set large enough to accommodate the number of concurrent database connections expected.

  See Connections in the *Oracle TimesTen In-Memory Database Reference* or Set the Semaphore Values and Configure shmmax and shmall in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.

- `ServerStackSize`: Set the stack size for each thread on the server.

  The `ServerStackSize` attribute is ignored unless the `MaxConnsPerServer` attribute is greater than one. If there is only one connection per server, the server process uses the process' main stack. Consider setting the `ServerStackSize` attribute to 1024 KB or greater if you are running complex SQL queries on your client connections.

- `MaxConnsPerServer`: Set the maximum number of client connections that are handled by a single server process. Setting `MaxConnsPerServer` > 1 enables multithreaded mode for the database so that it can use threads instead of processes to handle client connections. This reduces the time required for applications to establish new connections and increases overall efficiency in configurations that use a large number of concurrent client/server connections.

  The server, referenced by the server DSN, may have multiple server processes, where each process can only have the maximum number of connections as specified by this attribute. A new server process is automatically started if the number of connections exceeds `MaxConnsPerServer`.

> **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 `MaxConnsPerServer` that balances your requirements for reliability and availability versus the more efficient resource usage of multithreaded mode.

- `ServersPerDSN`: You can have multiple server processes serving multiple incoming connections on the server. The `ServersPerDSN` attribute specifies the number of server processes to spawn in connection with the `MaxConnsPerServer` attribute for distribution of incoming connections.

The `MaxConnsPerServer` and `ServersPerDSN` attributes are related in that they do not limit the number of connections, but control how connections are distributed over server processes. 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* server processes. If additional client connections beyond *N* * `MaxConnsPerServer` are opened to the same server DSN, then those connections are assigned to new server processes sequentially. For example, if you have 12 connections and `MaxConnsPerServer=3`, then there are four server processes.

This results in the following behavior:

1. The first incoming connection spawns a new server process. Each additional incoming connection spawns a new server process up to the `ServersPerDSN` value.
2. The next `ServersPerDSN` number of incoming connections are assigned to the existing server processes in a round-robin fashion up to `MaxConnsPerServer` connections per server.
3. Once the `MaxConnsPerServer` is reached for servers spawned up to `ServersPerDSN`, the next connection spawns another server process and repeats the process detailed in steps 1 and 2.

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.

See ServersPerDSN, ServerStackSize, and MaxConnsPerServer in the *Oracle TimesTen In-Memory Database Reference*.

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 Linux and UNIX platforms:

- Creating and Configuring a Logical Server Name on Windows
- Creating and Configuring a Logical Server Name on Linux and UNIX

Creating and Configuring a Logical Server Name on Windows

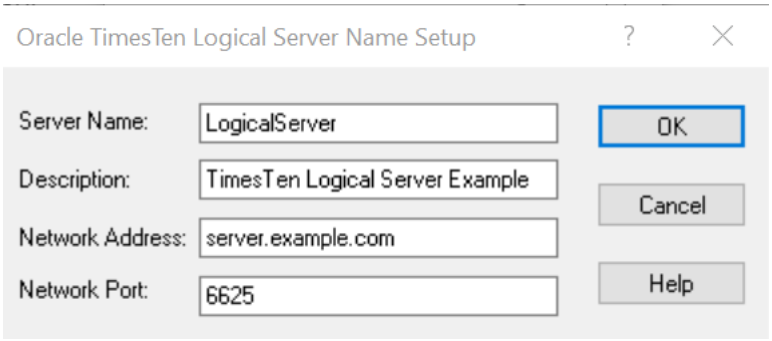
You can create and configure a logical server name on Windows.

1. On the Windows Desktop from the **Start** menu, select **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**.

   > **Note:**
   >
   > If no Client DSN exists, click **Add**, select **TimesTen Client 22.1** and click **Finish**. This opens the TimesTen Client DSN Setup dialog. See Creating and Configuring Client DSNs on Windows for directions on creating a Client DSN in the 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 the name of the system where the TimesTen server is running. For example, `example.com`
9. In the **Network Port** field, TimesTen displays the port number on which the TimesTen Logical Server listens by default. If the TimesTen server is listening on a different port, enter that port number in the **Network Port** field.

   For example:



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

To delete a server name:

1. On the Windows Desktop on the client system from the **Start** menu, 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 Linux and 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 *timesten_home*`/conf/sys.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 *timesten_home*`/conf/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 Linux and UNIX in the `ttconnect.ini` file. The format of a TimesTen Server specification in the `ttconnect.ini` file is shown in Table 3-1.

Table 3-1 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 22.1 is 6625.

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.example.com`
Local client/server connection that uses UNIX domain sockets	`ttLocalHost`

The following example shows a `ttconnect.ini` file that defines a logical server name, `LogicalServer`, for a TimesTen server running on the system `server.example.com` and listens on port `6625`. The instance name of the TimesTen installation is `instance`.

```
[LogicalServer]
Description=TimesTen server 22.1
Network_Address=server.example.com
TCP_Port=6625
```

If both the client and server are on the same Linux or UNIX system, applications using the TimesTen client 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]
Description=Local TimesTen Server 22.1 through domain sockets
Network_Address=ttLocalHost
TCP_Port=6625
```

Defining 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 List of 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 Linux and UNIX platforms:

- Creating and Configuring Client DSNs on Windows
- Creating and Configuring Client DSNs on Linux and UNIX

Creating and Configuring Client DSNs on Windows

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

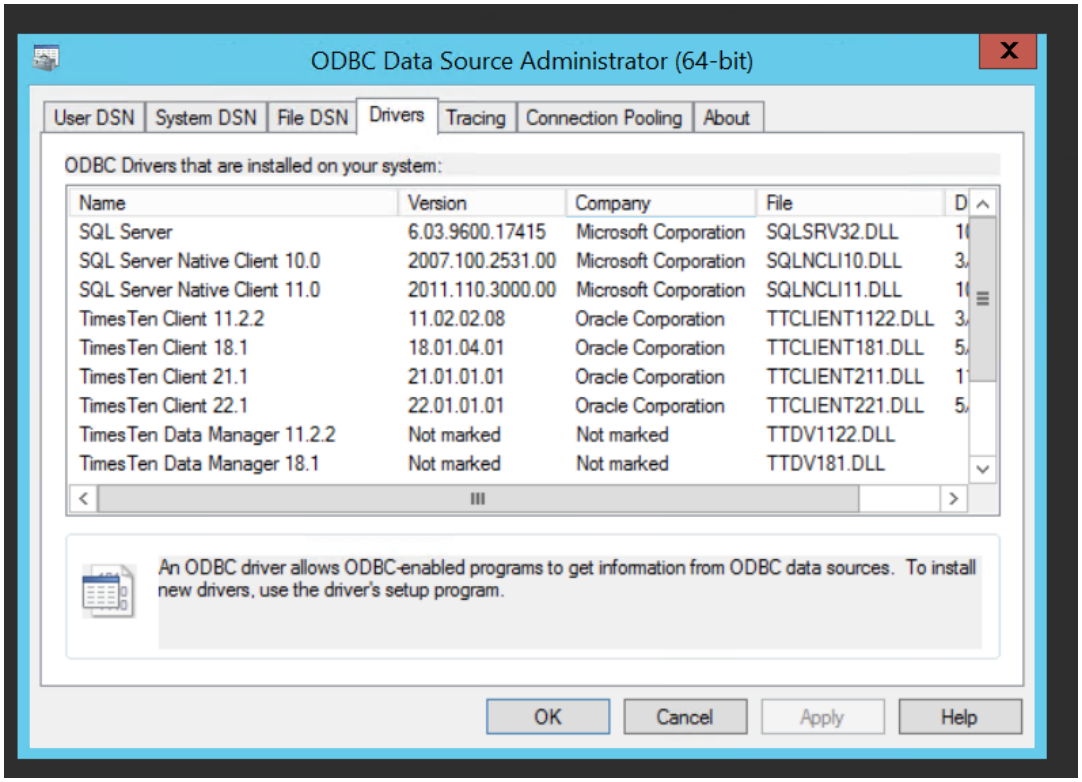
This section includes the following topics:

- Creating a Client DSN on Windows
- Setting the Network Timeout Interval and Authentication
- Accessing a Remote Database on Windows
- Testing Connections

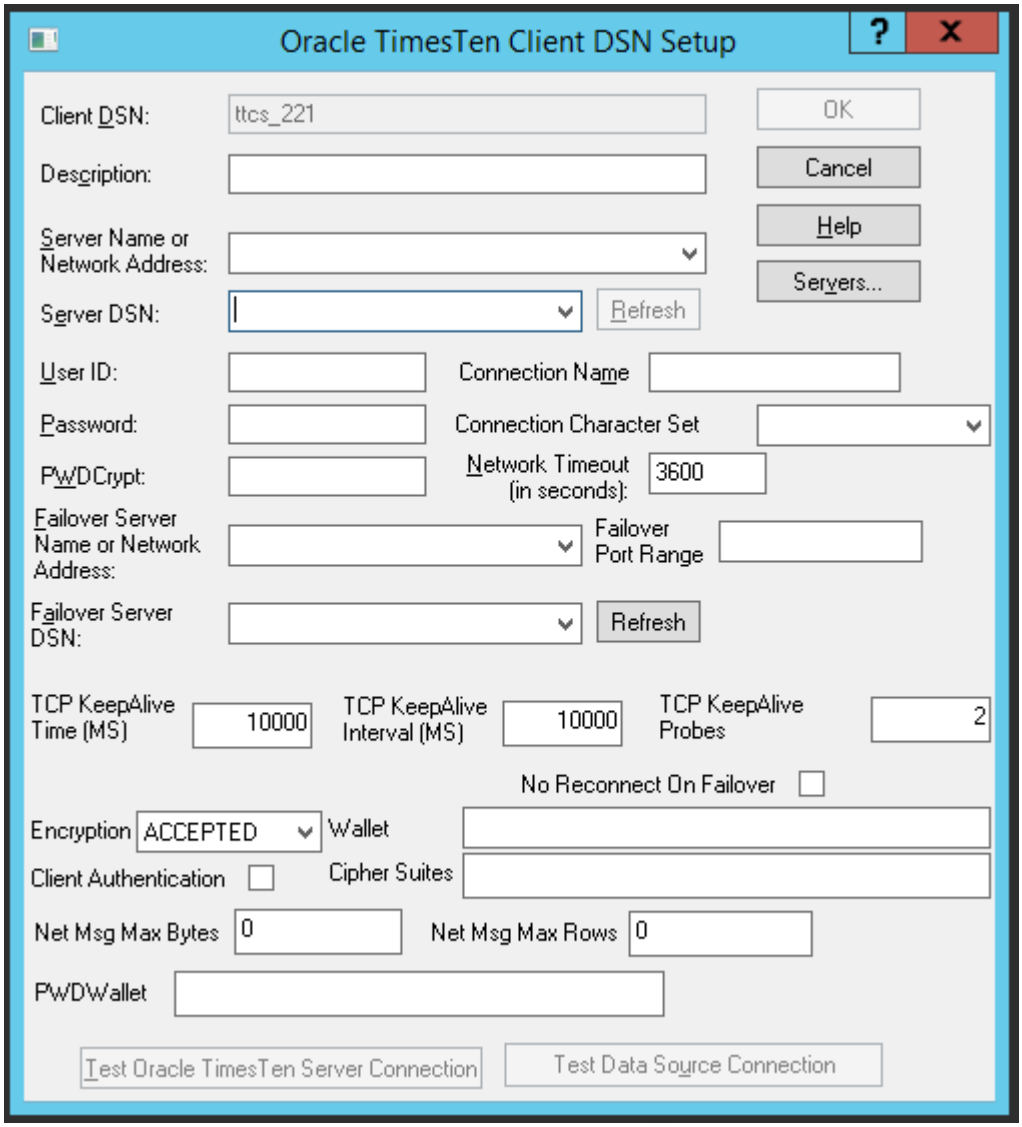
Creating a Client DSN on Windows

You can define a TimesTen client DSN on Windows.

1. On the Windows Desktop from the **Search** menu, search for and select ODBC Data Source Administrator (64 bit).
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.
3. Click **Add**. This opens the Create New Data Source dialog. Select the Drivers tab.



4. Choose **TimesTen Client 22.1** to select the TimesTen client driver. Click **Finish**. This opens the Oracle TimesTen Client DSN Setup dialog.



5. In the **Client DSN** field, enter a name for the Client DSN.

   The name must be unique to the current list of defined DSNs on the 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.

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.

   See ODBC Data Sources for more information about customizing which server DSNs show in this list.
9. In the **PwdWallet** field, supply the directory location to the Oracle Wallet to use for user credentials. See Providing a User Name and Password in an Oracle Wallet in the *Oracle TimesTen In-Memory Database Security Guide*.
10. 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`. See ConnectionCharacterSet in *Oracle TimesTen In-Memory Database Reference*.
11. If you are using automatic client failover, you would configure the **Failover Server Name**, **Failover Port Range**, **Failover DSN**, and **No Reconnect On Failover.** See Using Automatic Client Failover.
12. If you are using TimesTen Scaleout, you would configure the **TCP KeepAlive Time**, **TCP KeepAlive Interval**, and **TCP KeepAlive Probes**. See Configuring TCP Keep-Alive Parameters in this guide and TTC_TCP_KEEPALIVE_TIME_MS, TTC_TCP_KEEPALIVE_INTVL_MS, and TTC_TCP_KEEPALIVE_PROBES in the *Oracle TimesTen In-Memory Database Reference*.

Setting the Network Timeout Interval and Authentication

You can define the user name, password and network timeout interval for the client/server connection in the client DSN with the `UID`, `PWD`, and `TTC_TIMEOUT` attributes.

However, configuring the authentication in the client DSN is optional since you can provide the user name and password when connecting. Also, supplying the password in the client DSN is strongly discouraged since the password is stored unencrypted.

Alternatively, TimesTen enables you to store user names and associated passwords in an Oracle Wallet. This is the most secure and preferred method of providing credentials for connecting to a TimesTen database. However, you would provide the location and name of the Oracle Wallet on the connect string. For more information on creating the wallet and managing Oracle cache administration user IDs and passwords, see Providing a User Name and Password in an Oracle Wallet in *Oracle TimesTen In-Memory Database Security Guide*.

For a description of the `UID`, `PWD`, and `TTC_TIMEOUT` connection attributes, see List of Connection Attributes in *Oracle TimesTen In-Memory Database Reference*.

To set the network 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 a hashed password in the **PwdCrypt** field or use the `ttUser` utility to store the password in an Oracle Wallet and retrieve it from the wallet using the **PwdWallet** field. See `ttUser` in the *Oracle TimesTen In-Memory Database Reference*.

Provide the password in one of the following ways:

- By storing the password in an Oracle Wallet and then entering the path to the wallet in the **PwdWallet** field.
- By entering the password that corresponds to the user ID in the **Password** field.
- By entering an encrypted password that corresponds to the user ID in the **PwdCrypt** field

3. In the **Network Timeout** 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.

   The `TTC_Timeout` connection attribute sets a maximum time limit for a network operation that is completed by using the TimesTen client and server. The `TTC_Timeout` connection attribute also determines the maximum number of seconds a TimesTen client application waits for the result from the corresponding TimesTen server process before timing out.

4. Click **OK** to save the setup.

Accessing a Remote Database on Windows

In this example, the TimesTen client system is `client.example.com`. The client application is accessing the server DSN on the remote server system, `server.example.com`. The logical server name is `LogicalServer`.

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

1. On the server system `server.example.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 for TimesTen Server on a Linux or UNIX System, verify that the server DSN, `database1`, is defined as a system DSN on `server.example.com`.
3. On the client system, `client.example.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`.
   - In the Network Address field, enter `server.example.com`.
   - In the Network Port field, enter `6625`. This is the default port number for the TimesTen server for TimesTen Release 22.1. This value should correspond to the value displayed by `ttStatus` in Step 1.
4. On the client system, `client.example.com`, create a client DSN that corresponds to the remote server DSN, `database1CS`. In the TimesTen client DSN Setup dialog, enter the following values:
   - In the **Client DSN** field, enter `database1CS`.
   - In the **Server Name** or **Network Address** field, enter `LogicalServer`.
   - In the **Description** field, enter a description for the server. Entering data into this field is optional.

- In the **Server DSN** field, enter `database1`.

5. Run the client application from the system `client.example.com` using the client DSN, `database1CS`. The example below uses the `ttIsql` program installed with TimesTen client.

```
ttIsql -connStr "DSN=database1CS"
```

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.example.com` and the server is listening on port 6625. The following methods can be used to connect to a server DS:

1. Define the logical server name `LogicalServer` with `server.example.com` as the Network Address and `6625` as the Network Port. Define `Client_DSN` as the client DSN with `LogicalServer` as the server name and `Server_DSN` as the server DSN. Then, run the command:

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

2. Alternatively, define the logical server name `LogicalServer_tt` with `server.example.com` as the Network Address and the default port number as the Network Port. Define `Client_DSN` as the client DSN with `LogicalServer` 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=6625"
```

See Creating and Configuring a Logical Server Name on Windows.

Testing Connections

You can 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.
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 Linux and UNIX

On Linux and 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 `sys.odbc.ini` file for system DSNs.

> **Note:**
>
> The syntax for defining the Client DSN in the `odbc.ini` file is described in odbc.ini File Entry Descriptions. See Defining a Logical Server Name for details on the `ttconnect.ini` file. See Specifying Data Source Names to Identify TimesTen Databases for a description of user and system DSNs.

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:

- TimesTen ODBC client driver to use for the connection.
- The server (and the port number if you do not want to use the default port number) for the database 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.
>
> See Connecting Using TimesTen ODBC Drivers for a list of all available ODBC client drivers.

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 TimesTen ODBC client driver to use in the ODBC Data Sources section.
- 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 `database1CS` 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]
database1CS=TimesTen Client 22.1 Driver
```

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 `database1CS` where the server is configured with a server name of `server.example.com` with port of 6625 and the server DSN is `database1`.

```
[database1CS]
TTC_Server=server.example.com/6625
TTC_Server_DSN=database1
```

If you want to use logical server names, then the following is an example specification of the TimesTen client DSN `database1CS` where the server is configured with a logical server name of `LogicalServer` and the server DSN is `database1`:

```
[database1CS]
TTC_Server=LogicalServer
TTC_Server_DSN=database1
```

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.

> **Note:**
>
> See UID and PWD and PwdWallet in the *Oracle TimesTen In-Memory Database Reference* for options on how to provide the user ID when connecting.

By default, connections wait 60 seconds for TimesTen client and server to complete a network operation. The `TTC_Timeout` attribute sets a maximum time limit for a network operation that is completed by using the TimesTen client and server. The `TTC_Timeout` connection attribute also determines the maximum number of seconds a TimesTen client application waits for the result from the corresponding TimesTen server process before timing out.

```
[database1CS]
TTC_Server=server.example.com/6625
TTC_Server_DSN=database1
TTC_Timeout=60
```

Once the TimesTen client DSN is configured, you can connect to the server using the `ttIsqlCS` utility. The following connects to the server using the `database1CS` DSN.

```
ttIsqlCS -connStr "DSN=database1CS"
```

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*.

Sizing the Client Result Set Buffer

Executing a `SELECT` statement returns a result set (if there is one). The server stores a maximum number of rows from the result set that fits into a client result set buffer.

You can improve performance by changing the maximum size of the result set buffer or the maximum number of rows that can be stored into a client result set buffer with the following connection attributes:

- `TTC_NetMsgMaxBytes`: The size of a client result set buffer in bytes. The default is 2097152 bytes.
- `TTC_NetMsgMaxRows`: The maximum number of rows that can be stored in a client result set buffer. The default is 8192 rows.

The `SELECT` statement returns the maximum number of bytes or rows into a client result set buffer, whichever limit is reached first. For queries that return large result sets, you may want to increase the values of both parameters.

See Configuring the Result Set Buffer Size in Client/Server Using ODBC and Configuring the Result Set Buffer Size in Client/Server Using OCI in the *Oracle TimesTen In-Memory Database C Developer's Guide* and Configuring the Result Set Buffer Size in Client/Server Using JDBC in the *Oracle TimesTen In-Memory Database Java Developer's Guide* for details on setting these limits using ODBC, OCI and JDBC programmatic APIs.

Using Automatic Client Failover

Automatic client failover is for use in high availability client/server scenarios with TimesTen Scaleout or TimesTen Classic.

If there is a failure of the TimesTen database to which the client is connected, then failover (connection transfer) to an alternate TimesTen database occurs. Failover connections are created only as needed.

- For TimesTen Scaleout, the connection is transferred to an available data instance from a list of available data instances in the grid (and is not dependent on the DSN configuration in your `odbc.ini` file).
- For TimesTen Classic, you can use automatic client failover in two scenarios:
  - Active standby pair replication scheme: If you have an active standby pair replication configuration, then the connection is transferred to the new active (original standby) master within an active standby pair replication configuration. Consider a scenario where failure of the active master has resulted in the original standby master becoming the new active master. 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 master.

    > **Note:**
    >
    > Automatic client failover in TimesTen Classic (when using an active standby pair replication scheme) 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 *Oracle TimesTen In-Memory Database Replication Guide*.

  - Generic automatic client failover: If you create more than one database and have a process to ensure that the data is consistent on both databases, then the client can automatically fail over from one database to another. For example, with a suitable cache configuration, TimesTen automatically synchronizes the data in all of the included TimesTen databases with the data in the Oracle database. In such a scenario,

applications can safely failover between multiple caches without any concern about data consistency.

In each of these scenarios, applications are automatically connected to the failover TimesTen database. TimesTen provides features that enable applications to be alerted when this happens, enabling the application to take appropriate action.

> **Note:**
>
> If you issue an `ALTER SESSION` statement anytime after the initial connection, you must re-issue the statement after a failover. For information about connection persistence after failover, see ODBC Support for Automatic Client Failover in *Oracle TimesTen In-Memory Database C Developer's Guide*.

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

- Managing Automatic Client Failover in TimesTen Scaleout
- Managing Automatic Client Failover in TimesTen Classic Using an Active Standby Pair
- Managing Generic Automatic Client Failover in TimesTen Classic
- Using Automatic Client Failover

Managing Automatic Client Failover in TimesTen Scaleout

By default, if a connection fails in TimesTen Scaleout, then the client automatically attempts to reconnect to another data instance (if possible).

See Client Connection Failover in the *Oracle TimesTen In-Memory Database Scaleout User's Guide* for full details on managing client connection failover in TimesTen Scaleout.

Managing Automatic Client Failover in TimesTen Classic Using an Active Standby Pair

When you use an active standby pair replication scheme in TimesTen Classic and an application connects to the active master, then the connection is registered and this registration is replicated to the standby master. If the active master fails, the standby master becomes the active master and then notifies the client of the failover. At this point, the client has a new connection to the new active master.
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.

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

- If the client library loses the connection to the active master, it fails over and attempts to switch to the standby master.
- If, for some reason, there is no active master (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 master or times out.

- If a failover has already occurred and the client is already connected to the new active master, the next failover request results in an attempt to reconnect to the original active master. If that fails, alternating attempts are made to connect to the two servers until it times out.
- If the active master fails before the client registration is successfully propagated by replication to the standby master, 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 master is lost and initiates a failover attempt.

> **Note:**
>
> Using automatic client failover with an active standby pair replication scheme results in one additional database connection per server process. This should be considered when you choose a setting for the TimesTen `Connections` attribute (the 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 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, then there are four additional connections.

For C developers, details of how to create the callback and to facilitate an automatic client failover are discussed in ODBC Support for Automatic Client Failover 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*.

Managing Generic Automatic Client Failover in TimesTen Classic

You can configure generic automatic client failover across any set of TimesTen Classic databases.

You can configure a list of failover TimesTen databases in the user or system `odbc.ini` file. Once configured, the application can try to connect to each of them until a successful connection is made. This feature applies only to client/server connections, not direct connections.

> **Note:**
>
> You must ensure that automatic client failover is meaningful in your environment. That is, there is no possibility of any data consistency issues when failing over between the included databases. In addition, the client can locate the necessary tables and data in any database to which it fails over.

Consider the following for each TimesTen Classic database included in the failover server list:

- Participating databases must be TimesTen release 18.1.4.1.0 or later.
- The same database user name and password must be used for client connections to all databases.

- The failover databases should have the same table and/or cache group definitions. You must have a process to ensure that the data is consistent on all databases for all tables and/or cache group tables.
- To facilitate a fast failover, all of the databases in the client failover list should be loaded in memory.
- When using cache, the databases should have operational cache agents at all times.
- The databases included in the generic automatic failover do not use an active standby pair replication scheme.

A typical use case for generic automatic failover is a set of databases using read-only caching, where each database has the same set of cached data. For example, if you have several read-only cache groups, then you would create the same read-only cache groups on all TimesTen Classic databases included in the list of failover servers. When the client connection fails over to an alternate TimesTen database, the cached data is consistent since cache operations automatically refresh the data (as needed) from the Oracle database. See Read-Only Cache Group in the *Oracle TimesTen In-Memory Database Cache Guide* for more details on read-only cache groups.

> **Note:**
>
> When failover happens, the application will need to re-prepare all statements.

For C developers, details of how to create the callback and to facilitate an automatic client failover are discussed in ODBC Support for Automatic Client Failover 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*.

Configuring Automatic Client Failover for TimesTen Classic

A list of failover TimesTen databases can be configured on the client. Then, clients initiate a connection to each failover TimesTen database, until a successful connection is made.

> **Note:**
>
> The failover server information you provide depends on which type of automatic client failover you want to use. That is, the failover server should be either the standby database (when using an active standby pair replication scheme) or a second TimesTen database (when using read-only cache groups). See Using Automatic Client Failover for details on both types of automatic client failover.

The following sections describe how to configure for automatic client failover for TimesTen Classic:

- Configuring Automatic Client Failover for TimesTen Classic on Windows
- Configuring Automatic Client Failover for TimesTen Classic on Linux and UNIX

Configuring Automatic Client Failover for TimesTen Classic on Windows

In the 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, complete the following fields:

1. In the **Failover Server Name or Network Address** field, specify the logical server or network address of the failover server.
   - 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.

     See Creating and Configuring a Logical Server Name on Windows for more information on defining logical server names.
2. In the **Failover Server DSN** field, enter the server DSN corresponding to the failover server.
   - 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 database is running.
3. If you are using an active standby pair replication scheme for automatic client failover, then you can optionally specify the **Failover Port Range** for the port or port range where TimesTen listens for failover notifications. Specifying a port range helps accommodate firewalls between client and server systems. With the active standby pair, the active and standby masters notify the clients to failover.

   By default, TimesTen uses a single port chosen by the operating system. 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 need multiple ports on the firewall between the server and 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.
4. Optionally, select the **No Reconnect On Failover** check box to have TimesTen perform all the usual client failover processing except for the reconnect. For example, statement and connection handles are marked as invalid. This is useful if the application provides its own connection pooling or manages its own reconnection to the database after client failover.

Configuring Automatic Client Failover for TimesTen Classic on Linux and UNIX

You can configure for automatic client failover for TimesTen Classic.

On the client, configure the following:

1. In the `odbc.ini` file on the client, define the Client DSNs. Specify the failover servers with the `TTC_ServerN` connection attribute, where *N* >= 2. The `TTC_ServerN` connection attribute can specify the DNS name, host name or IP address of the system (with or without the port number).

   Set the `TTC_Server_DSNn` connection attribute to the name of this database, where *n* >= 2.

   Setting any of `TTC_ServerN` or `TTC_Server_DSNn` implies the following:

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

   If you are using automatic client failover for read-only cache groups, you can specify more than one failover server with the `TTC_ServerN` and `TTC_Server_DSNn` connection attributes, where *N,n* >= 2.

   You should specify your failover servers (identified by *N*) in a sequential order.

   For example:

   ```
   [MyDSN]
   TTC_Server=server.example.com/6625
   TTC_Server_DSN=MyDSN
   TTC_Timeout=60
   TTC_Server2=server2.example.com/6625
   TTC_Server_DSN2=MyDSN_Failover1
   TTC_Server3=server3.example.com/6625
   TTC_Server_DSN3=MyDSN_Failover2
   ```

   > **Note:**
   >
   > - Optionally, you can define logical server names for your failover servers in the `ttconnect.ini` file on the client with the `Network_Address` and `TCP_Port`. See Creating and Configuring a Logical Server Name on Linux and UNIX for more information on defining logical server names.
   > - Like other connection attributes, `TTC_ServerN` and `TTC_Server_DSNn` can be specified in the connection string, overriding any settings in the DSN. The default TCP/IP port number is assumed for `TCP_Port` and `TCP_PortN` unless you specify a value in the `TTC_ServerN` connection attribute, in the connection string, or in the logical server definition in `ttconnect.ini`.

   Refer to TTC_Server or TTC_Server1, TTC_Server2, TTC_ServerN, TTC_Server_DSN2, TTC_Server_DSNn, and TCP_Port2, TCP_PortN in *Oracle TimesTen In-Memory Database Reference*.

2. If you are using automatic client failover with read-only cache groups and you have specified more than one alternate failover server, you can specify how the client selects the failover server. By default, the client selects from a randomly ordered list of servers provided by `TTC_Server` and `TTC_ServerN` attributes.

   However, if you set `TTC_Random_Selection` to 0, then the client selects the first server specified by the `TTC_ServerN` connection attributes. If the client cannot connect to the first server specified, the client proceeds to the next failover server in the order specified numerically.

> **Note:**
>
> If you are using an active standby pair for automatic client failover and the client library cannot connect to the active master specified by `TTC_Server_DSN`, then it tries the standby master specified by `TTC_Server_DSN2`.

Refer to TTC_Random_Selection in *Oracle TimesTen In-Memory Database Reference*.

3. Optionally, set a timeout to specify a duration for all failover recovery attempts in the `odbc.ini` file on the client.

   - By default, connections wait 60 seconds for TimesTen client and server to complete a network operation. The `TTC_Timeout` attribute determines the maximum number of seconds a TimesTen client application waits for the result from the corresponding TimesTen server process before timing out.

     The `TTC_TIMEOUT` connection attribute also specifies the duration for all attempts at failover recovery. The connection might be blocked while failover is attempted.

   - Since you are using multiple servers for automatic client failover, then you can also set the `TTC_ConnectTimeout` attribute to specify the maximum number of seconds the client waits for a `SQLDriverConnect` or `SQLDisconnect` request. It overrides the value of `TTC_Timeout` for those requests. Set the `TTC_ConnectTimeout` when you want connection requests to timeout with a different timeframe than the timeout provided for query requests. For example, you can set a longer timeout for connections if you know that it takes a longer time to connect, but set a shorter timeout for all other queries.

```
[MyDSN]
TTC_Server=server.example.com/6625
TTC_Server_DSN=MyDSN
TTC_Timeout=60
TTC_ConnectTimeout=80
TTC_Server2=server2.example.com/6625
TTC_Server_DSN2=MyDSN_Failover1
TTC_Server3=server3.example.com/6625
TTC_Server_DSN3=MyDSN_Failover2
```

   If automatic client failover is defined and there is more than one `TTC_SERVER`*n* connection attribute configured, then the overall elapsed time is four times the value of the timeout (whether `TTC_Timeout` or `TTC_ConnectTimeout`). For example, if the user specifies `TTC_TIMEOUT`=10 and there are multiple `TTC_SERVER`*n* servers configured, then the `SQLDriverConnect` call could take up to 40 seconds.

   Refer to TTC_Timeout in *Oracle TimesTen In-Memory Database Reference*.

4. If you are using an active standby pair replication scheme for client failover, then you can optionally configure the `TTC_FailoverPortRange` attribute to specify a port or port range where the failover thread listens for failover notifications. Specifying a port range helps accommodate firewalls between client and server systems. With the active standby pair, the active and standby masters notify the clients to failover.

   By default, TimesTen uses a single port chosen by the operating system. 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 need multiple ports on the firewall between the server and 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.

5. Optionally, enable the `TTC_NoReconnectOnFailover` connnection attribute to have TimesTen perform typical client failover, but without reconnecting. This is useful where an application does its own connection pooling or attempts to reconnect to the database on its own after failover. The default value is 1 and indicates that TimesTen should attempt to reconnect.

Configuring TCP Keep-Alive Parameters

One of the ways that a client connection can fail is with a network failure, such as disconnecting a cable or a host that is suspending or crashing.

When the client connection is lost, then client connection failover is initiated. However, when a TCP connection is started, you can configure the TCP keep-alive parameters for the connection to ensure reliable and rapid detection of connection failures.

> **Note:**
>
> You can also detect that there is a problem with the connection by setting the `TTC_Timeout` attribute, which sets a maximum time limit for a network operation that is completed by using the TimesTen client and server. The `TTC_Timeout` attribute also determines the maximum number of seconds a TimesTen client application waits for the result from the corresponding TimesTen Server process before timing out.
>
> TimesTen recommends configuring the TCP keep-alive parameters for determining a failed TCP connection in addition to the `TTC_TIMEOUT` attribute, as some database operations may unexpectedly take longer than the value set for the `TTC_TIMEOUT` attribute.
>
> Refer to TTC_Timeout in *Oracle TimesTen In-Memory Database Reference*.

You can control the per connection keep-alive settings with the following parameters:

- `TTC_TCP_KEEPALIVE_TIME_MS`: The duration time (in milliseconds) between the last data packet sent and the first probe. The default is 10000 milliseconds.

  > **Note:**
  >
  > The Linux client platform converts this value to seconds by truncating the last three digits off of the value of `TTC_TCP_KEEPALIVE_TIME_MS`. Thus, a setting of 2500 milliseconds becomes 2 seconds, instead of 2.5 seconds.

- `TTC_TCP_KEEPALIVE_INTVL_MS`: The time interval (in milliseconds) between subsequential probes. The default is 10000 milliseconds.
- `TTC_TCP_KEEPALIVE_PROBES`: The number of unacknowledged probes to send before considering the connection as failed and notifying the client. The default is set to 2 unacknowledged probes.

If you keep the default settings, then TimesTen sends the first probe after 10 seconds (the `TTC_TCP_KEEPALIVE_TIME_MS` setting).

- If there is a response, then the connection is alive and the `TTC_TCP_KEEPALIVE_TIME_MS` timer is reset.
- If there is no response, then TimesTen sends another probe after this initial probe at 10 second intervals (the `TTC_TCP_KEEPALIVE_INTVL_MS` setting). If no response is received after 2 successive probes, then this connection is terminated and TimesTen redirects the connection to another data instance.

For example, you could modify the TCP keep alive settings to have a shorter wait time for the initial probe of 50000 milliseconds, and to check for a connection every 20000 milliseconds for a maximum number of 3 times as follows:

```
TTC_TCP_KEEPALIVE_TIME_MS=50000
TTC_TCP_KEEPALIVE_INTVL_MS=20000
TTC_TCP_KEEPALIVE_PROBES=3
```

See the TTC_TCP_KEEPALIVE_TIME_MS, TTC_TCP_KEEPALIVE_INTVL_MS, and TTC_TCP_KEEPALIVE_PROBES sections in the *Oracle TimesTen In-Memory Database Reference*.

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 is started or stopped.

You can explicitly start or shut down the daemon or service with the `ttDaemonAdmin` utility (use `ttGridAdmin instanceExec` to run this command on TimesTen Scaleout). The TimesTen server is run as the instance administrator.

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 The ttInstanceCreate Utility chapter in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade 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.

For instructions on modifying TimesTen server options, see Modifying the TimesTen Server Attributes.

Server Informational Messages

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

On Linux and UNIX, the TimesTen server logs messages to the *timesten_home*`/diag/ttmesg.log` and *timesten_home*`/diag/tterrors.log` files by default. Optionally, you can configure TimesTen to log messages to the `syslog` facility.

See Error, Warning, and Informational Messages.

Accessing a Remote Database on Linux and UNIX

In this example, the TimesTen Client application system is `client.example.com`. The client application is accessing the Server DSN `database1` on the remote server system, `server.example.com`. The logical server name is `LogicalServer`. The instance name of the TimesTen installation is `instance`.

1. On the server system `server.example.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 `database1` exists in the system `sys.odbc.ini` file on `server.example.com`.

   There should be an entry in the `sys.odbc.ini` file as follows. Note that `/disk1/timesten` is the *`timesten_home`*.

   ```
   [database1]
   DataStore=/disk1/timesten/server/database1
   ```

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

   ```
   [LogicalServer]
   Network_Address=server.example.com
   TCP_Port=6625
   ```

4. On the client system, `client.example.com`, create a client DSN corresponding to the remote server DSN, `database1`.

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

   ```
   [database1CS]
   TTC_Server=LogicalServer
   TTC_Server_DSN=database1
   ```

5. Run the client application from the system `client.example.com` using the client DSN, `database1CS`. The example below uses the `ttIsql` program that is installed with TimesTen client.

   ```
   ttIsqlCS -connStr "DSN=database1CS"
   ```

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.example.com` and the server is listening on Port `6625`. The following methods can be used to connect to a server DSN:

1. Define the logical server name `LogicalServer` with `server.example.com` as the network address and `6625` as the network port. Define `Client_DSN` as the client DSN with `LogicalServer` as the server name and `Server_DSN` as the server DSN. Run the command:

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

2. Alternatively, define the logical server name `logical_server` with `server.example.com` as the network address and the default port number as the network port. Define a client DSN with `LogicalServer` 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=6625"
   ```

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.example.com;TTC_Server_DSN=Server_DSN;TCP_Port=6625"
```

See Creating and Configuring Client DSNs on Linux and UNIX for information on the creating a `ttconnect.ini` file. See Overview of User and System DSNs for information on the location of the proper `odbc.ini` file.

Testing Connections

You can 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.example.com 6625
     ```

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

     ```
     Connected to server.example.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. See Modifying the TimesTen Server Attributes.
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 `sys.odbc.ini` file on the host where the database resides and where the TimesTen Server is running.

4

Working with the TimesTen Daemon

The TimesTen daemon starts when TimesTen is installed. The daemon runs in the background.

The TimesTen daemon performs the following functions:

- Starts the TimesTen server
- 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 for TimesTen Classic
- If requested, starts replication processes 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:

Starting and Stopping the Daemon

Automatically Starting the TimesTen Daemon

You have two options for automatic management of the TimesTen daemon.

Choose only one of these options.

- Automatically start and stop TimesTen daemon when the operating system boots and shuts down. See Using the setuproot Script to Automatically Start the TimesTen Daemon.
- Automatically manage the TimesTen daemon using systemd. See Using systemd to Automatically Manage the TimesTen Daemon.

> **Note:**
>
> See Manually Starting and Stopping the Daemon for details on how to use the `ttDaemonAdmin` utility to manually start and stop the TimesTen daemon.

Using the setuproot Script to Automatically Start the TimesTen Daemon

If you run the `setuproot` script for SysV, then the TimesTen daemon is automatically started and stopped when the operating system boots and shuts down.

When you run the TimesTen daemon startup `setuproot` script (as root), this script installs the appropriate SysV initialization scripts for TimesTen in the `/etc/init.d` directory.

The following sections describe the differences between running the `setuproot` script for TimesTen Classic or TimesTen Scaleout:

- Using the setuproot script for SysV on TimesTen Classic
- Using the setuproot script for SysV on TimesTen Scaleout

Using the setuproot script for SysV on TimesTen Classic

You can find the `setuproot` script in the *timesten_home*`/bin` directory. After the SysV intialization scripts are installed, TimesTen is automatically started when the operating system boots and automatically stopped on operating system shutdown.

When using SysV, run the `setuproot` script as follows:

```
cd $TIMESTEN_HOME/bin
setuproot -install
```

In all other scenarios, the instance administrator uses the `ttDaemonAdmin` utility to manage the TimesTen daemon. See ttDaemonAdmin in the *Oracle TimesTen In-Memory Database Reference*.

Using the setuproot script for SysV on TimesTen Scaleout

You can find the `setuproot` script in the *timesten_home*`/bin` directory of every instance of the grid. After you run the `setuproot` script, the daemon is automatically started for each data instance as part of the operations for the `ttGridAdmin dbOpen` or `ttGridAdmin modelApply` commands.

For TimesTen Scaleout, the directions on how to run the `setuproot` script are in the Setting Instances to Automatically Start at System Startup section in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

Using systemd to Automatically Manage the TimesTen Daemon

For TimesTen Classic, the root user can set up systemd to use for automatic management (including starting and stopping) of the TimesTen daemon.

Once you setup and start systemd, you can no longer use the `ttDaemonAdmin` utility to manage the daemon. See Using systemd to Manage a TimesTen Service in *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide* for full details on installing, configuring, and managing systemd.

Perform the following to setup systemd to automatically manage the TimesTen daemon:

> **Note:**
>
> These steps are an overview of what you must perform to use systemd. See About Creating an Instance on Linux/UNIX in *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.

1. Create the TimesTen instance with systemd.
2. Modify the TimesTen service file.
3. As the root user, run the *timesten_home*`/bin/setuproot` utility script with the `-install -systemd` options. The setuproot script copies the systemd initialization scripts to the appropriate location.
4. As the root user, run the systemd `systemctl start` command to start the TimesTen service.

> **Note:**
>
> You can use the systemd `systemctl` commands to manually control systemd.

Once set up and started, systemd takes precedence over any SysV initialization scripts.

Manually Starting and Stopping the Daemon

By default, you can use the `ttDaemonAdmin` utility to start and stop the TimesTen daemon.

You must be the instance administrator to start and stop the TimesTen daemon with the `ttDaemonAdmin` utility.

> **Note:**
>
> See Automatically Starting the TimesTen Daemon for details on how to automatically start and stop the TimesTen daemon.

- For TimesTen Classic, the instance administrator manually manages the TimesTen daemon. To manually start and stop the TimesTen main daemon, use the `ttDaemonAdmin` utility with the `-start` or `-stop` option.
- For TimesTen Scaleout, run the `ttDaemonAdmin -start` or `-stop` options within the `ttGridAdmin instanceExec` command to manually start the daemon for a data instance.

For more information on `ttDaemonAdmin`, see ttDaemonAdmin. For details on `ttGridAdmin instanceExec`, see Execute a Command or Script on Grid Instances (instanceExec) in *Oracle TimesTen In-Memory Database Reference*. See Automatically Starting the TimesTen Daemon for details on the `setuproot` script.

Shutting Down a TimesTen Application

You can shut down your TimesTen application using commands described in the following sections:

- Shutting Down an Application Within TimesTen Classic
- Shutting Down Applications Within TimesTen Scaleout

Shutting Down an Application Within TimesTen Classic

An application within TimesTen Classic 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 close all active connections to the TimesTen Classic database, run the `ttAdmin -disconnect` command. See Disconnecting from a Database in this book and ttAdmin in the *Oracle TimesTen In-Memory Database Reference*.

If you want to unload the TimesTen Classic database, see Unloading a Database from Memory for TimesTen Classic.

Shutting Down Applications Within TimesTen Scaleout

With TimesTen Scaleout, there are a few operations to perform when disconnecting all TimesTen applications from the database.

- *Close the database to user connections*. The `ttGridAdmin dbClose` command disables new user connections to a database.
- *Disconnect all applications from the database.* The `ttGridAdmin dbDisconnect` command terminates all user connections to a database.
- *Unload the database from memory*. The `ttGridAdmin dbUnload` command unloads every element of the database from the memory of their respective hosts.

See Unloading a Database from Memory in *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

Managing TimesTen Daemon Attributes

The `timesten.conf` file contains TimesTen daemon attributes.

- In TimesTen Scaleout, you can import the values in the `timesten.conf` file by using the `ttGridAdmin instanceConfigImport` command. See Import Instance Configuration Attributes (instanceConfigImport) in the *Oracle TimesTen In-Memory Database Reference*.
- In TimesTen Classic, you can create or change values for the `timesten.conf` file through connection attributes, through editing this file by hand, or with the `ttInstanceCreate` or `ttInstanceModify` utilities.

  Use the `ttInstanceModify` utility to make changes to the `timesten.conf` file for most commonly changed attributes. If you cannot use `ttInstanceModify` to change a particular attribute and must modify the `timesten.conf` 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 attributes, it is only necessary to stop the server. It is not necessary to stop the TimesTen daemon.

See ttInstanceCreate and ttInstanceModify in the *Oracle TimesTen In-Memory Database Reference*.

For more information about the `timesten.conf` file, see TimesTen Instance Configuration File in the *Oracle TimesTen In-Memory Database Reference*.

On Linux and UNIX, the `timesten.conf` file is located in the daemon home directory:

timesten_home`/conf`

Some features that the `timesten.conf` file controls are:

- 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 daemon log files and user log files
- The duration of daemon logging to save for critical events
- 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 an application using cache, OCI, Pro*C/C++, or ODP.NET. This option cannot be modified in this file.
- Modifying the default database recovery after a fatal error

The rest of this section includes the following topics:

- Determining the Daemon Listening Address
- Managing Subdaemons
- 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.

By default, both IPv4 and IPv6 addresses are used. You can disable the use of IPv6 addresses by setting the `enableIPv6` attribute to 0 on a separate line in the `timesten.conf` file:

```
enableIPv6=0
```

This change takes effect after the TimesTen main daemon restarts. You cannot disable the use of IPv4 addresses.

You can control which IP addresses that TimesTen listens on by specifying those addresses with the `listen_addr` attribute for IPv4 addresses and the `listen_6_addr` attribute for IPv6 addresses in the `timesten.conf` file. The addresses specified with this option can be either a host name or a numerical IP address. If you specify a hostname that resolves to multiple IP addresses, then TimesTen listens on all of those addresses.

The `listen_addr` and `listen_6_addr` attributes exist for situations where a server has multiple network addresses and multiple network cards. You can limit the network addresses on which the TimesTen daemon is listening to a subset of the server's network addresses by making entries only for those addresses on which the daemon listens.

> **Note:**
>
> Use of either the `listen_addr` or `listen_6_addr` attributes disables all available addresses listening for both IPv4 and IPv6. This means that if you set the `listen_addr` attribute, then you also must set the `listen_6_addr` attribute (and vice versa).

For example:

- 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 `listen_addr` attribute and/or a `listen_6_addr` attribute containing only the private address blocks all communications to TimesTen coming on the public address.
- By specifying only the local host (or the loopback addresses), the TimesTen main daemon can be cut off from all communications coming from outside the server and limited to communicate only with local clients and subdaemons.

TimesTen replication does not use the `listen_addr` or `listen_6_addr` attributes. There is no requirement for setting these attributes when replication is used. If replication is going to be used in an environment where `listen_addr` or `listen_6_addr` attributes are used, then the replication scheme needs to be configured to use the allowable network addresses.

To explicitly specify the address on which the daemons should listen, enter on a separate line in the `timesten.conf` file:

```
listen_addr=Ipv4Address
listen_6_addr=Ipv6Address
```

For example, if you want to restrict the daemon to listen to just the loopback addresses, you add:

```
listen_addr=127.0.0.1
listen_6_addr=::1
```

or

```
listen_addr=localhost
listen_6_addr=localhost
```

This means that only processes on the local system can communicate with the daemon. Processes from other systems would be excluded, so operations such as client access from other systems is not possible.

If you have multiple ethernet cards on different subnets, you can specify multiple `listen_addr` and/or `listen_6_addr` attributes to control which addresses the daemon listens on.

For example, if you set both `listen_addr=10.20.30.51` and `listen_6_addr=2020:100:100:300::51` in the `timesten.conf` file, the daemon listens on the IPv4 loopback address, the specified IPv4 address, the IPv6 loopback address and the specified IPv6 address.

```
127.0.0.1
10.20.30.51
::1
2020:100:100:300::51
```

You can enter up to four entries of each attribute on separate lines in the `timesten.conf` file. In addition to the addresses you specify, TimesTen always listens on the loopback address for its own purposes.

Managing Subdaemons

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 2 subdaemons and specifies the default maximum number of subdaemons at 50. However, you can change these settings by assigning new values to the `min_subs` and `max_subs` attributes in the `timesten.conf` file.

Configuring a Range for Shared Memory Keys

You can configure a range for all shared memory keys used by TimesTen with the `shmkey_range` daemon attribute.

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:

```
shmkey_range=low-high
```

For example:

```
shmkey_range=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 values for the shared memory keys, then the TimesTen daemon terminates 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 Attributes

This section includes the following topics:

Modifying the TimesTen Server Attributes

The TimesTen server is a child process of the TimesTen daemon that operates continually in the background.

To modify the TimesTen server attributes, you must do the following:

1. Stop the TimesTen server.
2. Modify the attributes in the `timesten.conf` file as described in the following sections.
3. Restart the TimesTen server.

Controlling the TimesTen Server

The `server_port=`*`portno`* attribute in the `timesten.conf` 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.

Prespawning TimesTen Server Processes in TimesTen Classic

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

In TimesTen Classic, you can prespawn a pool of reserve server processes by setting the `server_pool` attribute in the `timesten.conf` file on the server system. These server processes are immediately available for a client connection, which improves client/server connection performance.

The `server_pool=number` attribute in the `timesten.conf` 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 server process. You can set multiple connections to a single TimesTen server, either by using the server connection attributes or by setting the TimesTen daemon attributes described in this section.

See Defining Server DSNs for TimesTen Server on a Linux or UNIX System or Server First Connection Attributes in *Oracle TimesTen In-Memory Database Reference*.

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 attributes, the value of the server connection attributes takes precedence.

The following sections describe how to configure multiple child server processes so that a single server process can service multiple client connections to a database.

- Configuring the Maximum Number of Client Connections Per Child Server Process
- Configuring Connection Distribution Among Child Server Processes
- Configuring the Thread Stack Size of the Child Server Processes

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 `max_conns_per_server` attribute to the `timesten.conf` file.

```
max_conns_per_server=NumberOfClientConnections
```

See Defining Server DSNs for TimesTen Server on a Linux or UNIX System.

Configuring Connection Distribution Among Child Server Processes

To specify the number of child server processes for a particular server DSN that will use round-robin connection distribution (when `max_conns_per_server >1`), add the `servers_per_dsn` attribute to the `timesten.conf` file.

```
servers_per_dsn=NumberOfChildServerProcesses
```

By default (value=1), the first `max_conns_per_server` client connection to a Server DSN is assigned to a single child server process, the next `max_conns_per_server connection` is assigned to a second child server process and so on. See Defining Server DSNs for TimesTen Server on a Linux or UNIX System for a full description of how the `servers_per_dsn` and `max_conns_per_server` attributes relate to each other to control how connections are distributed over server processes.

Configuring the Thread Stack Size of the Child Server Processes

To set the size of the child server process thread stack for each client connection, add the `server_stack_size` attribute to the `timesten.conf` file.

```
server_stack_size=ThreadStackSize
```

See Defining Server DSNs for TimesTen Server on a Linux or UNIX System.

> **Note:**
>
> These changes to the TimesTen server do not occur until the TimesTen daemon is restarted.

Controlling the TimesTen Server Log Messages

The `noserverlog` attribute in the `timesten.conf` 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, change the value of the `noserverlog` attribute to `1`.
- To disable logging, change the value of the `noserverlog` attribute to `0`.

Error, Warning, and 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, TimesTen messages and diagnostic information are stored in:

- A user error log that contains error message information. Generally, these messages contain information on actions you may need to take. The default file is *timesten_home*`/diag/tterrors.log`.

- A daemon log file containing everything in the user error log plus information used by TimesTen Customer Support. The default file is *timesten_home*/diag/ttmesg.log.
- An invalidation file containing diagnostic information when TimesTen invalidates a database. This file provides useful troubleshooting information for TimesTen Customer Support. The invalidation file is created and named based on the value specified by the DataStore connection attribute. This connection attribute is not a file name. For example on Linux and UNIX systems, if the DataStore connection attribute is /home/ttuser/AdminData, the actual invalidation file name has a suffix, .inval, /home/ttuser/AdminData.inval. See Critical Event Logging.
- Critical events log files contain a collection of daemon log entries at the moment of a critical event to assist when diagnosing critical failures. See Critical Event Logging.

You can specify the location and size of the daemon log files and user log files, as well as the number of files to keep stored on your system in the TimesTen instance configuration file (the timesten.conf file).

You can also specify the syslog facility used to log TimesTen daemon and subdaemon messages on Linux or UNIX. On a separate line of the timesten.conf file add:

```
facility=name
```

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

See TimesTen Instance Configuration File in the *Oracle TimesTen In-Memory Database Reference*.

The ttDaemonLog utility enables you to control the type of events that TimesTen writes to and fetches from the TimesTen user and error logs. You can also display all messages or selected categories of messages from the log to the standard output with this utility. See ttDaemonLog in the *Oracle TimesTen In-Memory Database Reference*.

Critical Event Logging

When critical events occur, TimesTen collects the daemon log entries at the moment of the critical event to assist when diagnosing critical failures.

An example of critical events include database failure or a log-based catch up failure. Database invalidation occurs when TimesTen detects that the data in the database is corrupt and therefore unusable.

- TimesTen generates critical event files containing a snippet of the daemon log file that was collected at the moment of the critical event. Critical event log files are created and stored in the same directory as the daemon log files (specified by supportlog configuration attribute in the timesten.conf file). Critical event log files are named with the format of ttmesg.log.ts_*timestamp*.gz where *timestamp* is the current time on the host.
- When a critical event occurs, TimesTen records in the daemon log file that critical event information was collected and the name of the critical event log file.
- If a database invalidation occurs, then TimesTen records in the .inval file that critical event information was collected at the time of the database invalidation and the name of the critical event log file. The .inval file is located in the same directory configured as the DataStore directory.

If more than one critical event occurs in quick succession, TimesTen collects only new information for each subsequent event in the next critical event log file.

You can configure the duration of the daemon log collection with the `daemon_log_snippet_interval_in_mins` configuration attribute in the `timesten.conf` file. As with all changes to the `timesten.conf` file, you must restart the main daemon for any change to take effect. See the TimesTen Instance Configuration File in the *Oracle TimesTen In-Memory Database Reference*.

The occurrence of a critical event and the name of the critical event log file are noted in the daemon log file. The following example shows the message you would see in the daemon log file if two critical events occur in quick succession.

```
17:08:00.224 Err : : 12532: A critical event has happened. Saving last 600 seconds
snippet of daemon log at /timesten/instance/diag/ttmesg.log.ts_1568160480.gz.
17:08:00.643 Err : : 12532: Read 543878 bytes from daemon log file and wrote into
 the daemon log snippet file located at /timesten/instance/diag/
ttmesg.log.ts_1568160480.gz

17:11:12.657 Err : : 12532: A critical event has happened. Saving last 600 seconds
 snippet of daemon log at /timesten/instance/diag/ttmesg.log.ts_1568160672.gz.
17:11:12.657 Err : : 12532: Part of the daemon log snippet has already been
 recorded in a prior snippet file ending with timestamp 1568160480.gz
17:11:13.077 Err : : 12532: Read 2678582 bytes from daemon log file and wrote
 into the daemon log snippet file located at
 /timesten/instance/diag/ttmesg.log.ts_1568160672.gz
```

After which, you can evaluate the collected log records in the specified `ttmesg.log.ts_`*`timestamp`*`.gz` files.

Any database invalidation event and the name of the critical event log file are noted in the `.inval` file. The following example shows the messages you could see in the `.inval` file if two critical events occur in quick succession.

```
2019-09-10 17:08:00.752
Hostname: myhost
Invalidated data store: /timesten/instance/datastores/mydb
Data store created: 2019/09/10 17:04:25
TimesTen Release 18.1.3.1.0 (Linux x86-64, 64-bit dbg) (myhost)
Data store created by Release 18.1.3.1.0 (Linux x86-64, 64-bit dbg) (myhost)
Source: Data store marked invalid by master daemon: grid LBCU pre-condition check failed
A critical event has happened. Saving last 600 seconds snippet of daemon log at
/timesten/instance/diag/ttmesg.log.ts_1568160480.gz.
Read 953765 bytes from daemon log file and wrote into the daemon log snippet file
located at /timesten/instance/diag/ttmesg.log.ts_1568160480.gz

2019-09-10 17:11:13.905
Hostname: myhost
Invalidated data store: /timesten/instance/datastores/mydb
Data store created: 2019/09/10 17:04:25
TimesTen Release 18.1.3.1.0 (Linux x86-64, 64-bit dbg)
Data store created by Release 18.1.3.1.0 (Linux x86-64, 64-bit dbg)
Source: Data store marked invalid by master daemon: subdaemon managing database
exited or died
A critical event has happened. Saving last 600 seconds snippet of daemon log at
/timesten/instance/diag/ttmesg.log.ts_1568160673.gz.
Part of the daemon log snippet has already been recorded in a prior snippet file
ending with timestamp ts_1568160480.gz
Read 781026 bytes from daemon log file and wrote into the daemon log snippet file
located at /timesten/instance/diag/ttmesg.log.ts_1568160673.gz
```

After which, you can evaluate the collected log records in the specified `ttmesg.log.ts_`*`timestamp`*`.gz` files.

5

Globalization Support

The following sections describe TimesTen globalization support features:

- Overview of Globalization Support Features
- Choosing a Database Character Set
- Character Set Length Semantics Affect Data Storage
- Setting the Connection Character Set
- Linguistic Sort Rules Support Linguistic Conventions
- SQL String and Character Functions
- Setting Globalization Support Attributes
- Globalization Support During Migration

Overview of Globalization Support Features

TimesTen globalization support includes database character set, length semantics, linguistic sorts and indexes, and SQL string and character functions.

- Character set support

  You must choose a database character set when you create a database. See Choosing a Database Character Set 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 Setting the Connection Character Set.

- Length semantics

  You can specify byte semantics or character semantics for defining the storage measurement of character data types. See Character Set Length Semantics Affect Data Storage.

- Linguistic sorts and indexes. You can sort data based on linguistic rules. See Linguistic Sort Rules Support Linguistic Conventions. You can use linguistic indexes to improve performance of linguistic sorts. See Using Linguistic Indexes.

- 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.

> **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 caching 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

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 usually 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 a database with a Japanese character set. 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.

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.

Character Set Length Semantics Affect 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 NCHAR and NVARCHAR2 character data types are encoded as UTF-16, which requires at least 2 bytes for each character. Thus, NCHAR(20) in a table definition allows 40 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.

Setting the 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 Sort Rules Support Linguistic Conventions

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

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`:

```
Command> ALTER SESSION SET NLS_SORT=SWEDISH;
Command> 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.

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.

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.

These SQL functions 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

You can set globalization support attributes.

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.

The rest of the attributes are set during connection to a database. For more information about `ConnectionCharacterSet`, see Setting the Connection Character Set.

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*.

Globalization Support During Migration

Globalization support may cause issues during migration.

For complete details, see Back Up, Restore, and Migrate Data in TimesTen Classic in *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide* and the description of ttMigrate in *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 issue 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 run various ODBC function calls can also be displayed.

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

For more information on `ttIsql` commands, see the ttIsql section in the *Oracle TimesTen In-Memory Database Reference*.

Using ttIsql in Batch Mode or 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 run 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 running 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 running the `connect` command with the `database1` DSN.

```
C:\>ttIsql

Copyright (c) 1996, 2024 Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

Command> connect database1;
Connection successful:
DSN=database1;DataStore=/disk1/databases/database1;DatabaseCharacterSet=AL32UTF8;
ConnectionCharacterSet=AL32UTF8;PermSize=128;
(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 run.

```
C:\>ttIsql -connstr "DSN=database1"

Copyright (c) 1996, 2024 Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

connect "DSN=database1";
Connection successful:
DSN=database1;DataStore=/disk1/databases/database1;DatabaseCharacterSet=AL32UTF8;
ConnectionCharacterSet=AL32UTF8;PermSize=128;
(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, running a file containing a `CREATE TABLE` statement looks like the following:

```
C:\>ttIsql -f create.sql -connstr "DSN=database1"

Copyright (c) 1996, 2024 Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

connect "DSN=database1";
Connection successful:
DSN=database1;DataStore=/disk1/databases/database1;DatabaseCharacterSet=AL32UTF8;
ConnectionCharacterSet=AL32UTF8;PermSize=128;
(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 run:

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

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

Customizing Startup Command Line Options for ttIsql Sessions

The `ttIsql` utility can be customized to automatically run 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=database1" -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, 2024 Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

Command> connect "DSN=database1";
Connection successful:
DSN=database1;DataStore=/disk1/databases/database1;DatabaseCharacterSet=AL32UTF8;
ConnectionCharacterSet=AL32UTF8;PermSize=128;
(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:

```
help attributes
```

Using the ttIsql 'editline' Feature for Linux and UNIX Only

On Linux and 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>	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).
<ESC-k> or <Ctrl-Up-Arrow>	Scroll up one line to edit within a multiple line PL/SQL block.
<ESC-j> or <Ctrl-Down-Arrow>	Scroll down one line to edit within a multiple line PL/SQL block.

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 run `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.

Keystroke	Action
i, I	Insert mode, Insert mode at beginning of the line.
a, A	Add ("Insert after") mode, Append at end of line
R	Replace mode.
C	Change to end of line.
B	Move to previous word.
e	Move to end of word.
<Ctrl-P>	Previous History (see Up Arrow).
<Ctrl-N>	Next History (see up Down Arrow).

Using the ttIsql Command History

The `ttIsql` utility stores a list of the last 100 commands that ran within the current `ttIsql` session. The commands in this list can be viewed or run 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 previous 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 that were run. 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 run the command again. For example:

```
Command>
Command> ! 12;

autocommit 0
Command>
```

To run the last command again simply type a sequence of two `!` characters:

```
Command> !!;

autocommit 0
Command>
```

To run 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.

```
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.

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

You can clear the list of commands that `ttIsql` stores by using the `clearhistory` 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`. See Changing the Default Text Editor for the ttIsql edit Command.

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:

```
edit new.sql;
```

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

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

If you run the `ttIsql edit` command with a `file` parameter, `ttIsql` does not run 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.

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

```
edit !2;
```

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

```
edit !create;
```

The following example runs 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 run the `ttIsql edit` command with a `history_search_command` parameter, `ttIsql` runs the contents of the file after you exit the editor. The contents of the file run as a single `ttIsql` command. If you do not want to run 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`.

```
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 Linux and UNIX and `notepad.exe` is used for Windows.

Displaying Characters in ttIsql

The ability of `ttIsql` to display characters depends on the native operating system locale settings of the terminal on which you are using `ttIsql`.

The `ttIsql` utility supports the character sets listed in Supported Character Sets in the *Oracle TimesTen In-Memory Database Reference*.

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 Structures Using ttIsql

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, 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: SALES.READTAB
 Table Type: Read Only
Cache Group CACHEUSER.WRITECACHE:
 Cache Group Type: Asynchronous Writethrough (Dynamic)
 Autorefresh: No
 Aging: LRU on
 Root Table: SALES.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.

However, for TimesTen Scaleout, the `dssize` command only reports on the current memory status for the current element.

The `dssize` command reports the same information that is displayed in the `SYS.V$MONITOR` and `SYS.GV$MONITOR` system views.

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:**
>
> See ttComputeTabSizes in the *Oracle TimesTen In-Memory Database Reference*.

Running 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-run `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, rollback operations 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:
```

Using ttIsql to List 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:**
>
> See ttIsql 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 database1
Copyright (c) 1996, 2024 Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.
connect "DSN=database1";
Connection successful:
DSN=database1;UID=terry;DataStore=/disk1/databases/
database1;DatabaseCharacterSet=AL32UTF8;
ConnectionCharacterSet=AL32UTF8;PermSize=128;
(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=database1;UID=PAT"
Copyright (c) 1996, 2024 Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.
connect "DSN=database1;UID=PAT";
Connection successful: DSN=database1;UID=PAT;
DataStore=/disk1/databases/database1;DatabaseCharacterSet=AL32UTF8;
ConnectionCharacterSet=AL32UTF8;PermSize=128;
(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.
```

Using ttIsql to View and Set Connection Attributes

You can view and set connection attributes with the `ttIsql show` and `set` commands.

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;
```

Using ttIsql to Manage 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 run 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.

To turn the autocommit feature off, run the `ttIsql autocommit` command with an argument of 0. When autocommit is turned off, transactions must be committed or rolled back manually by running 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 run time of a SQL statement exceeds the number of seconds set by the `sqlquerytimeout` command, the SQL statement does not run and an 6111 error is generated. 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 roll back and query timeout features do not stop 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, 2024 Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

Command> connect "DSN=database1";
Connection successful:
DSN=database1;DataStore=/disk1/databases/database1;DatabaseCharacterSet=AL32UTF8;
ConnectionCharacterSet=AL32UTF8;PermSize=128;
(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.
```

Using ttIsql with Prepared and Parameterized SQL Statements

Preparing a SQL statement just once and then running it multiple times is much more efficient for TimesTen applications than re-preparing the statement each time it is to be run. The `ttIsql` utility 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` - Runs a previously prepared statement. Corresponds to a `SQLExecute` ODBC call.
- `execandfetch` - Runs 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 run statement. Corresponds to one or more `SQLFetch` calls.
- `fetchone` - Fetches only one row for a previously run statement. Corresponds to exactly one `SQLFetch` call.
- `close` - Closes the result set cursor on a previously run 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 runtime.

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 runs 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 run the primary ODBC function call associated with each command.

```
Command> connect "DSN=database1";
Connection successful:
DSN=database1;DataStore=/disk1/databases/database1;DatabaseCharacterSet=AL32UTF8;
ConnectionCharacterSet=AL32UTF8;PermSize=128;
(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.
```

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 runs, 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. See ttIsql in the *Oracle TimesTen In-Memory Database Reference* or type `help` at the `ttIsql` command prompt.

The example below prepares and runs 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 runs. 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=database1";
Connection successful:
```

```
DSN=database1;DataStore=/disk1/databases/database1;DatabaseCharacterSet=AL32UTF8;
ConnectionCharacterSet=AL32UTF8;PermSize=128;
(Default setting AutoCommit=1)

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:**
>
> See Introduction to PL/SQL in TimesTen in *Oracle TimesTen In-Memory Database PL/SQL Developer's Guide*.

Using, Declaring, and Setting Variables in ttIsql

You can 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:**
>
> 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('LockLevel');

PARAMNAME, PARAMVALUE
< LockLevel, 0 >
1 row found.

Command> IF :paramvalue = 1 THEN "e:Database-level locking is enabled";
Command> IF NOT  :paramvalue = 1 THEN "e:Row-level locking is enabled";
Row-level locking is 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 Running PL/SQL Blocks Within ttIsql

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

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

```
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 PL/SQL Blocks in the *Oracle TimesTen In-Memory Database PL/SQL Developer's Guide*. 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 Within ttIsql

You can pass data back to applications from PL/SQL by using `OUT` parameters.

This example returns information about how full is a TimesTen database.

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 run 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.
```

Using the IF-THEN-ELSE Command Construct Within ttIsql

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 run commands within the `THEN` clause or the optional `ELSE` clause. The commands 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 which type of locking is enabled for the TimesTen database. 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.

```
Command> SET AUTOVARIABLES ON;
Command> CALL TTCONFIGURATION('LockLevel');
PARAMNAME, PARAMVALUE
< LockLevel, 0 >
1 row found.
Command> IF :paramvalue = 1 THEN "e:Database-level locking is enabled"
> ELSE "e:Row-level locking is enabled";
Row-level locking is enabled
```

The following example checks to see that the `employees` table exists. If it does not, it runs 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 Without Cache

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 group and cache table to contain the results. TimesTen provides tools that run 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 runs 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, runs 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.
- The `ttTableSchemaFromOraQueryGet` built-in procedure evaluates the user-provided `SELECT` statement to generate a `CREATE TABLE` statement that can be run to create a table on TimesTen, which would be appropriate to receive the result set from the `SELECT` statement. The `ttLoadFromOracle` built-in procedure runs 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.

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 run 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 run. 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, use the `numThreads` parameter on `ttIsql createandloadfromoraquery` or `ttLoadFromOracle` to specify the number of parallel threads that you would like to use in parallel when loading the table with the result set. This defaults to four.
- The SQL `SELECT` statement that runs 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 run 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, 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. However, the `ttLoadFromOracle` built-in procedure enables you to ignore constraint violations by setting the options parameter to `IgnoreDuplicates=Y`.

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
- Cancel a Parallel Load Operation

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 run 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 it runs 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*.

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 runs 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=database1;UID=hr;PWD=hr;OraclePWD=oracle"
Copyright (c) 1996, 2024 Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql
connect -connstr "DSN=database1;UID=hr;PWD=********;OraclePWD=********";
Connection successful: DSN=database1;UID=hr;
DataStore=/timesten/install/sample_db/DemoDataStore/database1;
DatabaseCharacterSet=AL32UTF8;
ConnectionCharacterSet=AL32UTF8;DRIVER=/timesten/install/lib/libtten.so;
PermSize=128;TempSize=64;OracleNetServiceName=inst1;

(Default setting AutoCommit=1)
Command> createandloadfromoraquery employees 2 SELECT * FROM hr.employees;
Mapping query to this table:
    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)
     )
Table employees created
< 107, 0, 0, Started=2015-09-11 21:12:45 (GMT); Ended=2015-09-11 21:12:46 (GMT);
Load successfully completed; OracleSCN=779534; Rows Loaded=107; Errors=0;
Statement=ttLoadFromOracle(HR, EMPLOYEES, SELECT * FROM hr.employees, 2) >
1 row found.
```

Issue 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 `hr`.

```
Command> DESCRIBE employees;

Table HR.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 `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 "HR"."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, 0, 0, Started=2015-09-11 21:30:56 (GMT); Ended=2015-09-11
21:30:57 (GMT);
Load successfully completed; OracleSCN=780073; Rows Loaded=100; Errors=0;
Statement=ttLoadFromOracle(HR, EMP, SELECT * FROM hr.employees
WHERE employee_id < 200, 4) >1 row found.
```

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 HR.EMP already exists
< 6, 0, 0, Started=2015-09-11 21:34:31 (GMT); Ended=2015-09-11 21:34:31 (GMT);
Load successfully completed; OracleSCN=780176; Rows Loaded=6; Errors=0;
Statement=ttLoadFromOracle(HR, EMP, SELECT * FROM hr.employees
WHERE employee_id > 200, 4) >
1 row found.
```

A parallel load operation may take a long time to run and you may want to cancel the operation. See Cancel a Parallel Load Operation.

Use TimesTen Built-In Procedures to Recommend a Table and Load SQL Query Results

You can create the TimesTen table and load the result set from the Oracle database into the table using built-in procedures.

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 issue. In order to run this statement, the user should have all required privileges to run the query on the Oracle database. This enables you to view the table structure without processing. However, it does require you to issue the recommended `CREATE TABLE` statement yourself.
2. The `ttLoadFromOracle` built-in procedure runs 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. Optionally, you can specify error

threshold options to avoid issues such as uniqueness violations and data conversion issues with the `IgnoreDuplicates` option.

> **Note:**
>
> See ttTableSchemaFromOraQueryGet and ttLoadFromOracle in the *Oracle TimesTen In-Memory Database Reference*.

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. Then, the `CREATE TABLE` statement runs. 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 roll back manually after loading the table.

```
% ttisql -connstr "DSN=database1;uid=hr;pwd=hr;OraclePWD=oracle"
Copyright (c) 1996, 2024 Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql
connect -connstr "DSN=database1;UID=hr;PWD=********;OraclePWD=********";
Connection successful: DSN=database1;UID=hr;
DataStore=/timesten/install/sample_db/DemoDataStore/database1;
DatabaseCharacterSet=AL32UTF8;
ConnectionCharacterSet=AL32UTF8;DRIVER=/timesten/install/lib/libtten.so;
PermSize=128;TempSize=64;OracleNetServiceName=inst1;

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> 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)
 );

Command> call ttLoadFromOracle ('HR','EMPLOYEES','SELECT * FROM HR.EMPLOYEES');
< 107, 0, 0, Started=2015-09-11 21:52:51 (GMT); Ended=2015-09-11 21:52:51 (GMT);
Load successfully completed; OracleSCN=780552; Rows Loaded=107; Errors=0;
Statement=ttLoadFromOracle(HR, EMPLOYEES, SELECT * FROM HR.EMPLOYEES, 4) >
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.

```
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));
```

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 ('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.
```

A parallel load operation may take a long time to run and you may want to cancel the operation. See Cancel a Parallel Load Operation.

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 (this applies for ODBC 2.5 and ODBC 3.5) or by pressing Ctrl-C in the `ttIsql` utility.

See TimesTen ODBC Support in the *Oracle TimesTen In-Memory Database C Developer's Guide* and Canceling ODBC Functions 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.

Using ttIsql to View and Change Query Optimizer Plans

You can 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 TimesTen for processing 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. See The TimesTen Query Optimizer.

- `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 runs 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

You can view commands and their explain plans.

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.

If you run 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, run 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.
- 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.

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 run 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
```

Using ttIsql to Manage 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 run 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 runtime associated with the primary ODBC function call corresponding to the `ttIsql` command.

For example, when running the `ttIsql connect` command several ODBC function calls run, 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=database1";
Connection successful:
DSN=database1;DataStore=/disk1/databases/database1;DatabaseCharacterSet=AL32UTF8;
ConnectionCharacterSet=AL32UTF8;PermSize=128;
(Default setting AutoCommit=1)
Execution time (SQLDriverConnect) = 1.2626 seconds.
Command>
```

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

When using the `timing` command to measure queries, the time required to run 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 running the query.

```
Command> timing 1;
Command> verbosity 0;
Command> SELECT * FROM t1;
Execution time (SQLExecute + FetchLoop) = 0.064210 seconds.
Command>
```

Specifying Error Recovery Within ttIsql

Issue 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 running 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 runs 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 run 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, run 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 (such as locks and result sets) that were acquired or opened by the transaction are released and freed.

When using SQL statements or supported APIs (such as ODBC and JDBC) to commit or rollback your transaction:

- When you commit the current transaction, updates made in the transaction are made available to concurrent transactions.
- When you rollback the current transaction, all updates made in the transaction are undone.

Read-only transactions hold minimal resources, so you do not need to commit every read operation. When running write operations, commit 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 the file system.

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

You can configure whether the application enables implicit commit behavior or requires explicit commit behavior for DML or DDL statements.

Transaction Autocommit Behavior

Autocommit configures whether TimesTen issues an implicit commit after DML or DDL statements.

Some database APIs (such as ODBC and JDBC) support autocommit, which is enabled by default when using those APIs. Other APIs (such as OCI) do not provide an autocommit feature.

When autocommit is on, the following behavior occurs:

- An implicit commit is issued immediately after a statement runs successfully.
- An implicit rollback is issued immediately after a statement fails, such as a primary key violation.
- If the statement generates a result set, the automatic commit is not issued until that result set and any other open result set in the transaction have been explicitly closed. Any statements processed while a result set is open is not committed until all result sets have been closed.

  Fetching all rows of a result set does not automatically close the result set. After the result set has been processed, the result set 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 result sets.

- 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 run 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 run 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, run `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

The TimesTen database issues an implicit `COMMIT` before and after any DDL statement. A durable commit is performed after the processing of each DDL statement. This behavior is the same as the Oracle database.

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`

There are certain things to keep into consideration:

- 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.

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.

- 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.
- 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.

The following table shows how TimesTen uses locks and transaction logs:

If	Then
Transaction is terminated successfully (committed)	• Transaction log is written to the file system (depending on durability connection attribute). See Transaction Consistency and Durability. • 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 result sets 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 result sets 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.
Application fails	• All outstanding transactions are rolled back.

Transaction Consistency and Durability

TimesTen 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 the file system.
  - For TimesTen Classic, a successful checkpoint operation makes all transactions that have been committed at the time of the checkpoint operation consistent and durable.
  - In TimesTen Scaleout, the data in your database is distributed across elements. Each element keeps its own checkpoint and transaction log files. As a result, the data stored in each element is independently durable.

Each data instance in a grid manages one element of a database. In the event of a failure, a data instance can automatically recover the data stored in its element from the checkpoint and transaction logs files while the remaining data instances continue to service applications. If the K-safety value is 2 or greater, then a failed element can be recovered from another element in its replica set.

- All transactions are logged to an in-memory transaction log buffer, which is written to the file system either with durable or nondurable transactions. See Durability Options.

> **Note:**
>
> Checkpointing and logging are further described in Checkpoint Operations and Transaction Logging.

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 commits or rolls back and then releases its locks.

The `Isolation` connection attribute sets the isolation level for a connection. 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 run 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 runs the same query multiple times within the same transaction, the commit of an update from another transaction may cause the results from the second read operation to retrieve different results. A phantom row can appear (in modified form) in two different read operations within the same transaction, as the result of an 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:

- Connect with the `Isolation` connection attribute set to 1. You can also modify this value with the `ALTER SESSION` SQL statement.

  ```
  ALTER SESSION SET ISOLATION_LEVEL=READ COMMITTED;
  ```

  See Isolation in the *Oracle TimesTen In-Memory Database Reference* or ALTER SESSION in *Oracle TimesTen In-Memory Database SQL Reference*.

- When using `ttIsql`, run `isolation 1` or `isolation READ_COMMITTED`.

- ODBC applications can run the `SQLSetConnectOption` ODBC function with the `SQL_TXN_ISOLATION` flag set to `SQL_TXN_READ_COMMITTED`. See Option Support for ODBC 2.5 SQLSetConnectOption and SQLGetConnectOption in the *Oracle TimesTen In-Memory Database C Developer's Guide*.

- JDBC applications can run the `setTransactionIsolation` JDBC method of the `Connection` object to `TRANSACTION_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. But read operations do not block other read operations.

  - If an initial transaction reads a row, a second transaction can also read the row.

  - A row that has been read by an initial transaction cannot be updated or deleted by a second transaction until the initial transaction commits or rolls back. However, the second transaction can insert rows that are not in the same range that are locked by the first transaction. For example, if the first transaction is reading rows 1 through 10, then the second transaction cannot update rows 1 through 10. However, the second transaction can insert into rows greater than 10.

  - A row that has been inserted, updated or deleted by an initial transaction cannot be accessed in any way by a second transaction until the initial transaction either commits or rolls back.

  Serializable isolation level provides for repeatable reads and increased isolation at the expense of decreased concurrency. A transaction that runs 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 isolation, do one of the following:

  - Connect with the `Isolation` connection attribute set to 0. You can also modify this value with the `ALTER SESSION` SQL statement.

    ```
    ALTER SESSION SET ISOLATION_LEVEL=SERIALIZABLE ;
    ```

See Isolation in the *Oracle TimesTen In-Memory Database Reference* or ALTER SESSION in *Oracle TimesTen In-Memory Database SQL Reference*.

- When using `ttIsql`, run `isolation 0` or `isolation SERIALIZABLE`.
- ODBC applications can run the `SQLSetConnectOption` ODBC function with the `SQL_TXN_ISOLATION` flag set to `SQL_TXN_SERIALIZABLE`.
- JDBC applications can run the `setTransactionIsolation` JDBC method of the `Connection` object to `TRANSACTION_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 rollback a specified transaction.

Locking Granularities

TimesTen supports row-level locks, table-level locks and database-level locks.

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.

> **Note:**
>
> When multiple locks have been obtained within the same transaction, the locks are released sequentially when the transaction ends.

- *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.
  - In TimesTen Classic, 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.

  - Run the `ttOptSetFlag` built-in 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*.

- *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, run 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*.

- *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 can provide 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 run 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.
  - In TimesTen Classic, 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. 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.

When running a workload of high lock-contention potential, consider setting the `LockWait` connection attribute to a smaller value for faster return of control to the application, or setting `LockWait` to a larger value to increase the successful lock grant ratio (with a risk of decreased throughput).

See LockWait in *Oracle TimesTen In-Memory Database Reference*.

Checkpoint Operations

A checkpoint operation synchronizes the current state of the TimesTen in-memory database with the state in the latest on disk checkpoint file.

A checkpoint operation writes any regions of the in-memory database that have been changed since the last checkpoint operation. Checkpoint operations alternate between two checkpoint files so that there is always the latest and latest – 1 checkpoint images available for recovery. Once the checkpoint completes, TimesTen purges the related transaction log files (which are now a part of the checkpoint) as they are no longer required.

By default, TimesTen automatically performs background checkpoints at regular intervals. Checkpointing may generate a large amount of I/O activity and have a long processing 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.

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 file system space that can be used for new transaction log files. If these files were never purged, they would eventually consume all available space in the transaction log files directory, causing database operations to fail due to log space exhaustion.

Usage of Checkpoint Files

TimesTen creates two checkpoint files for each database, 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.

- In TimesTen Classic, the database maintains one set of checkpoint and transaction log files.
- In TimesTen Scaleout, each element maintains its own independent set of checkpoint and transaction log files. See Understanding Distributed Transactions in TimesTen Scaleout in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

TimesTen uses the most recent checkpoint file and transaction log files to recover the database to its most recent transaction-consistent state after a database shutdown or system failure. (The most recent transaction log files are those written since the checkpoint was done.) If any errors occur during this process, or if the more recent checkpoint image is incomplete, then recovery restarts using the other checkpoint file. See Transaction Logging.

Types of Checkpoints

TimesTen uses the most recent 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.

TimesTen supports two types of database checkpoint operations:

- Fuzzy or Non-Blocking Checkpoints
- Blocking Checkpoints

Fuzzy or Non-Blocking Checkpoints

Fuzzy checkpoints, or non-blocking checkpoints, enable transactions to run against the database while the checkpoint is in progress.

Fuzzy checkpoints do not obtain locks, 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 checkpoint file and the transaction log to bring the database into its most recent transaction-consistent state.

Blocking Checkpoints

Blocking checkpoints obtain an exclusive lock on the database for a portion of the checkpoint operation, which blocks 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.

In TimesTen Classic, an application uses the `ttCkptBlocking` built-in procedure to request a blocking checkpoint. The actual checkpoint is delayed until the requesting transaction commits or rolls back. If a blocking 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

There is a default behavior for TimesTen checkpoints.

- TimesTen performs periodic fuzzy checkpoints in the background. You can modify this behavior. See Configuring or Turning Off Background Checkpointing.
- TimesTen performs a blocking checkpoint operation of a database just before the database is unloaded from memory. See Blocking Checkpoints.

The following sections describe how to manage checkpointing:

- Programmatically Performing a Checkpoint in TimesTen Classic
- 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 in TimesTen Classic

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 against a TimesTen Classic database, it can call the `ttCkpt` built-in procedure to request a fuzzy checkpoint or the `ttCkptBlocking` built-in procedure to request a blocking checkpoint. This is not recommended. See ttCkpt and ttCkptBlocking in the *Oracle TimesTen In-Memory Database Reference*.

Configuring or Turning Off Background Checkpointing

You can configure TimesTen to checkpoint either at a specific timed frequency or when the transaction log files contain a certain amount of data.

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.

In most cases, it is recommended to use `CkptLogVolume` over `CkptFrequency`, since `CkptFrequency` does not take into account the rate of database transactions. If both `CkptFrequency` and `CkptLogVolume` attributes are set with a value greater than 0, then a checkpoint is performed when either of the two conditions becomes true.

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

Select from the `SYS.GV$CKPT_HISTORY` or `SYS.V$CKPT_HISTORY` system views to display information on the most recent eight checkpoints and checkpoint attempts.

> **Note:**
>
> You can also use the `ttCkptHistory` built-in procedure to display this information.

For example, some of the information that you can confirm:

- The amount of time a checkpoint takes (subtract the completion time of the checkpoint from the start time).
- The type of the checkpoint: fuzzy, blocking, static or none. See Types of Checkpoints.
- The status of the checkpoint: in-progress, completed, or failed. When examining checkpoint history, verify whether any recent checkpoints failed (indicated with status of

FAILED). If a checkpoint has failed, the reason is displayed in the error or additional details columns.

- The initiator of the checkpoint. From a user-level application (including TimesTen utilities), from a background checkpoint request, or the managing subdaemon of the database.
- The reason why the checkpoint occurred. The most popular reasons are after database creation, after recovery, final checkpoint after shutdown, after the user runs a built-in procedure, or after a flush operation.
- The reason for a failure if a checkpoint fails.
- The amount of data (total number of bytes) written by a typical checkpoint.
- The checkpoint rate. See Setting the Checkpoint Rate for details on how to calculate the checkpoint rate from the data provided.
- The percentage of the checkpoint that has been completed. If there is an in-progress checkpoint, indicates the percentage of the checkpoint that has been completed.
- The number of actual transaction log files purged by this checkpoint. If this column displays a zero, this does not mean that no log records were purged within the transaction log file. Instead, log records can be purged continually within a transaction log file. This column displays the actual number of transaction log files purged to show when file system space is freed.

  There are times when TimesTen may not be able to purge some of the transaction files if there is a hold set on the transaction log. For example, this value may show when the checkpoint cannot purge transaction log files due to a long-running transaction or a replication bookmark located far back in the transaction log files.
- The bookmark name (the reason for the transaction log hold) that is preventing the removal of one or more transaction logs. This name can help identify why transaction log files are remaining on the file system longer than expected. See Log Holds by TimesTen Components or Operations for details on the different bookmark (transaction log hold) names.

  If this information does not provide enough context on why transaction logs are not purged, then you can also run the `ttXactAdmin` built-in procedure for more details on the transaction logs.

See SYS.GV$CKPT_HISTORY, SYS.V$CKPT_HISTORY, SYS.GV$LOG_HOLDS or SYS.V$LOG_HOLDS in the *Oracle TimesTen In-Memory Database System Tables and Views Reference*. See ttCkptHistory, ttLogHolds or ttXactAdmin in the *Oracle TimesTen In-Memory Database Reference*.

This example shows a checkpoint in progress:

```
% SELECT * FROM SYS.V$CKPT_HISTORY;

< 2019-02-05 16:56:34.169520, <NULL>,
Fuzzy           , In Progress     , User            ,
BuiltIn         , <NULL>,
0, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>,
<NULL>, <NULL>, <NULL>, 13, 6, 0, <NULL>, <NULL> >

< 2019-02-05 16:55:47.703199, 2019-02-05 16:55:48.188764,
Fuzzy           , Completed       , Checkpointer    ,
Background      , <NULL>,
1, 0, 8964304, 294, 33554432, 291, 5677288, 27, 1019512,
1065408, <NULL>, 5, 0, Checkpoint, <NULL> >

< 2019-02-05 16:54:47.106110, 2019-02-05 16:54:47.723379,
Static          , Completed       , Subdaemon       ,
```

```
FinalCkpt       , <NULL>,
0, 0, 8960328, 294, 33554432, 291, 5677288, 256, 33157172,
5321548, <NULL>, 4, 0, Checkpoint, <NULL> >

< 2019-02-05 16:54:41.633792, 2019-02-05 16:54:42.568469,
Blocking        , Completed       , User            ,
BuiltIn         , <NULL>,
1, 0, 8958160, 294, 33554432, 291, 5677288, 31, 1162112,
6604976, <NULL>, 3, 0, Checkpoint, <NULL> >

< 2019-02-05 16:54:37.438827, 2019-02-05 16:54:37.977301,
Static          , Completed       , User            ,
DbCreate        , <NULL>,
0, 0, 1611984, 93, 33554432, 92, 1853848, 93, 33554432,
1854052, <NULL>, 2, 0, Checkpoint, <NULL> >

< 2019-02-05 16:54:36.861728, 2019-02-05 16:54:37.438376,
Static          , Completed       , User            ,
DbCreate        , <NULL>,
1, 0, 1609936, 93, 33554432, 92, 1853848, 93, 33554432,
1854052, <NULL>, 1, 0, Checkpoint, <NULL> >
```

This example shows that an error occurred during the most recent checkpoint attempt, which was a user-initiated checkpoint:

```
% SELECT * FROM SYS.V$CKPT_HISTORY;

< 2019-02-05 16:57:14.476860, 2019-02-05 16:57:14.477957,
Fuzzy           , Failed , User           ,
BuiltIn         , 847,
1, <NULL>, <NULL>, 0, 0, 0, 0, 0, 0, 0, <NULL>, 7, 0, <NULL>,
Errors   1: TT0847: 16:57:14 (2019-02-05) >

< 2019-02-05 16:56:34.169520, 2019-02-05 16:56:59.715451,
Fuzzy           , Completed       , User            ,
BuiltIn         , <NULL>,
0, 0, 8966472, 294, 33554432, 291, 5677288, 5, 522000,
532928, <NULL>, 6, 0, Checkpoint, <NULL> >

< 2019-02-05 16:55:47.703199, 2019-02-05 16:55:48.188764,
Fuzzy           , Completed       , Checkpointer    ,
Background      , <NULL>,
1, 0, 8964304, 294, 33554432, 291, 5677288, 27, 1019512,
1065408, <NULL>, 5, 0, Checkpoint, <NULL> >

< 2019-02-05 16:54:47.106110, 2019-02-05 16:54:47.723379,
Static          , Completed       , Subdaemon       ,
FinalCkpt       , <NULL>,
0, 0, 8960328, 294, 33554432, 291, 5677288, 256, 33157172,
5321548, <NULL>, 4, 0, Checkpoint, <NULL> >

< 2019-02-05 16:54:41.633792, 2019-02-05 16:54:42.568469,
Blocking        , Completed       , User            ,
BuiltIn         , <NULL>,
1, 0, 8958160, 294, 33554432, 291, 5677288, 31, 1162112,
6604976, <NULL>, 3, 0, Checkpoint, <NULL> >

< 2019-02-05 16:54:37.438827, 2019-02-05 16:54:37.977301,
Static          , Completed       , User            ,
DbCreate        , <NULL>,
0, 0, 1611984, 93, 33554432, 92, 1853848, 93, 33554432,
1854052, <NULL>, 2, 0, Checkpoint, <NULL> >
```

```
< 2019-02-05 16:54:36.861728, 2019-02-05 16:54:37.438376,
Static          , Completed       , User            ,
DbCreate        , <NULL>,
1, 0, 1609936, 93, 33554432, 92, 1853848, 93, 33554432,
1854052, <NULL>, 1, 0, Checkpoint, <NULL> >
```

This example demonstrates the checkpoint history output. You use the `ttCkptHistory` built-in procedure to select specific columns from the checkpoint history:

```
% SELECT type, reason, bookmarkname, logsPurged FROM ttCkptHistory;;

< Fuzzy          , BuiltIn         , Oldest Transaction Undo, 0 >
< Static         , FinalCkpt       , Checkpoint, 6 >
< Blocking       , BuiltIn         , Checkpoint, 0 >
< Blocking       , BuiltIn         , Checkpoint, 0 >
< Blocking       , BuiltIn         , Checkpoint, 0 >
< Blocking       , BuiltIn         , Backup, 5 >
< Blocking       , BuiltIn         , Backup, 0 >
< Blocking       , BuiltIn         , Backup, 0 >
```

The output from this example shows that the oldest checkpoints (the last rows displayed) did not purge any transaction log files (indicated by a `0` in the `logsPurged` column) because there was a log hold set by an incremental backup. The backup caused a transaction log file accumulation. However, eventually, the log hold was removed and five transaction log files could be purged.

A checkpoint operation is started on the fourth row from the bottom. The checkpoint places a log hold that prevents transaction log files from being purged. Six transaction log files were purged by the final checkpoint (`FinalCkpt`) operation.

The most recent checkpoint shows that it is a fuzzy checkpoint with the log hold of `Oldest Transaction Undo`. This hold marks a point for the transaction that the checkpoint log purge operation cannot pass. If you see this message over several consecutive rows, then this may indicate a long running transaction that could be causing a transaction log file accumulation.

Setting the Checkpoint Rate

By default, there is no limit to the rate at which checkpoint data is written to the file system. You can use the `CkptRate` connection attribute or the `ttCkptConfig` built-in procedure to set the maximum rate at which background checkpoint data is written to the file system.

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*.

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.

Setting a rate too high can cause checkpoints to consume too much of the file system buffer cache bandwidth that could result in the following:

- Reduce overall database transaction throughput, as transaction logs are prevented from writing to the file system as quickly as they usually would.
- Cause contention with other file system I/O operations.

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 `SYS.GV$CKPT_HISTORY` or `SYS.V$CKPT_HISTORY` system views or the `ttCkptHistory` built-in procedure.

If a running checkpoint appears to be progressing too slowly, you evaluate the progress of this checkpoint with the `Percent_Complete` column of the `SYS.GV$CKPT_HISTORY` or `SYS.V$CKPT_HISTORY` system views. 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.

Calculate the checkpoint rate (by viewing the `SYS.GV$CKPT_HISTORY` or `SYS.V$CKPT_HISTORY` system views or calling the `ttCkptHistory` built-in procedure):

1. For any given checkpoint, subtract the *starttime* from the *endtime*.
2. Divide the number of bytes written by this elapsed time in seconds to get the number of bytes per second.
3. 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 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:**
>
> See Set CkptReadThreads in this book, and CkptReadThreads in the *Oracle TimesTen In-Memory Database Reference*.

Transaction Logging

TimesTen maintains a transaction log for each database to track all updates made within each transaction, so that those updates can be undone if the transaction is rolled back.

TimesTen recovers transactions using both the most recent checkpoint file and the most recent transaction log, which was written from the time of the last checkpoint operation after a system failure.

A transaction log record is created for each database modification, 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.

The transaction log is a logically ordered sequence of transaction log records, but physically consists of a set of one or more transaction log files and the in-memory log buffer within the TimesTen database. This log is shared by all concurrent connections.

The following sections describe how to manage and monitor the transaction log buffers and file:

- Managing Transaction Log Buffers and Files
- Using NFS-Mounted Systems for Checkpoint and Transaction Log Files
- Monitoring Accumulation of Transaction Log Files

Managing Transaction Log Buffers and Files

There are configuration options for 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 connection attribute.

  Strands divide the transaction log buffer available memory into a number of different regions, which can be accessed concurrently by different connections. The number of transaction log buffer strands is configured with the `LogBufParallelism` connection attribute. After which, each connection can run data independent DML statements in parallel using those strands as if each has its own transaction log buffer. See LogBufParallelism in the *Oracle TimesTen In-Memory Database Reference*.
- Transaction log files: The transaction log files are created in the location that the `LogDir` connection attribute specifies.

> **Note:**
>
> The `LogDir` connection attribute is optional and if it is not specified, the transaction log files are created in the same directory as the checkpoint files. However, for best performance, TimesTen recommends that you use the `LogDir` connection 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.
>
> If the database is already loaded into RAM and the transaction log files and checkpoint files for your database are on the same physical device, TimesTen writes a message to the daemon log file.
>
> See LogDir in the *Oracle TimesTen In-Memory Database Reference* and Check Transaction Log File Use of File System Space in the *Oracle TimesTen In-Memory Database Monitoring and Troubleshooting Guide*.

You can configure the maximum size for the transaction log files with the `LogFileSize` DSN connection attribute. See LogFileSize in the *Oracle TimesTen In-Memory Database Reference*.

The transaction log file names have the form *`dsname`*`.log`*`n`*. The *`dsname`* is the database path name that is specified by the `DataStore` DSN connection attribute and is provided within the database's DSN. The suffix *`n`* is the transaction log file number, starting at zero.

When the database is created, TimesTen creates reserve files named *`dsname`*`.res0`, *`dsname`*`.res1`, and *`dsname`*`.res2`. These files contain pre-allocated space that serves 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 that contains the transaction logs becomes full, the database does not allow any new transactions to begin. Transactions that are in progress are allowed to continue by using the reserve files until they either commit or rollback. If there are transactions that are still in progress and the reserve files are all consumed, all database operations that generate log files are blocked.

Using NFS-Mounted Systems for Checkpoint and Transaction Log Files

To enable TimesTen checkpoint and transaction log files to be located on NFS-mounted file systems (on Linux platforms platforms only), ensure that `allow_network_files=1` in the `timesten.conf` file. The `allow_network_files=1` configuration is the default setting.

See TimesTen Instance Configuration File in *Oracle TimesTen In-Memory Database Reference*.

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 file system 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:

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 standard 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 file system space.

These components and operations include the following:

- Transactions writing log records to the transaction log file have been committed or rolled back. These can be either local database transactions or XA transactions.
- Changes recorded in the transaction log file have been written to both checkpoint files.
- 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 standard operations resume. Criteria for when to do this includes the amount of time required to duplicate, the amount of available file system 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.

  See Improving Replication Performance in *Oracle TimesTen In-Memory Database Replication Guide* and Troubleshooting Replication in *Oracle TimesTen In-Memory Database Monitoring and 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.

  See Monitoring AWT Cache Groups in *Oracle TimesTen In-Memory Database Cache Guide* and Troubleshooting AWT Cache Groups in *Oracle TimesTen In-Memory Database Monitoring and 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.

  See Monitoring Cache Groups with Autorefresh in *Oracle TimesTen In-Memory Database Monitoring and Troubleshooting Guide*.

- Incremental TimesTen backup: There is a log hold until you manually run an incremental backup and until that backup completes.

  The most likely failure is if you configure incremental backups (which starts TimesTen holding log files), but you do not actually run an incremental backup, which would free the log file hold.

  Another 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 file system 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 commits or rolls back.

  A possible failure mode can occur if an application transaction does not commit or rollback for a long time, so that it becomes necessary for the application to terminate the long-running transaction.

  Alternatively, you can rollback a transaction using the `ttXactAdmin` utility with the `-xactIdRollback` option. See ttXactAdmin in *Oracle TimesTen In-Memory Database Reference*.

- Oldest transaction undo: Each transaction that has not yet committed or rolled back establishes a log hold. This hold starts with the first (oldest) log record of the transaction and marks a point for the transaction that the checkpoint log purge operation cannot pass. You can retrieve this log hold information from the `SYS.GV$CKPT_HISTORY` or `SYS.V$CKPT_HISTORY` system views or from the output of the `ttCkptHistory` built-in procedure.

Purging Transaction Log Files

Any transaction log file is kept until TimesTen determines it can be purged. Under standard 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 or a configurable log volume with the `CkptLogVolume` connection attribute. In TimesTen Classic, you can also initiate a checkpoint (either manually or in a background checkpointing application thread) with the `ttCkpt` built-in function.

If you are running out of file system space because of log files accumulating, use the `CkptLogVolume` connection attribute instead of the `CkptFrequency` connection attribute. In addition, you can tell if reclamation is blocked when frequently selecting from the `SYS.GV$LOG_HOLDS` or `SYS.V$LOG_HOLDS` system views or calling the `ttLogHolds` built-in procedure.

> **Note:**
>
> To improve performance, locate your log files on a separate physical device from the one on which the checkpoint files are located. See Managing Transaction Log Buffers and Files.

See Checkpointing in the *Oracle TimesTen In-Memory Database Introduction* for general information on checkpointing. See Configuring or Turning Off Background Checkpointing for more details on `CkptFrequency` and `CkptLogVolume` connection attributes.

For more information on log holds, see SYS.GV$LOG_HOLDS or SYS.V$LOG_HOLDS in the *Oracle TimesTen In-Memory Database SQL Reference* or ttLogHolds in the *Oracle TimesTen In-Memory Database Reference*.

See CkptFrequency, CkptLogVolume, and ttCkpt in the *Oracle TimesTen In-Memory Database Reference*.

Monitoring Log Holds and Log File Accumulation

There are options for periodic monitoring of excessive transaction log accumulation.

- For details of all log holds, select from the `SYS.GV$LOG_HOLDS` or `SYS.V$LOG_HOLDS` system views or call the `ttLogHolds` built-in procedure. 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

    See SYS.GV$LOG_HOLDS or SYS.V$LOG_HOLDS in the *Oracle TimesTen In-Memory Database SQL Reference* or ttLogHolds in the *Oracle TimesTen In-Memory Database Reference*.

- Select from the `SYS.GV$CKPT_HISTORY` or `SYS.V$CKPT_HISTORY` system views or call the `ttCkptHistory` built-in procedure to check the last several checkpoints to confirm none of the returned rows has a status of `FAILED`.

  See SYS.GV$CKPT_HISTORY or SYS.V$CKPT_HISTORY in the *Oracle TimesTen In-Memory Database SQL Reference* or 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.

  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.

  See SYS.TRANSACTION_LOG_API in the *Oracle TimesTen In-Memory Database System Tables and Views Reference*.

Durability Options

Databases in TimesTen are persistent across power failures and crashes. TimesTen accomplishes durability by periodically saving to disk.

- Checkpoint files: A checkpoint operation writes the current database image to a checkpoint file on the file system, 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 the file system. 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 the file system. A durably committed transaction is not lost even in the event of a system failure.

Any recovery uses the last checkpoint image together with the transaction log to reconstruct the latest transaction-consistent state of the database.

When committing transactions, TimesTen reduces the performance cost for durable commits with a group commit of multiple concurrently running transactions. TimesTen performs a single file system write operation to commit 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 the file system write operation to complete, but it can significantly improve the throughput of a series of concurrent transactions.

The following sections describe the durability implementation options for TimesTen Scaleout and TimesTen Classic:

- Durability for TimesTen Scaleout
- Durability for TimesTen Classic

Durability for TimesTen Scaleout

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

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

TimesTen Scaleout also enables you to keep multiple copies of your data to increase durability and fault tolerance. If the K-safety value is 2 or greater, then a failed element can be recovered from another element in its replica set. For more information on K-safety, see High Availability and Fault Tolerance and Creating a Grid in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

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

- Guaranteed Durability for TimesTen Scaleout
- Non-Durable Distributed Transactions for TimesTen Scaleout

Guaranteed Durability for TimesTen Scaleout

In TimesTen Scaleout, you configure how durable your transactions are with the `Durablity` connection attribute.

If you set the `Durability` attribute to `1`, participants write durable prepare-to-commit log records and nondurable commit log records for distributed transactions. Having the `Durability` attribute set `1` ensures that committed transactions are recoverable in the case of a failure. See Durability Settings in *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

Non-Durable Distributed Transactions for TimesTen Scaleout

Set the `Durability` connection attribute to 0 so that participants write nondurable prepare-to-commit and commit log records for distributed transactions.

K-safety provides a method for failure recovery. K-safety enables you to achieve near-continuous availability or workload distribution by providing two copies of the data. Thus, if one copy fails, another copy of the data exists.

If any element of a replica set has failed the other element in the replica set writes durable prepare-to-commit log records until the failed element recovers. This ensures that transactions in a grid with K-safety set to 2 and `Durability`=0 are durable under standard conditions. A transaction may become nondurable only if both elements of the replica set fail simultaneously.

TimesTen Scaleout periodically promotes transactions to epoch transactions. An epoch transaction and all transactions committed before it are more resilient to catastrophic failures,

since you can recover a database to the globally consistent point marked by the epoch transaction. Each epoch commit log record is associated to a specific checkpoint file on every element. In the case of an unexpected failure of an element, the recovery process must use the checkpoint file on each element that is associated with the latest epoch commit log record, which is not necessarily the latest checkpoint available on the element.

By default, TimesTen Scaleout generates one epoch transaction every second. You can configure how often epoch transactions are automatically generated with the `EpochInterval` first connection attribute. In addition, you can use the `ttEpochCreate` or `ttDurableCommit` built-in procedures to manually promote a transaction to an epoch transaction.

See Durability Set to 0 in the *Oracle TimesTen In-Memory Database Scaleout User's Guide* for more information on epoch transactions and how to promote a transaction to an epoch transaction.

Durability for TimesTen Classic

TimesTen Classic provides durability by using the last checkpoint image together with the transaction log to reconstruct the latest transaction-consistent state of the database.

Transaction log records are written to disk asynchronously or synchronously to the completion of the transaction and controlled by the application at the transaction level.

- You can create an environment for guaranteed durability with no loss of data for those cases where data integrity must be preserved.
- You can create an environment for durability for those cases where maximum throughput is paramount but can also tolerate minimal exposure to some data loss.

TimesTen Classic replication makes data highly available to applications with minimal performance impact. Replication enables you to achieve near-continuous availability or workload distribution by sending updates between two or more hosts. A master host is configured to asynchronously send updates and a subscriber host is configured to receive them.

> **Note:**
>
> For the highest availability for TimesTen Classic, use the active standby pair replication scheme configuration. See Active Standby Pair With Read-Only Subscribers in the *Oracle TimesTen In-Memory Database Replication Guide*.

With an Asynchronous WriteThrough (AWT) cache group, TimesTen transmits committed updates between the TimesTen cache tables and the cached Oracle Database tables to keep these tables in the two databases synchronized. See Asynchronous WriteThrough (AWT) Cache Group in the *Oracle TimesTen In-Memory Database Cache Guide*.

- Guaranteed Durability for TimesTen Classic
- Non-Durable Transactions for TimesTen Classic

Guaranteed Durability for TimesTen Classic

Guaranteed durability ensures that there is no loss of data (at the cost of some performance) for those cases where data integrity must be preserved.

To guarantee durability for all transactions, you should do at least one of the following. (There is very little advantage in using both of these options.)

- Use replication enabled with the return twosafe service to replicate transactions. The return twosafe service provides a fully synchronous option by blocking the application until replication confirms that the update has been both received and committed on the subscriber before it is committed on the master. This provides applications a higher level of confidence that the effects of a committed transaction are durable, even if the master subsequently fails. This option is usually faster than the durable commit. See Return Twosafe Replication in the *Oracle TimesTen In-Memory Database Replication Guide*.
- Configure durable commit operations by setting the `DurableCommits` connection attribute to 1, which ensures that the transaction is synchronously written to the transaction log file while the application waits. See DurableCommits in the *Oracle TimesTen In-Memory Database Reference*.

> **Note:**
>
> If most of your transactions commit durably, you may want to set the `LogFlushMethod` first connection attribute to 1. This connection attribute configures how TimesTen writes and synchronizes log data to transaction log files. For more information, see Use Durable Commits Appropriately.

Non-Durable Transactions for TimesTen Classic

Non-durable transactions run considerably faster than durable transactions. Connections that use non-durable transactions can coexist with connections that use guaranteed durability.

As with guaranteed durability, each transaction enters records into the in-memory transaction log buffer as it makes modifications to the database. However, when a transaction commits in non-durable mode, it does not wait for the transaction log buffer to be written to the file system before returning control to the application. Thus, unless you have configured asynchronous replication or an asynchronous writethrough cache group, a non-durable transaction may be lost in the event of a database failure. Eventually, transactions are flushed to the file system into the transaction log files by the database subdaemon process or when the in-memory transaction log buffer is full.

If your transactions require extremely high throughput with minimal exposure to data loss, then consider doing BOTH of these options:

- Use one of these methods to perform an asynchronous write operation of your data to another database: a TimesTen database or an Oracle database.
  - Use the default asynchronous replication that replicates committed transactions from the master to the subscriber. With the default asynchronous replication, an application updates a master database and continues working without waiting for the updates to be received and applied by the subscribers. While asynchronous replication provides the best performance, it does not provide the application with confirmation that the replicated updates were committed on subscriber databases. See Copying Updates Between Databases *Oracle TimesTen In-Memory Database Replication Guide*.
  - Set up an Asynchronous WriteThrough (AWT) cache group to cause committed updates on TimesTen cache tables to be automatically and asynchronously propagated to the cached Oracle Database tables. A transaction commit operation on the TimesTen database occurs asynchronously from the commit on the Oracle database. This enables an application to continue issuing transactions on the

TimesTen database without waiting for the Oracle Database transaction to complete. However, your application cannot ensure when the transactions are completed on the Oracle database. See Asynchronous WriteThrough (AWT) Cache Group in the *Oracle TimesTen In-Memory Database Cache Guide*.

- Configure for non-durable transactions by setting the `DurableCommits` connection attribute to 0, which asynchronously writes non-durable commit log records to the transaction log file.

  See DurableCommits in the *Oracle TimesTen In-Memory Database Reference*.

The only transactions vulnerable to loss in the event of a system failure are those that committed non-durably after the last durable commit and were not replicated prior to the failure.

However, if you have critical transactions that cannot be exposed to any data loss, you can either:

- Replicate those transactions synchronously from the master database to the subscriber database by running the `ttRepSubscriberWait` built-in procedure.

  Running the `ttRepSubscriberWait` built-in procedure on the master database using the DSN and host of the subscriber database causes the caller to wait until all transactions that committed before the call have been transmitted to the subscriber. See ttRepSubscriberWait in the *Oracle TimesTen In-Memory Database Reference*.

- Force the individual transactions to commit durably to the transaction log files with the `ttDurableCommit` built-in procedure. Committing a transaction durably makes that transaction and all previous transactions durable. However, this does not wait for the transaction to be replicated.

  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.

  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.

  See ttDurableCommit in the *Oracle TimesTen In-Memory Database Reference*.

Transaction Reclaim Operations

After a transaction is marked by TimesTen as committed, there is a *reclaim* phase of the commit.

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 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 files. 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.

> **Note:**
>
> - The reclaim phase occurs as part of commit processing. Thus, once the reclaim phase has begun, the transaction is considered to be committed and can no longer be rolled back.
> - If an application is terminated during the reclaim phase, the cleanup operation completes 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.

> **Note:**
>
> It is not recommended that you perform an extremely large transaction commit (such as 1 million rows) as it is very resource intensive. In case that you are committing an extremely large transaction, do not cancel this transaction as it could cause the transaction to take even longer to complete. This is due to the fact that if you cancel a transaction while it has not completed, TimesTen has to perform rollback 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. 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. 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.

  In TimesTen Classic, you can use the `ttCommitBufferStatsReset` built-in procedure to reset 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.

  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. 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 the file system 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. See Checkpoint Operations, Transaction Logging and Error, Warning, and Informational Messages.

8

Working with Data in a TimesTen Database

There is detailed information on the basic objects in a TimesTen database and simple examples of how you can use SQL to manage these objects.

For more information about SQL, see Data Types in the *Oracle TimesTen In-Memory Database SQL Reference*.

For information on how to issue SQL from within an application, see the appropriate TimesTen developer's guide.

This chapter includes the following topics:

Database Objects, Users, and Owners

A TimesTen database has several permanent and temporary objects. TimesTen authenticates users with passwords. By default, the login name that is being used to run the application becomes the owner of objects in the database.

The following sections describe the main TimesTen database objects, users, and owners:

Database Objects

A TimesTen database has several permanent objects.

- **Tables**. The primary objects of a TimesTen database are the tables that contain the application data. See Understanding Tables.
- **Materialized Views**. Read-only tables that hold a summary of data selected from one or more "regular" TimesTen tables. See Understanding Materialized Views.
- **Views**. Logical tables that are based on one or more tables called *detail tables*. A view itself contains no data. See Understanding Views.
- **Indexes**. Indexes on one or more columns of a table may be created for faster access to tables. See Understanding Indexes.

- **Rows**. Every table consists of 0 or more rows. A row is a formatted list of values. See Understanding Rows.
- **Synonyms**. Aliases for a database object. Synonyms are often used for security and convenience, because they can be used to mask object name and object owner. See Understanding Synonyms.
- **System tables and views**. System tables and views contain TimesTen metadata, such as a table of all tables. See System Tables and Understanding System Views.

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

Database Users and Owners

TimesTen authenticates user names 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 objects in the database.

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. Ensure that you use the fully qualified name of a database object, for example *`owner.table_name`*, whenever you issue a SQL statement.

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.

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 an Aging Policy 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
- Tables in TimesTen Scaleout
- 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, or the `DELETE FIRST` clause. For more information, see Avoid Large DELETE Statements.

The maximum sizes of inline and out-of-line portions of a row are listed in Using the ttIsql tablesize Command.

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 for one or more columns, those default values for those columns are 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 or TTREP.

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*.

Table Access

Applications access tables through SQL statements. The TimesTen query optimizer automatically chooses the optimal 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. 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.

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 must be declared NOT NULL. 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.

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*.

> **Note:**
>
> Columns of a primary key must be `NOT NULL`; however, a unique index can be built on columns declared as `NOT NULL`.

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*.

Tables in TimesTen Scaleout

In TimesTen Scaleout, tables are the objects used to define how to distribute data in your database.

TimesTen Scaleout manages the distribution of data according to the defined distribution scheme of the table. When you create the table, it exists on every element of the database. Rows of data in the table exist on different elements of the database. See Working with Tables in *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

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 directly update system tables. TimesTen system tables are described in the System Tables and Views chapter in the *Oracle TimesTen In-Memory Database System Tables and Views Reference*.

> **Note:**
>
> TimesTen system table formats may change between releases of TimesTen.

Working with Tables

To perform any operation that creates, drops or manages a table, the user must have the appropriate privileges.

See 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 SQL Statements in the *Oracle TimesTen In-Memory Database SQL Reference*. TimesTen converts table names to upper case characters.

The following SQL statement example creates a table, called `customer`, with two columns: `cust_id` and `cust_name` of two different data types.

```
Command> CREATE TABLE customer (cust_id TT_INTEGER, cust_name VARCHAR2(50));
```

This example creates a table, called `customer`, with a hash index. The customer table has columns: `cust_id`, `cust_name`, `addr`, `zip`, and `region`. The `cust_id` 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.

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.

See `SET PAGES` in the ALTER TABLE section in the *Oracle TimesTen In-Memory Database SQL Reference*. See Size Hash Indexes Appropriately.

```
Command> 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;
```

Dropping a Table

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

The following example drops the table `customer`.

```
Command> DROP TABLE customer;
```

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:

 Index PAT adds 6282 bytes
  Total index bytes = 6282

Total = 23806
```

You can also calculate the size of an existing table with the `ttIsql tablesize` command. See Using the ttIsql tablesize Command.

Implementing an Aging Policy 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 only one type of aging for a specified 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 of materialized views

You can also implement aging in cache groups. See Implementing Aging in a Cache Group for TimesTen Classic in *Oracle TimesTen In-Memory Database Cache Guide*.

> **Note:**
>
> There is no support to specify aging policies for tables within TimesTen Scaleout. See Comparison Between TimesTen Scaleout and TimesTen Classic in *Oracle TimesTen In-Memory Database Scaleout 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
- Determining the Effectiveness of Aging

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`.

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. See Scheduling When Aging Starts.

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 inserted.
- 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.

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.

There are two LRU aging policies:

- LRU aging based on set thresholds for the amount of permanent memory in use. This is the default. Once you create (or alter) a table to use LRU aging, the LRU aging policy initially uses the default thresholds for permanent memory in use. See Defining LRU Aging Based on Thresholds for Permanent Memory in Use.
- LRU aging based on row thresholds for a set of tables. See Defining LRU Aging Based on Row Thresholds for Tables.

Both types of LRU aging can co-exist. Row threshold based aging takes precedence over permanent memory in use based aging. For each cycle after row based aging is processed, the current permanent space usage is used as the threshold for LRU aging of tables that use permanent memory in use threshold aging.

Defining LRU Aging Based on Thresholds for Permanent Memory in Use

You can define that LRU aging occurs based on set thresholds for the amount of permanent memory in use.

This method uses the `PERM_IN_USE_SIZE` metric to determine the thresholds for LRU aging across all tables. If the high threshold is reached, TimesTen deletes rows from all tables in order to reduce the permanent memory in use.

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.

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`*.

Defining LRU Aging Based on Row Thresholds for Tables

After a table or cache group is enabled for LRU aging, you can use the `ttAgingTableLRUConfig` built-in procedure to configure that LRU aging occurs based on row thresholds for a set of tables.

In some situations, the permanent memory in use based aging may result in excessive purging of data from specific tables. You may find that using row based aging for such tables enables a better space management to be achieved. When you set LRU aging using the `ttAgingTableLRUConfig` built-in procedure, it sets thresholds based on rows for specified tables and on cache instances when applied to specified cache groups. You can limit the number of rows deleted from a specified table by setting row thresholds for the table.

Both types of LRU aging can co-exist. Table-based aging takes precedence over permanent memory in use based aging. Thus, a table configured with the attributes specified with the `ttAgingTableLRUConfig` built-in procedure is aged solely based on the specified row thresholds. For each cycle after row based aging is processed, the current permanent space usage is used as the threshold for LRU aging of tables that use permanent memory in use threshold aging.

The following table summarizes the LRU aging attributes:

If you have tables or cache group tables in a parent/child relationship, provide the parent table to ensure that cascade delete operations perform correctly.

LRU Aging Attribute	Description
`tblOwner`	TimesTen table or cache group owner.
`tblName`	Name of a table. Provide the cache root table for a cache group. LRU aging automatically deletes rows from child tables.
`LowRowsThreshhold`	The number of the rows at which LRU aging is deactivated. LRU aging stops when the number of rows reaches `LowRowsThreshold` rows in the table.
`HighRowsThreshhold`	The number of the rows at which LRU aging is activated. LRU aging starts when the number of rows reaches `HighRowsThreshold` rows in the table.
`AgingCycle`	The number of minutes between aging cycles. The defaults and behavior is the same as the `AgingCycle` in the `ttAgingLRUConfig` built-in procedure.

If both `LowRowsThreshold` and `HighRowsThreshold` are set to zero, table based LRU aging is disabled for this table. LRU aging on this table is switched to be based on the permanent memory in use threshold. See Defining LRU Aging Based on Thresholds for Permanent Memory in Use.

See ttAgingTableLRUConfig in the *Oracle TimesTen In-Memory Database Reference*.

The following example sets the aging threshold for rows in the `user1.table1` table to a low threshold of 10K rows and a high threshold of 100K rows. The aging cycle is set to run at the default of once every minute.

```
Command> Call ttAgingTableLRUConfig('user1', 'table1', 10000, 100000);
< USER1, TABLE1, 10000, 100000, 1 >
1 row found.
```

The following example sets the aging threshold for rows in the `user1.table1` table to a low threshold of 5K rows and a high threshold of 12K rows. The aging cycle is set to run once every two minutes.

```
Command> Call ttAgingTableLRUConfig('user1', 'table1', 5000, 12000, 2);
< USER1, TABLE1, 10000, 100000, 0 >
1 row found.
```

The following example turns off aging by setting the low and high thresholds to zero.

```
Command> Call ttAgingTableLRUConfig('user1', 'table1', 0, 0);
< USER1, TABLE1, 0, 0, 1 >
1 row found.
```

You can retrieve the threshold settings by running the built-in procedure with just the schema owner and table name.

```
Command> Call ttAgingTableLRUConfig('user1', 'table1');
< USER1, TABLE1, 10000, 100000, 1 >
1 row found.
```

Time-Based Aging

Time-based aging removes data from a table based on the specified data lifetime and frequency of the aging process.

You can 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. See Scheduling When Aging Starts.

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.

There are two ways to define an LRU aging policy for your table:

- Define the LRU aging policy at the database level based on the percentage of the permanent memory in use. See Defining LRU Aging Based on Thresholds for Permanent Memory in Use.
- Define the LRU aging policy at the table level based on table rows or, in the case of cache groups, cache instances. See Defining LRU Aging Based on Row Thresholds for Tables.

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 multi-master 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.

Determining the Effectiveness of Aging

As aging deletes rows, TimesTen frees empty pages and reuses empty slots on non-full pages. The `ttPageLevelTableInfo` built-in procedure shows the page allocation for each table to determine when TimesTen is reusing empty slots and freeing empty pages or if new pages are allocated to store new rows.

The following demonstrates the output received for `user1.table1` and `user2.table2` using the `ttPageLevelTableInfo` built-in procedure:

```
Command> vertical 1;
Command> call ttPageLevelTableInfo;

  TBLOWNER:                 USER1
  TBLNAME:                  TABLE1
  CARD:                     6
  LOGICAL_PGCNT:            1
  LOGICAL_NONFULLPAGES:     1
  LOGICAL_FREESLOTS:        250
  PHYSICAL_PGCNT:           2
  PHYSICAL_NONFULLPAGES0:   0
  PHYSICAL_NONFULLPAGES1:   0
  PHYSICAL_NONFULLPAGES2:   1
  PHYSICAL_NONFULLPAGES3:   1
  PHYSICAL_FREESLOTS0:      0
  PHYSICAL_FREESLOTS1:      0
  PHYSICAL_FREESLOTS2:      253
  PHYSICAL_FREESLOTS3:      253


  TBLOWNER:                 USER2
  TBLNAME:                  TABLE2
  CARD:                     3
  LOGICAL_PGCNT:            1
  LOGICAL_NONFULLPAGES:     1
  LOGICAL_FREESLOTS:        253
  PHYSICAL_PGCNT:           1
  PHYSICAL_NONFULLPAGES0:   0
  PHYSICAL_NONFULLPAGES1:   1
  PHYSICAL_NONFULLPAGES2:   0
  PHYSICAL_NONFULLPAGES3:   0
  PHYSICAL_FREESLOTS0:      0
  PHYSICAL_FREESLOTS1:      253
  PHYSICAL_FREESLOTS2:      0
  PHYSICAL_FREESLOTS3:      0

2 rows found.
```

See ttPageLevelTableInfo in the *Oracle TimesTen In-Memory Database Reference*.

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.

A view is basically a SQL statement that is stored in a database which enables you to treat the results of a SQL query as a table itself. TimesTen generates the results of a view every time you query that view, so a view always has the latest data results. You can query a view just like you would query a regular table.

A materialized view enables you to precompute expensive SQL operations. There are cases where primarily read-only databases can increase the speed of their `SELECT` operations by precomputing the results of queries and storing them in a materialized view. The downside in this case is that the materialized view requires additional storage space. Keep in mind that when you perform DML operations on the detail tables of a materialized view, these DML operations are more resource intensive and more storage is required to store the materialized view rows. If you use too many materialized views on a table, this may reduce the performance of that table.

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

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`:

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

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

```
Command> 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.

The `SELECT` query used to define the contents of a matierialized view has 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 a processed statement.

See SQL Statements in the *Oracle TimesTen In-Memory Database SQL Reference*.

Dropping a View

The DROP VIEW statement deletes the specified view.

The following statement drops the cust_order view.

```
Command> DROP VIEW cust_order;
```

Restrictions on Views and Detail Tables

There are certain restrictions on views and their detail tables.

- When a view is referenced in the FROM clause of a SELECT statement, its name is replaced by its definition as a derived table during SQL compilation. The content of the derived table is said to be **materialized**, but this materialization is temporary and only exists for the duration of the SQL statement. 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

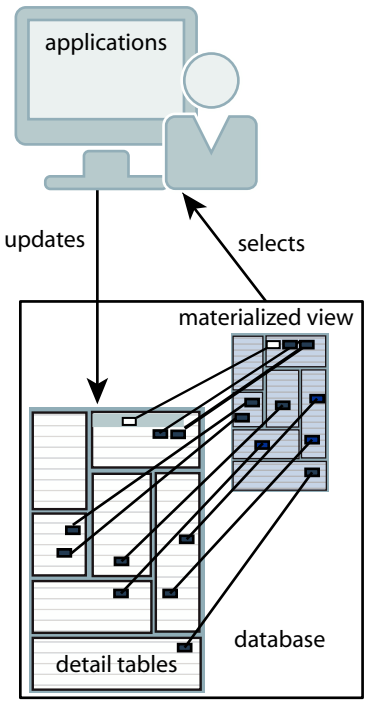
Overview of Materialized Views

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



The synchronous materialized view, 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 materialized view is never out of sync with the detail tables. However, this can affect your performance. A single transaction, the user transaction, runs the updates for both the detail table and any materialized views.

Working with Materialized Views

This section includes the following topics:

- Creating a Materialized View
- Dropping a Materialized View
- 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 in the *Oracle TimesTen In-Memory Database Security Guide*.

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.

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*.

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

The following example creates a synchronous materialized view, named `sample_mv`, that generates a result set from selected columns in the `customer` and `book_order` detail tables described above.

```
Command> CREATE TABLE customer
(cust_id INT NOT NULL,
cus_name CHAR(100) NOT NULL,
addr CHAR(100),
zip INT,
region CHAR(10),
PRIMARY KEY (cust_id));

CREATE TABLE book_order
(order_id INT NOT NULL,
cust_id INT NOT NULL,
book CHAR(100),
PRIMARY KEY (order_id),
FOREIGN KEY (cust_id) REFERENCES customer(cust_id));

Command> CREATE MATERIALIZED VIEW sample_mv AS
SELECT customer.cust_id, cust_name, order_id, book
FROM customer, book_order
WHERE customer.cust_id=book_order.cust_id;
```

Dropping a Materialized View

To drop any materialized view, issue the `DROP VIEW` statement.

The following statement drops the `sample_mv` materialized view.

```
Command> DROP VIEW sample_mv;
```

See SQL Statements chapter in the *Oracle TimesTen In-Memory Database SQL Reference*.

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, nor can it be updated 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 materialized 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.
- You cannot create a foreign key to reference a table or another materialized view. Regular tables cannot use a foreign key to refer to a 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 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*.

Performance Implications of Materialized Views

The performance of `UPDATE`, `INSERT`, and `DELETE` 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 materialized view is a persistent, up-to-date copy of a query result. To keep the materialized view up to date, TimesTen must perform "materialized view maintenance" when you change a materialized view's detail table. For example, if you have a materialized 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 "materialized view maintenance."

Materialized view maintenance needs appropriate indexes just like regular database operations. If they are not there, materialized view maintenance performs poorly.

All update, insert, or delete statements on detail tables have execution plans, as described in The TimesTen Query Optimizer. For example, an update of a row in `T1` initiates the first stage of the plan where it updates the materialized view `V`, followed by a second stage where it updates T1.

For fast materialized 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 materialized view should select `K1`, `K2`, and `K3`.

3. Create an index on the materialized 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 run 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 materialized view, taking up more space or more performance than you might wish. An alternative method is as follows:

1. For each table in the materialized view's `FROM` clause (each detail table), check which ones are frequently changed by `UPDATE`, `INSERT` and `CREATE VIEW` statements. For example, a materialized 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 materialized view selects its rowids. In this example, the materialized view should select `T2`.rowid and `T3`.rowid.

3. Create an index on the materialized 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, materialized 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 materialized 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.

Indexes are used automatically by the query optimizer to speed up the processing of a query. See The TimesTen Query Optimizer.

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, which enables a unique index to have multiple rows with the same set of values.

TimesTen Scaleout supports both local and global indexes. When you create a local index, then an index is created in each element that maps to rows in that element. A global index maps all rows in the database in a hash distribution scheme by creating a materialized view with a local index and a hash distribution scheme to the index key columns. See Understanding Indexes in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

When sorting data values, TimesTen considers null values to be larger than all non-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 SQL Statements in the *Oracle TimesTen In-Memory Database SQL Reference*.

The following sections describe how to manage your index:

Overview of Index Types

TimesTen provides two types of indexes to enable fast access to tables. You can create up to 500 range or hash indexes on a table.

- **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.

  Range indexes and equi-joins 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.

  Range indexes are optimized for in-memory data management and provide efficient sorting by column value.

- **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.

  Hash indexes are faster than range indexes for exact match lookups, but they require more space than range indexes. Hash indexes can only be used for exact value lookups. Hash indexes cannot be used if the SQL query returns a range of values. Also, hash indexes are not useful for sorting values that come from a table scan.

  TimesTen may create temporary hash and range indexes automatically during query processing to speed up query processing. Alternatively, you can perform lookups by `RowID` for fast access to data. See ROWID Data Type in the *Oracle TimesTen In-Memory Database SQL Reference*.

See CREATE INDEX and CREATE TABLE sections of the *Oracle TimesTen In-Memory Database SQL Reference* for details on creating hash indexes. For details on how to size a hash table, see Size Hash Indexes Appropriately.

Creating an Index

To create an index, issue the SQL statement `CREATE INDEX`.

TimesTen converts index names to upper case characters.

When you create an index on TimesTen Scaleout, it creates the index on all elements that are in the distribution map. TimesTen Scaleout populates each element's index with the rows that are stored on that element.

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` for indexes on replication tables.

> **Note:**
>
> You cannot create an index on LOB columns.

The following example creates an index `ixid` over column `cust_id` of table `customer`.

```
Command> CREATE INDEX ixid ON customer (cust_id);
```

The following creates a unique hash index on the `customer` table as part of the table creation:

```
Command> 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 non-unique hash index on the `customer` table over the customer name:

```
Command> CREATE HASH INDEX custname_idx ON customer(cust_name);
```

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.

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 drop a TimesTen index, issue the `DROP INDEX` SQL statement.

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`.

The following example drops the index named `ixid`.

```
Command> DROP INDEX ixid;
```

All indexes in a table are dropped automatically when the table is dropped. When you drop an index on TimesTen Scaleout, it drops the index on all elements.

Estimating the Size of an Index

Increasing the size of a TimesTen database can be done on first connect. Use the `ttSize` utility to estimate database size, including any indexes.

Correctly estimating the size of a database helps you to avoid having to resize the database in the future.

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:

 Index PAT 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. Use the Index Advisor 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 issue. 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 Running 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
- Example Using Index Advisor Built-In Procedures

Prepare for Running the Index Advisor

Before you run the Index Advisor, you can optionally set any relevant optimizer hints and update statistics for the tables included in the SQL workload and force statements to be re-prepared during the capture.

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. See Use Optimizer Hints to Modify the Execution Plan.
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-rund. 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:**
>
> 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.

```
Command> 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.

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.

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 run 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 processing of the SQL workload.
  - Base the collection of index information not on a current processing of the SQL workload, but on existing computed statistics and query plan analysis. In this scenario,

the SQL statements have been prepared, but not run. This mode can only be performed 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:**
>
> Run `ttIndexAdviceCaptureDrop` to free the temporary space after a capture. See Drop Data Collected for the Index Advisor and Finalize Results.

The following example starts a collection for the Index Advisor at the connection-level for the current processing of a SQL workload:

```
call ttIndexAdviceCaptureStart(0,0);
```

The following example ends the collection for the connection-level capture:

```
call ttIndexAdviceCaptureEnd(0);
```

> **Note:**
>
> See ttIndexAdviceCaptureStart and ttIndexAdviceCaptureEnd in the *Oracle TimesTen In-Memory Database Reference*.

Retrieve Index Recommendations and Data Collection Information

Use the `ttIndexAdviceCaptureInfoGet` and `ttIndexAdviceCaptureOutput` built-in procedures to retrieve the data collection overview and Index Advisor recommendations. Run either or both for the data you want.

1. Call the `ttIndexAdviceCaptureInfoGet` built-in procedure to retrieve data collection overview information for the Index Advisor. See Retrieve Data Collection Information with ttIndexAdviceCaptureInfoGet.
2. Call the `ttIndexAdviceCaptureOutput` built-in procedure to retrieve the recommended indexes. See Retrieve Index Recommendations with ttIndexAdviceCaptureOutput.
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 run 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 run 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:**
>
> 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, run `ttIndexAdviceCaptureOutput` with `captureLevel` set to 0 in the same connection that initiated the capture. For a database-level capture, run `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 SQL workload.
- `createStmt` - The 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:**
>
> 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:**
>
> See ttIndexAdviceCaptureDrop in the *Oracle TimesTen In-Memory Database Reference*.

You can repeat the steps in Prepare for Running the Index Advisor and Retrieve Index Recommendations and Data Collection Information until a SQL workload runs with no more index recommendations. You can keep updating the statistics for the tables on which the new indexes were applied and re-run the Index Advisor to see if any new indexes are now recommended.

Example Using Index Advisor Built-In Procedures

The Index Advisor built-in procedures shows the flow of a data collection for a SQL workload and provides index advice.

```
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.

- 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:ffffff`.

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, issue `INSERT` or `INSERT SELECT`. You can also use the `ttBulkCp` utility.

The following example demonstrates how to insert a row in the table `customer`.

```
Command> INSERT INTO customer 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, issue the `DELETE` statement.

The following example deletes all the rows from the table `customer` for names that start with the letter "S."

```
Command> DELETE FROM customer WHERE cust_name 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 in the *Oracle TimesTen In-Memory Database Security Guide*.

After synonyms are created, they can be viewed using the following views:

- `SYS.ALL_SYNONYMS`: describes the synonyms accessible to the current user. 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. 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. See SYS.USER_SYNONYMS in the *Oracle TimesTen In-Memory Database System Tables and Views Reference*.

The following sections describe using synonyms in TimesTen.

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 issue the `CREATE SYNONYM` or `CREATE OR REPLACE SYNONYM` statements, the user must have the appropriate privileges, as described in Object Privileges for Synonyms in the *Oracle TimesTen In-Memory Database Security Guide*.

- *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. Issue 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 runs 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.
```

See CREATE SYNONYM 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. See DROP SYNONYM 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.

Understanding System Views

There are several local (`V$`) global (`GV$`) system views you can query to retrieve metadata information about your database.

In TimesTen Scaleout:

- The `V$` views contain data for the element to which your application is connected.
- The `GV$` views contain the contents of the `V$` view for every element of the database.

In TimesTen Classic:

- The `V$` views contain rows of data for the database to which your application is connected.

- The GV$ views contain the same contents as their corresponding V$ view.

In addition, there are several views that you can query that are based on TimesTen built-in procedures. See System Tables and Views in the *Oracle TimesTen In-Memory Database System Tables and Views Reference*.

9

The TimesTen Query Optimizer

The TimesTen query optimizer is a cost-based optimizer that determines the most efficient way to run a given query by considering possible query plans. The query optimizer uses information about an application's tables and their available indexes to choose a fast path to the data.

A query plan in TimesTen Scaleout is affected by the distribution scheme and the distribution keys of a hash distribution scheme as well as the column and table statistics, the presence or absence of indexes, the volume of data, the number of unique values, and the selectivity of predicates.

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:

Using the Query Optimizer to Choose Optimal Plan

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. See Control the Invalidation of Commands in the SQL Command Cache.
- 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. In addition, see ttOptUpdateStats, or ttOptEstimateStats in the *Oracle TimesTen In-Memory Database Reference*.

An invalid statement is usually re-prepared automatically just before it is re-run. 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 run.
`SQLExecute`	Command is re-run.
Create Index on T1	Command is invalidated.
`SQLExecute`	Command is reprepared, then is run.
`SQLExecute`	Command is re-run.
`ttOptUpdateStats` on T1	Command is invalidated if the invalidate flag is passed to the `ttOptUpdateStats` procedure.
`SQLExecute`	Command is reprepared, then is run.
`SQLExecute`	Command is re-run.
`SQLTransact`	Command is committed.
`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 run.

Action	Description
`PreparedStatement.execute()`	Command is re-run.
Create Index on T1	Command is invalidated.
`PreparedStatement.execute()`	Command is reprepared, then is run.
`PreparedStatement.execute()`	Command is re-run.
`ttOptUpdateStats` on T1	Command is invalidated if the invalidate flag is passed to the `ttOptUpdateStats` procedure.
`PreparedStatement.execute()`	Command is reprepared, then is run.
`PreparedStatement.execute()`	Command is re-run.
`Connection.commit()`	Command is committed.
`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 a SQL Query Execution Plan. 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 after they run—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.

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 running 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 run 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 runtime in seconds for the statement.
- Last measured runtime in seconds for the statement.
- The minimum runtime in seconds for the statement.
- The unique identifier of the statement compiled across the TimesTen.
- Total temporary size in bytes used for this TimesTen statement the last time it ran.
- The maximum temporary size in bytes ever used to run this TimesTen 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:

Example: Displaying all commands in the SQL command cache

This example runs 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, run `vertical on` before calling the `ttSQLCmdCacheInfo` procedure.

```
Command> vertical 1;
Command> call ttSQLCmdCacheInfo;

  SQLCMDID:                          43402480
  PRIVATE_COMMAND_CONNECTION_ID:     -1
  EXECUTIONS:                        1
  PREPARES:                          1
  REPREPARES:                        0
  FREEABLE:                          1
  SIZE:                              4344
  OWNER:                             HR
  QUERYTEXT:                         INSERT INTO employees VALUES
        ( 191
        , 'Randall'
        , 'Perkins'
        , 'RPERKINS'
        , '650.505.4876'
        , TO_DATE('19-DEC-1999', 'dd-MON-yyyy')
        , 'SH_CLERK'
        , 2500
        , NULL
        , 122
        , 50
        )
  FETCHCOUNT:                        0
  STARTTIME:                         2015-04-09 15:22:22.139000
  MAXEXECUTETIME:                    0
  LASTEXECUTETIME:                   0
  MINEXECUTETIME:                    0
  GRIDCMDID:                         <NULL>
  TEMPSPACEUSAGE:                    0
  MAXTEMPSPACEUSAGE:                 0


  SQLCMDID:                          43311000
  PRIVATE_COMMAND_CONNECTION_ID:     -1
  EXECUTIONS:                        1
  PREPARES:                          1
  REPREPARES:                        0
  FREEABLE:                          1
  SIZE:                              4328
  OWNER:                             HR
  QUERYTEXT:                         INSERT INTO employees VALUES
        ( 171
        , 'William'
        , 'Smith'
        , 'WSMITH'
        , '011.44.1343.629268'
        , TO_DATE('23-FEB-1999', 'dd-MON-yyyy')
        , 'SA_REP'
        , 7400
        , .15
```

```
        , 148
        , 80
        )
  FETCHCOUNT:                   0
  STARTTIME:                    2015-04-09 15:22:22.139000
  MAXEXECUTETIME:               0
  LASTEXECUTETIME:              0
  MINEXECUTETIME:               0
  GRIDCMDID:                    <NULL>
  TEMPSPACEUSAGE:               0
  MAXTEMPSPACEUSAGE:            0

 ...
102 rows found.
```

Example: 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, run `vertical on` before calling the `ttSQLCmdCacheInfo` built-in.

```
Command> call ttSQLCmdCacheInfo(43311000);
< 43311000, -1, 1, 1, 0, 1, 4328, HR, INSERT INTO employees VALUES
( 171, 'William', 'Smith', 'WSMITH', '011.44.1343.629268', TO_DATE('23-FEB-1999',
 'dd-MON-yyyy'), 'SA_REP', 7400, .15, 148, 80), 0, 2015-04-09 15:22:22.139000, 0,
 0, 0, <NULL>, 0, 0 >
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 run, 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:**
>
> See Control the Invalidation of Commands in the SQL Command Cache. 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 runs, 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 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 and Views in the *Oracle TimesTen In-Memory Database System Tables and Views Reference*.

The following example generates a query plan with the following query:

```
Command> 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 processing.

Step	Level	Operation	TblNames	IXName	Pred	Other Pred
1	3	TblLkRangeScan	t1	IX1	N/A	N/A
2	3	TblLkRangeScan	t2	IX2(D)	N/A	t2.b <> 0
3	2	MergeJoin	N/A	N/A	t1.a = -t2.a	N/A
4	2	TblLkRangeScan	t3	IX3(D)	N/A	N/A
5	1	MergeJoin	N/A	N/A	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.

Describing the PLAN Table Columns

The `SYS.PLAN` table has seven columns.

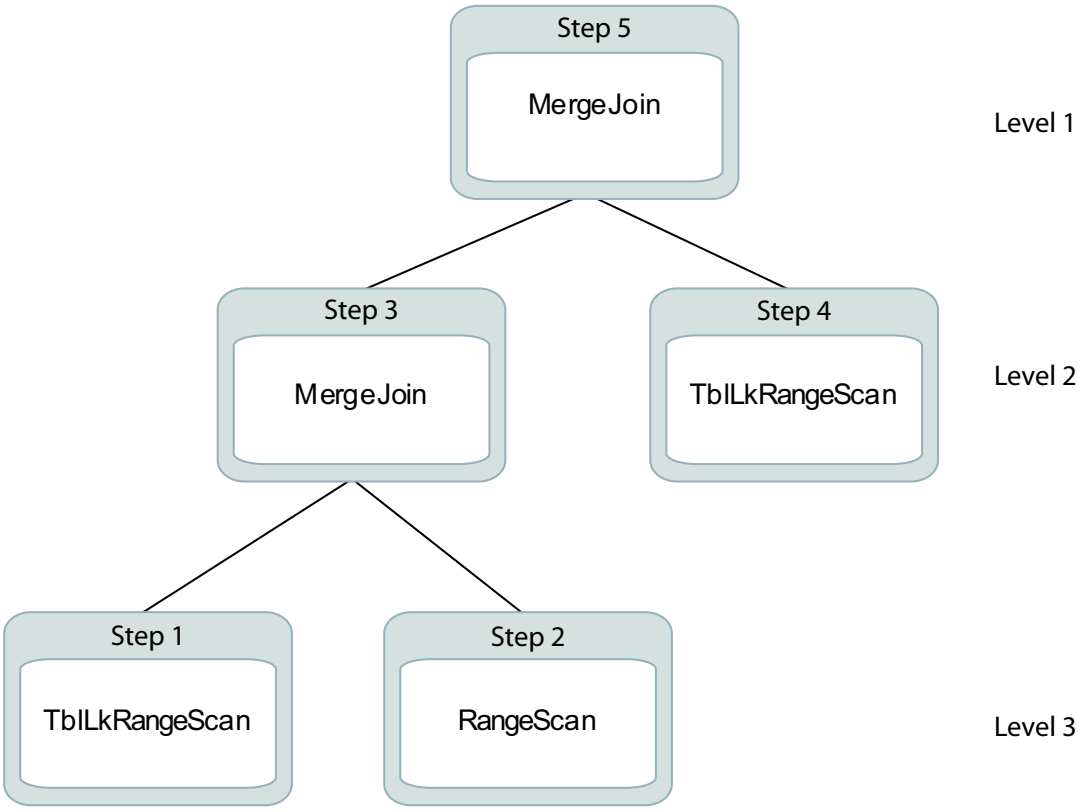
- Column 1 (Step)

  Indicates the order of operation, which always starts with one. This example 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 query processing. For this example, the join tree is as follows:



- Column 3 (Operation)

  Indicates the type of operation being run. For a description of the potential values in this field and the type of table scan each represents, see `SYS.PLAN` in System Tables and Views 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 runtime 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 this example, 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 runs. 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 this example, 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.

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.

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:**
>
> See Reading Query Plan from the PLAN Table. 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 run the `ttIsql explain` command for a formatted version of this output. See Display Query Plan for Statement in SQL Command Cache.

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.

Example: 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 running 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, run `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> >
< 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: 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, run `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 a SQL Query Execution Plan

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.

See Viewing a Query Plan **from** the System PLAN Table.

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?

You evaluate and modify an execution plan if it is ill-suited for your application or if it is not optimal in performance.

- **The plan is optimally fast but is ill-suited for the application.** The optimizer may select the fastest query processing 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 run 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.

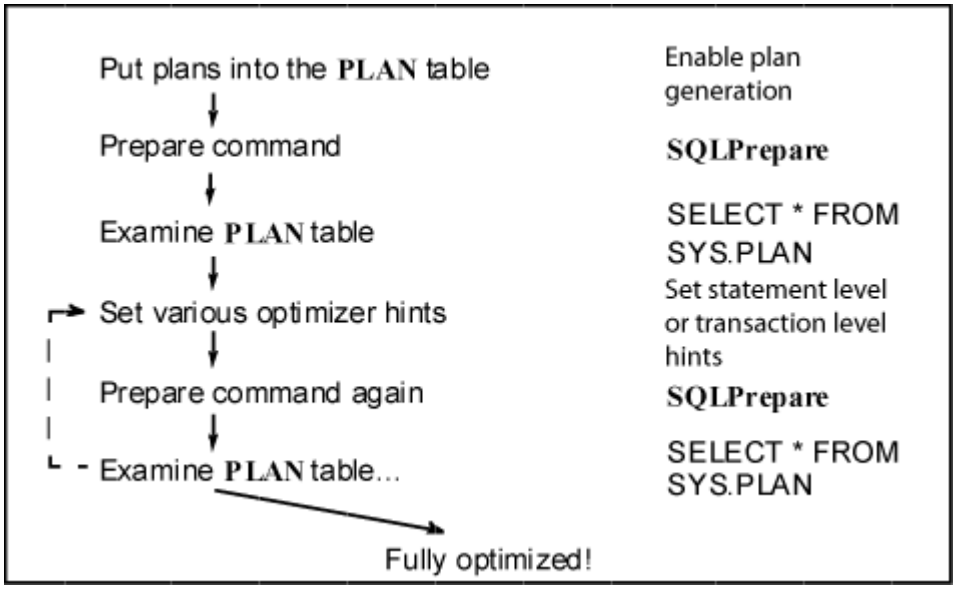
Transaction level hints are in effect for all calls to the ODBC `SQLPrepare` function or JDBC `PreparedStatement` objects in the transaction.

- If a command is prepared with certain hints in effect, those hints continue to apply if the command is re-prepared 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 Using the Query Optimizer to Choose Optimal Plan.
- If a command is prepared without hints, subsequent hints do not affect the command if it is re-prepared automatically. An application must call the ODBC `SQLPrepare` function or JDBC `Connection.prepareStatement()` method a second time so that hints have an effect.

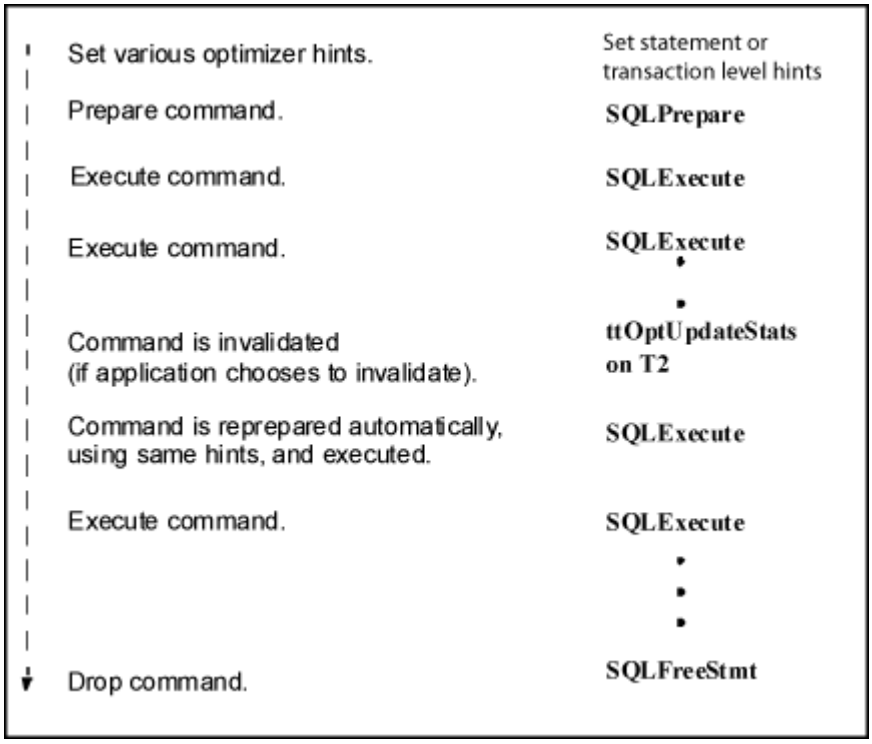
Tuning a Join When Using ODBC

A developer can tune 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.



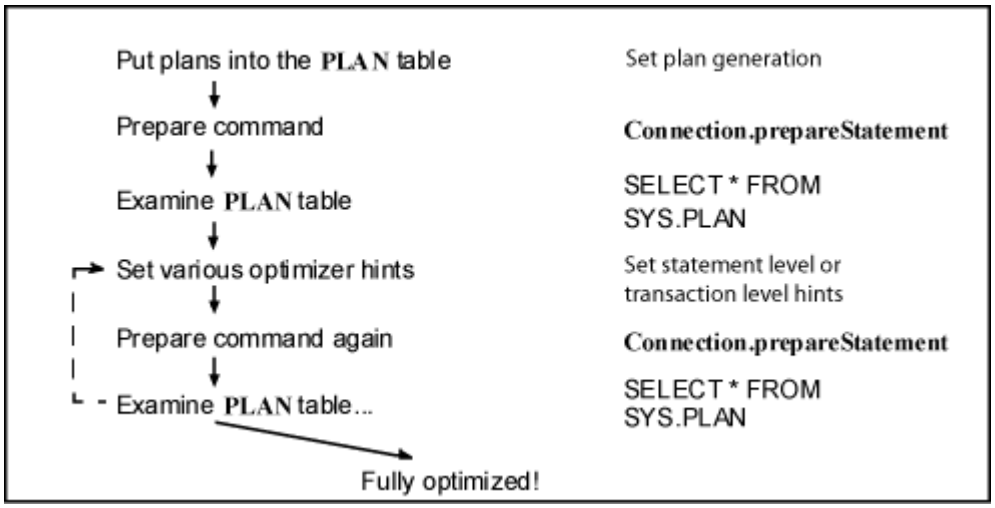
During processing, the application may then go through the steps shown in the following figure.



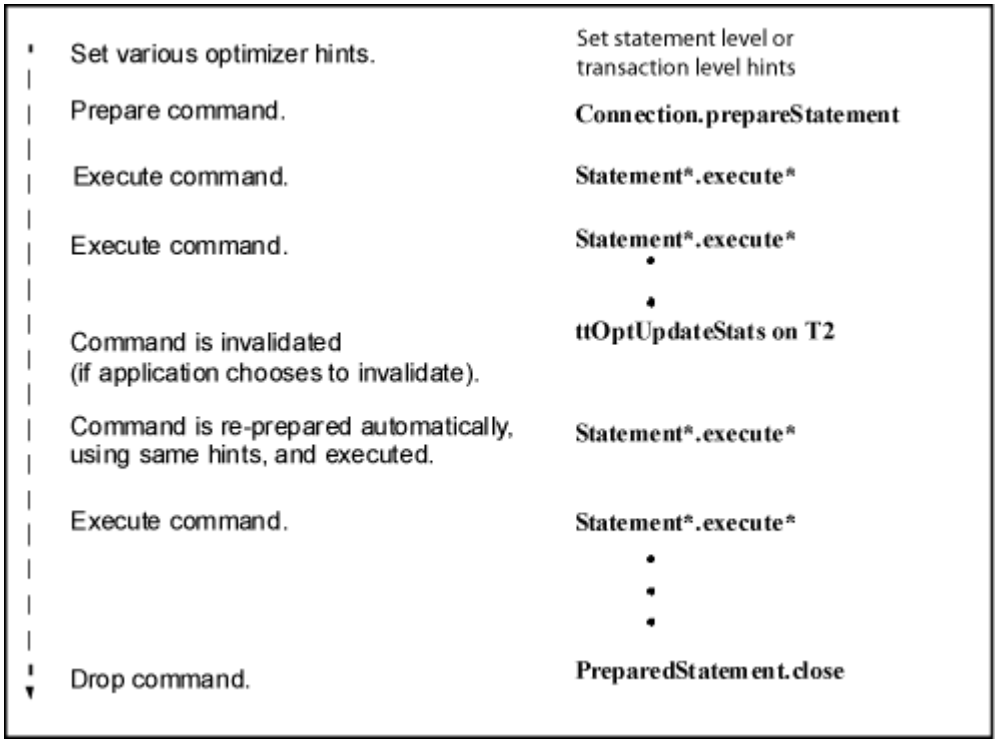
Tuning a Join When Using JDBC

A developer can tune a join when using JDBC.

When using JDBC, a developer tuning a join on `T1` and `T2` might go through the steps shown in the following figure.



During processing, the application may then go through the steps shown in the following figure.



Use Optimizer Hints to Modify the Execution Plan

You can apply hints to pass instructions to the TimesTen query optimizer.

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

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

> **Note:**
>
> TimesTen concurrently processes read and write operations optimally. Your read operations 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 operations that operate concurrently with read optimized operations 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.

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 Connection Level Optimizer Hints for a TimesTen Connection

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`.

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.

See Statement Level Optimizer Hints in the *Oracle TimesTen In-Memory Database SQL Reference* for information specific to statement level optimizer hints. See Using Optimizer Hints in the *Oracle TimesTen In-Memory Database Scaleout User's Guide* for optimizer hints that are specific to TimesTen Scaleout.

Apply Transaction Level Optimizer Hints for a Transaction

To change the query optimizer behavior for all statements in a transaction, an application calls the `ttOptSetFlag` built-in procedure 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 runs 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 TimesTen 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 runs. 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: 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: 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 */
...
```

Apply Connection Level Optimizer Hints for a TimesTen Connection

To change the query optimizer behavior for all statements of a specific connection, define the `OptimizerHint` connection attribute.

The value of the `OptimizerHint` connection attribute is a string that uses the same format as a statement level optimizer hint, but without the delimiters of `/*+`, `*/`, and `--+`.

You cannot include comments with connection level optimizer hints.

Transaction level optimizer hints overwrite connection level optimizer hints for the current transaction. After a commit, the transaction level optimizer hints are lost and the connection level optimizer hints take effect. Statement level optimizer hints overwrite transaction level optimizer hints and connection level optimizer hints for the scope of the statement. Since this is a connection attribute, the `ttConfiguration` utility shows the connection level optimizer hints.

> **Note:**
>
> In a client server setting, the client connection setting of this connection attribute overwrites the Server DSN setting of this attribute.

This example illustrates how to use connection level optimizer hints. This example uses `TT_RowLock`, `TT_TblLock`, and `TT_MergeJoin` to enable row locking, disable table locking, and disable merge joins. Note that `/disk1/timesten` is the *`timesten_home`*.

```
...
[database1]
Driver=/disk1/timesten/install/lib/libtten.so
DataStore=/disk1/timesten/info/DemoDataStore/database1
PermSize=128
TempSize=64
DatabaseCharacterSet=AL32UTF8
OptimizerHint= TT_RowLock (1) TT_TblLock (0) TT_MergeJoin (0)
...
```

TimesTen does not support the use of the semicolon (;) character in connection attributes. Therefore, when you want to use more than one `TT_INDEX` optimizer hint at the connection level, you cannot use the same syntax that you use for the statement level.

For example, `TT_INDEX(EMPLOYEES,EMP_NAME_IX,1;EMPLOYEES,EMP_MANAGER_IX,1)` is a valid statement level optimizer hint, but it is an invalid connection level optimizer hint. TimesTen merges multiple `TT_INDEX` optimizer hints if they are specified on the same line. Therefore, `TT_INDEX(EMPLOYEES,EMP_NAME_IX,1) TT_INDEX(EMPLOYEES,EMP_MANAGER_IX,1)` is equivalent to `TT_INDEX(EMPLOYEES,EMP_NAME_IX,1;EMPLOYEES,EMP_MANAGER_IX,1)`.

This example illustrates the use of multiple `TT_INDEX` optimizer hints at the connection level. Note that `/disk1/timesten` is the *`timesten_home`*.

```
...
[database1]
Driver=/disk1/timesten/install/lib/libtten.so
DataStore=/disk1/databases/info/DemoDataStore/database1
PermSize=128
```

```
TempSize=64
DatabaseCharacterSet=AL32UTF8
OptimizerHint= TT_INDEX(EMPLOYEES,EMP_NAME_IX,1) TT_INDEX(EMPLOYEES,EMP_MANAGER_IX,1)
...
```

See OptimizerHint in the *Oracle TimesTen In-Memory Database Reference*.

10

TimesTen Database Performance Tuning

An application using TimesTen should obtain an order of magnitude performance improvement in its data access over an application using a traditional RDBMS. However, poor application design and tuning can erode the TimesTen advantage.

There are 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

There are 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).

For Linux systems, you must configure shared memory so that the maximum size of a shared memory segment is large enough to contain the size of the total shared memory segment for the database.

In TimesTen Classic, the entire database resides in a single shared memory segment. In TimesTen Scaleout, you set settings for the shared memory segment of each host. Set the appropriate operating system kernel parameters for `shmmax`, `shmall`, `memlock`, and `HugePages`. See Operating System Prerequisites in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide* and Configuring Linux Kernel Parameters in the *Oracle TimesTen In-Memory Database Scaleout User's Guide*.

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*.

Avoid Paging of the Memory Used by the Database

Performance impact: Variable

On Linux, the MemoryLock connection attribute specifies whether the real memory used by the database should be locked while the database is loaded into memory.

Set MemoryLock to 4 to avoid paging of the memory used by the database. If MemoryLock is not set to 4, parts of the shared memory segment used by the database may be paged out to the swap area, resulting in poor database performance. See MemoryLock 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 Memory Region Sizes of a Database in this book and Configure shmmax and shmall in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.

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 running or have recently run. The size of this shared memory area is controlled by the PLSQL_MEMORY_SIZE first connect attribute.

When a new PL/SQL program unit is prepared for execution, it is loaded into shared memory. If shared memory space is not available, the cached recently-processed 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 running program units, then attempts by a new connection to run 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 run 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 run 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 repeatedly run, 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 runs 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 runtime 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 runtime was 0.26 seconds.

The default value for PLSQL_MEMORY_SIZE is 128 MB on Linux and UNIX systems. This should be sufficient for several hundred PL/SQL program units of reasonable complexity to run. 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.

Set Appropriate Limit on the Number of Open Files

Performance impact: Variable

For multithreaded applications that maintain many concurrent TimesTen database connections, the default number of open files permitted to each process by the operating system may be too low.

See Limits on Number of Open Files in the *Oracle TimesTen In-Memory Database Reference*.

Configure Log Buffer and Log File Size Parameters

Performance impact: Large

Increasing the value of LogBufMB and LogFileSize could have a substantial positive performance impact.

LogBufMB is related to the LogFileSize connection attribute, which should be set to the same value or integral multiples of LogBufMB for best performance.

Transaction log buffer waits occur when application processes cannot insert transaction data to the transaction log buffer and must stall to wait for transaction 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 transaction log buffer space is insufficient, file system transfer rate is insufficient, writing to the file system is taking too long, or the log flusher is CPU-bound.

- A small LogBufMB could slow down write transactions when these transactions have to wait for a log flush operation to make space for redo logging. An oversized log buffer has no impact other than consuming extra memory. The value of LOG_BUFFER_WAITS shows the number of times a thread was delayed while trying to insert a transaction log record into the log buffer because the log buffer was full. Generally, if this value is increasing, it indicates that the log buffer is too small.
- The LogFileSize attribute specifies the maximum size of each transaction log file in megabytes. If LogFileSize is too small, then TimesTen has to create multiple log files within a transaction log flush operation. The overhead of file creation often leads to LOG_BUF_WAITS events, which could significantly impact performance.

You can observe the value of the LOG_BUFFER_WAITS metric (using either the SYS.MONITOR table or the ttIsql monitor command) to see if it increases while running a workload. After which, you can increase the value of LogBufMB to improve the database performance.

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 value of LogBufMB value does not improve performance.

If log buffer waits still persist after increasing the LogBufMB and LogFileSize values, then review the other possible issues mentioned above.

The LogBufParallelism connection attribute specifies the number of transaction log buffer strands. This attribute improves transaction throughput when multiple connections run write transactions concurrently. Configure this to the lesser value of the number of CPU cores on the system and the number of concurrent connections running write transactions.

See LogBufMB, LogFileSize and LogBufParallelism in the *Oracle TimesTen In-Memory Database Reference*.

Avoid Connection Overhead for TimesTen Classic Databases

Performance impact: Large

An idle database is a database with no connections. By default, TimesTen loads an idle database 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 the file system.

If applications are continually connecting and disconnecting from a database, the database may be loaded to and unloaded from memory continuously, resulting in excessive file system 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 a TimesTen Classic database to enable databases to always remain in memory. The trade-off is that since the TimesTen Classic database is never unloaded from memory, a final disconnect checkpoint never occurs. So, applications should configure frequent checkpoints with the `ttCkptConfig` built-in procedure. See ttCkptConfig in the *Oracle TimesTen In-Memory Database Reference*.

Alternatively, you can specify that the TimesTen Classic database remain in memory for a specified interval of time and accept new connections. If no new connections occur in this interval, TimesTen Classic 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 TimesTen Classic database, use the `ttAdmin` utility. For more details on the RAM policy and the `ttAdmin` utility, see Specifying a RAM Policy 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

Load a TimesTen Classic database into memory with the `ttAdmin -ramLoad` command before you duplicate this database with the `ttRepAdmin -duplicate` utility. This places the database in memory, available for connections, instead of unloading it with a blocking checkpoint.

See Avoid Connection Overhead for TimesTen Classic Databases in this book and the ttAdmin and ttRepAdmin sections in the *Oracle TimesTen In-Memory Database Reference*.

Prevent Reloading of the Database After Automatic Recovery Fails

Performance impact: Large

When TimesTen Classic 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 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.
- 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.

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 standard 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.

See ttBulkCp and ttMigrate in the *Oracle TimesTen In-Memory Database Reference*.

Check Your Driver

Performance impact: Large

The TimesTen direct driver: If you only use direct connections and have no need to use both direct and client-server connections from the same application process, then link directly with the TimesTen direct driver for maximum performance.

The TimesTen client driver on Windows: Use the TimesTen client driver in your DSN when using the Windows ODBC driver manager. If you do not need the functionality of the Windows ODBC driver manager, link direct with the TimesTen client driver for improved performance.

The TimesTen data manager driver on Linux and UNIX platforms: There are two versions of the TimesTen data manager driver for Linux and UNIX platforms: a debug and production version. Unless you are debugging, use the production version. The debug library can be significantly slower.

The TimesTen driver manager on Linux and UNIX platforms: If you need to use direct and client/server connections from the same application process to a TimesTen database, then you should use the TimesTen driver manager (as opposed to a generic ODBC driver manager) to minimize the performance impact. If you do not want to use the TimesTen driver manager, you can use an open source or commercial driver manager, such as unixODBC or iODBC. However, generic driver managers could result in significant performance penalties and functionality limitations.

See Creating a DSN on Linux and UNIX for TimesTen Classic and Connecting to TimesTen with ODBC and JDBC Drivers.

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, latches, locks, XLA, and TimesTen connection attributes.

Call the `ttStatsConfig` built-in procedure to modify statistics collection parameters that affect the `ttStatus` utility. See ttStatsConfig in the *Oracle TimesTen In-Memory Database Reference*.

See ttStats in the *Oracle TimesTen In-Memory Database Reference*. See TT_STATS in the *Oracle TimesTen In-Memory Database PL/SQL Packages Reference*.

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.

Use TimesTen Native Integer Data Types If Appropriate

Performance impact: Variable

Declare the column types of a table with TimesTen native integer data types, `TT_TINYINT`, `TT_SMALLINT`, `TT_INTEGER`, or `TT_BIGINT` instead of the `NUMBER` data type to save space and improve the performance of calculations.

TimesTen internally maps the `SMALLINT`, `INTEGER`, `INT`, `DECIMAL`, `NUMERIC`, and `NUMBER` data types to the `NUMBER` data type. When you define the column types of a table, ensure that you choose the appropriate native integer data type for your column to avoid overflow. For more information on native integer data types and the `NUMBER` data type, see Numeric Data Types in the *Oracle TimesTen In-Memory Database SQL Reference*.

If you use a TimesTen native integer data type when you create a TimesTen cache group that reads and writes to an Oracle database, ensure that the data type of the columns of the TimesTen cache group and the Oracle database are compatible. For more information about the mapping between Oracle Database and TimesTen data types, see Mappings Between Oracle Database and TimesTen Data Types in the *Oracle TimesTen In-Memory Database Cache Guide*.

Configure the Checkpoint Files and Transaction Log Files to be on a Different Physical Device

Performance impact: Large

For best performance, TimesTen recommends that you use the `LogDir` connection attribute to place the transaction log files on a different physical device from the checkpoint files. This is important if your application needs to have consistent response times and throughput.

TimesTen places the checkpoint files in the location specified by the `DataStore` connection attribute, which is the full path name of the database and the file name prefix. If the transaction log files and checkpoint files are on different physical devices, I/O operations for checkpoints do not block I/O operations to the transaction log and vice versa. For more information on transaction logging, see Transaction Logging.

See LogDir and DataStore in the *Oracle TimesTen In-Memory Database Reference*.

> **Note:**
>
> If the database is already loaded into RAM and the transaction log files and checkpoint files for your database are on the same physical device, TimesTen writes a message to the daemon log file.

Client/Server Tuning

You can tune your client/server.

The following sections include tips for client/server tuning:

Diagnose Client/Server Performance

Performance impact: Variable

You can analyze your client/server performance with statistics that are tracked in the `SYS.SYSTEMSTATS` table.

These statistics 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 run 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 a TimesTen client to access databases on a remote server system adds network overhead to your connections. Whenever possible, write your applications to access TimesTen locally using a direct driver, and link the application directly with TimesTen.

Choose Timeout Connection Attributes for Your Client

Performance impact: Variable

By default, connections wait 60 seconds for a TimesTen client and server to complete a network operation. The `TTC_Timeout` attribute determines the maximum number of seconds a client application waits for the result from the corresponding server process before timing out.

If you are using multiple servers for automatic client failover, then you can set the `TTC_ConnectTimeout` attribute to specify the maximum number of seconds the client waits for a `SQLDriverConnect` or `SQLDisconnect` request. It overrides the value of `TTC_Timeout` for those requests. Set the `TTC_ConnectTimeout` when you want connection requests to timeout with a different timeframe than the timeout provided for query requests. For example, you can set a longer timeout for connections if you know that it takes a longer time to connect, but set a shorter timeout for all other queries.

The overall elapsed time allotted for the `SQLDriverConnect` call depends on if automatic client failover is configured:

- If there is just one `TTC_SERVER`*n* connection attribute defined, then there is no automatic client failover configured and the overall elapsed time allotted for the initial connection attempt is limited to 10 seconds.
- If automatic client failover is defined and there is more than one `TTC_SERVER`*n* connection attribute configured, then the overall elapsed time is four times the value of the timeout (whether `TTC_Timeout` or `TTC_ConnectTimeout`). For example, if the user specifies `TTC_TIMEOUT`=10 and there are multiple `TTC_SERVER`*n* servers configured, then the `SQLDriverConnect` call could take up to 40 seconds.

However, within this overall elapsed timeout, each connection attempt can take the time specified by the `TTC_Timeout` or `TTC_ConnectTimeout` value, where the `TTC_ConnectTimeout` value overrides the `TTC_Timeout` value.

See Choose SQL and PL/SQL Timeout Values for information about how `TTC_Timeout` and `TTC_ConnectTimeout` relates to SQL and PL/SQL timeouts.

See TTC_Timeout and TTC_ConnectTimeout in the *Oracle TimesTen In-Memory Database Reference*.

Choose a Lock Wait Timeout interval

Performance impact: Variable

By default, connections wait 10 seconds to acquire a lock. To change the timeout interval for locks, set the `LockWait` conection attribute or use the `ttLockWait` built-in procedure.

When running a workload of high lock-contention potential, consider setting the `LockWait` connection attribute to a smaller value for faster return of control to the application, or setting `LockWait` to a larger value to increase the successful lock grant ratio (with a risk of decreased throughput).

See Setting Wait Time for Acquiring a Lock in this book or LockWait or 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 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:**
>
> 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:**
>
> 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.

See Transaction Isolation Levels.

Enable Prefetch Close for Read-Only Transactions

Performance impact: Variable

A TimesTen extension enables applications to optimize read-only query performance in client/server applications. When you enable prefetch close, the server closes the cursor and commits the transaction after the server has fetched the entire result set for a read-only query. This enhances performance by decreasing the network round-trips between client and server.

- ODBC: Use the `SQLSetConnectOption` to set the `TT_PREFETCH_CLOSE` ODBC connection option. See Optimizing Query Performance in the *Oracle TimesTen In-Memory Database C Developer's Guide*.
- JDBC: Call the `TimesTenConnection` method `setTtPrefetchClose()` with a setting of `true`. See Optimizing Query Performance in the *Oracle TimesTen In-Memory Database Java Developer's Guide*.

Use a Connection Handle When Calling SQLTransact

Performance impact: Large

You should always call `SQLTransact` with a valid HDBC and a null environment handle.

```
SQLTransact (SQL_NULL_HENV, ValidHDBC, fType)
```

It is possible to call `SQLTransact` with a non-null environment handle and a null connection handle, but that usage is not recommended as it abruptly commits or rolls back every transaction for the process, which is slow and can cause unexpected application behavior.

Enable Multi-Threaded Mode to Handle Concurrent Connections

The `MaxConnsPerServer` connection attribute sets the maximum number of concurrent connections allowed for a TimesTen server process.

By default, `MaxConnsPerServer` equals to 1, meaning that each TimesTen server process can only handle one client connection.

Setting `MaxConnsPerServer` > 1 enables multi-threaded mode for the database so that it can use threads instead of processes to handle client connections. This reduces the time required for applications to establish new connections and increases overall efficiency in configurations that use a large number of concurrent client/server connections.

See Defining Server DSNs for TimesTen Server on a Linux or UNIX System.

SQL Tuning

After you have determined the overall locking and I/O strategies, make sure that the individual SQL statements run as efficiently as possible.

The following sections describe how to streamline your SQL statements:

- Tune Statements and Use Indexes
- Collect and Evaluate Sampling of Runtimes for SQL Statements
- Creating Tables Without Indexes for Performance When Loading Data
- Select the Appropriate Type of Index
- Size Hash Indexes Appropriately
- Use Foreign Key Constraint Appropriately
- Compute Exact or Estimated Statistics
- Update Table Statistics for Large Tables in Parallel
- Create Script to Regenerate Current Table Statistics
- Control the Invalidation of Commands in the SQL Command Cache
- Avoid ALTER TABLE
- Avoid Nested Queries
- Prepare Statements in Advance
- Avoid Unnecessary Prepare Operations
- Store Data Efficiently with Column-Based Compression of Tables
- Control Read Optimization During Concurrent Write Operations
- Choose SQL and PL/SQL Timeout Values

Tune Statements and Use Indexes

Performance impact: Large

Verify that all statements run 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...
```

The TimesTen Query Optimizer describes how to view the plan that TimesTen uses to run a statement. Alternatively, you can use the `ttIsql showplan` command to view the plan. View the plan for each frequently run 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.

Collect and Evaluate Sampling of Runtimes for SQL Statements

Performance impact: Variable

TimesTen provides built-in procedures that measure the runtime of SQL operations to determine the performance of SQL statements.

Instead of tracing, the built-in procedures sample the runtime of SQL statements during processing. The built-in procedures measure the runtime of SQL statements by timing the processing within the `SQLExecute` API.

You can configure the sampling rate and how the runtimes are collected with the `ttStatsConfig` built-in procedure and the following name-value pairs:

> **Note:**
>
> 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 runtime 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 runtime for every 10th statement.

Table 10-1 (Cont.) ttStatsConfig Parameter and Value Descriptions

Parameter	Description
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 runtime 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 runtime 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 runtime statistics:

- Number of fetch executions performed internally for this statement.
- The timestamp when the statement started.
- The maximum wall clock run time in seconds of this statement.
- Last measured runtime in seconds of the statement.
- The minimum runtime 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:                       135680792
  PRIVATE_COMMAND_CONNECTION_ID:  -1
  EXECUTIONS:                     97414
  PREPARES:                       50080
  REPREPARES:                     1
  FREEABLE:                       1
  SIZE:                           3880
  OWNER:                          SALES
  QUERYTEXT:                      select min(unique2) from big1
  FETCHCOUNT:                     40
  STARTTIME:                      2018-04-10 13:10:46.808000
  MAXEXECUTETIME:                 .001319
  LASTEXECUTETIME:                .000018
  MINEXECUTETIME:                 .000017
  EXECLOC:                        0
  GRIDCMDID:                      00000000000000000000
TEMPSPACEUSAGE:                 0  MAXTEMPSPACEUSAGE:              0
1 row found.
```

See ttSQLCmdCacheInfo in the *Oracle TimesTen In-Memory Database Reference*.

The `ttSQLExecutionTimeHistogram` built-in procedure displays a histogram of SQL runtimes 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 runtime 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 runtime.
- The runtime limit that denotes each time frame.
- The last row shows the number of SQL statements that ran 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 ran. The total time for all 1919 statements to run was 1.090751 seconds. This example shows that SQL statements ran in the following time frames:

- 278 statements ran in a time frame that was less than or equal to .00001562 seconds.
- 1484 statements ran in a time frame that was greater than .00001562 seconds and less than or equal to .000125 seconds.
- 35 statements ran in a time frame that was greater than .000125 seconds and less than or equal to .001 seconds.
- 62 statements ran in a time frame that was greater than .001 seconds and less than or equal to .008 seconds.
- 60 statements ran 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.
```

Creating Tables Without Indexes for Performance When Loading Data

Performance impact: Variable

If you are planning to load your tables with data, consider creating your tables without indexes.

After the data is loaded, you can then create your indexes. This reduces the time it takes to load the data into the tables. The exception is if you are using foreign keys and reference tables.

Select the Appropriate Type of Index

Performance impact: Variable

The TimesTen database supports hash and range 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.

> **Note:**
>
> If you are planning to load your tables with data, consider creating your tables without indexes. After the data is loaded, you can then create your indexes. This reduces the time it takes to load the data into the tables. The exception is if you are using foreign keys and reference tables.

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:

```
Command> CREATE TABLE T(i1 tt_integer, i2 tt_integer, i3 tt_integer, ...);
Command> CREATE INDEX IXT on T(i1, i2, i3);
```

The index `IXT` can be used to speed up the following queries:

```
Command> SELECT * FROM T WHERE i1>12;
Command> SELECT * FROM T WHERE i1=12 and i2=75;
Command> SELECT * FROM T WHERE i1=12 and i2 BETWEEN 10 and 20;
Command> 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:

```
Command> 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:

```
Command> SELECT * FROM T WHERE i1=12 and i2<50 and i3=630;
```

Range indexes have a dynamic structure that adjusts itself automatically to accommodate changes in table size. A range index can be either unique or non-unique and can be declared over nullable columns. It also allows the indexed column values to be changed once a record is inserted. A range index is likely to be more compact than an equivalent hash index.

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 The TimesTen Query Optimizer.

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 run 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.

> **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:**
>
> See ttIsql and Built-In Procedures in the *Oracle TimesTen In-Memory Database Reference*.

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. 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 run the `ttOptCmdCacheInvalidate('',1)` built-in procedure. See ttOptCmdCacheInvalidate in the *Oracle TimesTen In-Memory Database Reference*. During high transaction load times run 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:**
>
> 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 run 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 re-prepared automatically just before it is re-run. A single command may be prepared several times.

> **Note:**
>
> See Using the Query Optimizer to Choose Optimal Plan 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. 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 enables 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 DML statements and queries on the table to run more slowly.

The actual performance degradation that 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:

```
Command> 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`:

```
Command> 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:

```
Command> SELECT * FROM (SELECT x1 FROM t1 UNION SELECT x2 FROM t2),
(SELECT x3 FROM t3 GROUP BY x3) WHERE x1=x3;
```

See Subqueries in the *Oracle TimesTen In-Memory Database SQL Reference*.

Prepare Statements in Advance

Performance impact: Large

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 run 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 runs 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*.

Store Data Efficiently with Column-Based Compression of Tables

Performance impact: Large

TimesTen provides the ability to compress tables at the column level, which stores the data more efficiently at the cost of lower performance.

This mechanism provides space reduction for tables by eliminating the redundant storage of duplicate values within columns and improves the performance of SQL queries that perform full table scans.

When compressing columns of a TimesTen table, consider the following:

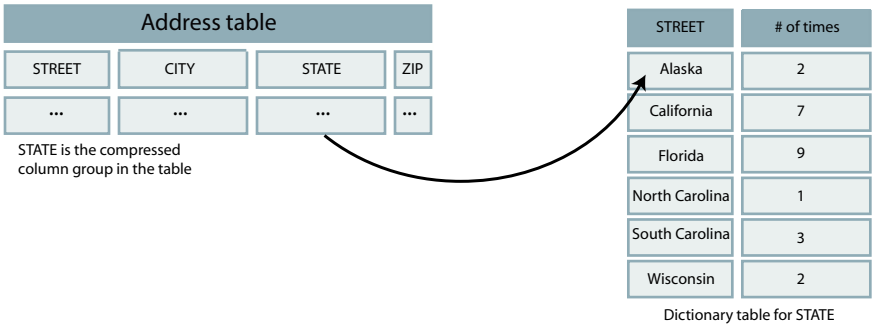
- Compress a column if values are repeated throughout such as the name of countries or states.
- Compress a column group if you often access multiple columns together.
- Do not compress columns that contain data types that require a small amount of storage such as `TT_TINYINT`.
- TimesTen does not compress `NULL` values.

You can define one or more columns in a table to be compressed together, which is called a compressed column group. You can define one or more compressed column groups in each table.

A dictionary table is created for each compressed column group that contains a column with all the distinct values of the compressed column group. The compressed column group now contains a pointer to the row in the dictionary table for the appropriate value. The width of this pointer can be 1, 2, or 4 bytes long depending on the maximum number of entries you defined for the dictionary table. So if the sum of the widths of the columns in a compressed column group is wider than the 1, 2, or 4 byte pointer width, and if there are a lot of duplicate values of those column values, you have reduced the amount of space used by the table.

Figure 10-1 shows the compressed column group in the table pointing to the appropriate row in the dictionary table.

Figure 10-1 Column-Based Compression of Tables



The dictionary table has a column of pointers to each of the distinct values. When the user configures the maximum number of distinct entries for the compressed column group, the size of the compressed column group is set as follows:

- 1 byte for a maximum number of entries of 255 (2^8-1). When the maximum number is between 1 and 255, the dictionary size is set to 255 (2^8-1) values and the compressed column group pointer column is 1 byte.
- 2 bytes for a maximum number of entries of 65,535 (2^{16}-1). When the maximum number is between 256 and 65,535, the dictionary size is set to 65,535 (2^{16}-1) values and the compressed column group pointer column is 2 bytes.
- 4 bytes for a maximum number of entries of 4,294,967,295 (2^{32}-1). When the maximum number is between 65,536 and 4,294,967,295, the dictionary size is set to 4,294,967,295 (2^{32}-1) values and the compressed column group pointer column is 4 bytes. This is the default.

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. 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 operations optimally. Your read operations 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 operations 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.
>
> For more information about optimizer hints, see Use Optimizer Hints to Modify the Execution Plan.
>
> 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.

Running 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:**
>
> See ttDBWriteConcurrencyModeSet, ttLockWait, and ttDBWriteConcurrencyModeGet in the *Oracle TimesTen In-Memory Database Reference*.

Choose SQL and PL/SQL Timeout Values

Performance impact: Variable

You should consider the relationship between certain connection attributes when setting timeout values.

- SQLQueryTimeout or SQLQueryTimeoutMSec: Controls how long a SQL statement runs before timing out.

  By default, SQL statements do not time out. In some cases, you may want to specify a value for either the SQLQueryTimeout or SQLQueryTimeoutMSec connection attribute to set the time limit in seconds or milliseconds within which the database should run SQL statements. (Note that this applies to any SQL statement, not just queries.)

  Both SQLQueryTimeout and SQLQueryTimeoutMsec attributes are internally mapped to one timeout value in milliseconds. If different values are specified for these attributes, only one value is retained. See SQLQueryTimeout and SQLQueryTimeoutMSec in the *Oracle TimesTen In-Memory Database Reference*.

- PLSQL_TIMEOUT: Controls how long a PL/SQL block runs before timing out.

  By default, PL/SQL program units (PL/SQL procedures, anonymous blocks and functions) are allowed to run for 30 seconds before being automatically terminated. In some cases,

you may want to modify the PLSQL_TIMEOUT connection attribute value to allow PL/SQL program units additional time to run. You can also modify this attribute with an ALTER SESSION statement during runtime.

See PLSQL_TIMEOUT in the *Oracle TimesTen In-Memory Database Reference*.

- TTC_Timeout: Controls how long a TimesTen client waits for a response from the TimesTen Server when the client has requested the server to run a SQL statement or PL/SQL block.
- TTC_ConnectTimeout: Controls how long the client waits for a SQLDriverConnect or SQLDisconnect request. It overrides the value of TTC_Timeout for those requests.

  See Choose Timeout Connection Attributes for Your Client.

If you use a TimesTen client/server, then SQLQueryTimeout (or SQLQueryTimeoutMSec) and PLSQL_TIMEOUT (relevant for PL/SQL) should be set to significantly lower values than TTC_Timeout, to avoid the possibility of the client mistaking a long-running SQL statement or PL/SQL block for a non-responsive server.

If you use PL/SQL, the relationship between SQLQueryTimeout (or SQLQueryTimeoutMSec) and PLSQL_TIMEOUT depends on how many SQL statements you use in your PL/SQL blocks. If none, there is no relationship. If the maximum number of SQL statements in a PL/SQL block is *n*, then PLSQL_TIMEOUT should presumably equal at least *n* x SQLQueryTimeout (or *n* x 1000 x SQLQueryTimeoutMSec), including consideration of processing time in PL/SQL.

> **Note:**
>
> If SQLQueryTimeout (or SQLQueryTimeoutMSec) and PLSQL_TIMEOUT are not set sufficiently less than TTC_Timeout, and the client mistakes a long-running SQL statement or PL/SQL block for a non-responsive server and terminates the connection, the server will cancel the SQL statement or PL/SQL block as soon as it notices the termination of the connection.

Materialized View Tuning

You can improve 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, `cust_order` is a materialized view of customer orders, based on two tables. The tables are `customer` and `book_order`. The former has two columns (`cust_id` and `cust_name`) and the latter has three columns (`order_id`, `book`, and `cust_id`). If you often update the `book_order` table to change a particular order by using the condition `book_order.order_id=const`, then create an index on `cust_order.order_id`. On the other hand, if you often update based on the condition `book_order.cust_id=const`, then create an index on `Cust_order.cust_id`.

If you often update using both conditions and cannot afford to create both indexes, you may want to add `book_order.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:

```
Command> CREATE MATERIALIZED VIEW v1 AS SELECT x1 FROM t1, t2 WHERE x1=x2;
Command> 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 run 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

You can increase performance when using transactions.

- Locate Checkpoint and Transaction Log Files on Separate Physical Device
- Size Transactions Appropriately
- Use Durable Commits Appropriately
- Avoid Manual Checkpoints
- Turn Off Autocommit Mode
- Avoid Transaction Roll Back
- Avoid Large DELETE Statements
- Increase the Commit Buffer Cache Size

Locate Checkpoint and Transaction Log Files on Separate Physical Device

To improve performance, locate your transaction log files on a separate physical device from the one on which the checkpoint files are located. The `LogDir` connection attribute determines where log files are stored.

See Managing Transaction Log Buffers and Files in this book or LogDir in the *Oracle TimesTen In-Memory Database Reference*.

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 file system write operation when the transaction commits. File system I/O affects response time and may affect throughput, depending on how effective group commit is.

Performance-sensitive applications should avoid unnecessary file system write operations at commit. Use a performance analysis tool to measure the amount of time your application spends in file system write operations (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 file system write operations are performed at transaction commit. See Use Durable Commits Appropriately.

Long transactions perform fewer file system write operations per unit of time than short transactions. However, long transactions also can reduce concurrency. See 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 rollback operations.
- 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 file system write operation 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 file system write operations by performing non-durable commits.
Non-durable commits do everything that a durable commit does except write the transaction log to the file system. Locks are released and cursors are closed, but no file system write operation is performed.

> **Note:**
>
> Some drivers only write data into cache memory or write to the file system 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 file system to write to the recording media before reporting completion or use an uninterruptible power supply.

The advantage of non-durable 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 be persisted to the file system 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 the file system when internal buffers fill up, limiting the amount of data that could be lost.

Transactions can be made durable or can be made nondurable on a connection-by-connection basis. Applications can force a durable commit of a specific transaction by calling the `ttDurableCommit` procedure.

Applications that use durable commits can benefit from using synchronous writes in place of write and flush. To turn on synchronous writes set the first connection attribute `LogFlushMethod=1`. The `LogFlushMethod` attribute controls how TimesTen writes and synchronizes data to transaction log files. Although application performance can be affected by this attribute, transaction durability is not affected. See LogFlushMethod in the *Oracle TimesTen In-Memory Database Reference*.

The `txn.commits.durable` column of the `SYS.SYSTEMSTATS` table indicates the number of transactions that were durably committed.

Avoid Manual Checkpoints

Performance impact: Large

If possible it is best to avoid performing manual checkpoints. You should configure background checkpoints with the `ttCkptConfig` built-in procedure. Background checkpoints are less obtrusive and use non-blocking (or "fuzzy") checkpoints.

Manual checkpoints only apply to TimesTen Classic. Manual checkpoints are not supported in TimesTen Scaleout.

If a manual checkpoint is deemed necessary, it is generally better to call `ttCkpt` to perform a non-blocking (or "fuzzy") checkpoint than to call `ttCkptBlocking` to perform a blocking checkpoint. Blocking or ("transaction-consistent") checkpoints can have a significant performance impact because they require exclusive access to the database. 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 (`ckptLogVolume` parameter of the `ttCkptConfig` built-in procedure) by increasing the amount of file system 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, it is important to properly tune the `ttCkptConfig` built-in procedure. The `ckpt.completed` column of the `SYS.SYSTEMSTATS` table indicates how often checkpoints have successfully completed.

See ttCkptConfig in the *Oracle TimesTen In-Memory Database Reference*.

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.

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 Roll Back

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 roll back transactions to recover from deadlock or timeout conditions. This is not desirable from a performance point of view, as a roll back consumes resources and the entire transaction is wasted.

Applications should avoid unnecessary rollback operations. 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

You should consider avoiding large delete statements.

- Avoid DELETE FROM Statements
- 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 the transaction log records to the file system can be very time-consuming.

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.

See DELETE 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.

Recovery Tuning

You can improve the performance of database recovery after database shutdown or system failure.

- Set RecoveryThreads
- Set CkptReadThreads

Set RecoveryThreads

Performance impact: Large

`RecoveryThreads` defines the number of parallel threads used to rebuild indexes during database recovery time.

For performance reasons, TimesTen does not log changes to indexes. In the event of a crash, all indexes need to be rebuilt. To reduce the index rebuild time, set the `RecoveryThreads` attribute to the number of CPUs to improve recovery performance. See RecoveryThreads in the *Oracle TimesTen In-Memory Database Reference*.

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, 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. This reduces the amount of time it takes to load the TimesTen database to memory by enabling parallel threads to read the TimesTen database checkpoint files.

See CkptReadThreads in the *Oracle TimesTen In-Memory Database Reference*.

Scaling for Multiple CPUs

There are tips for improving performance for multiple CPUs.

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 runs, including options to vary the number of processes that run 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 runs 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 *installation_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 runs 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 the file system, 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 are resource-intensive operations that do not scale well compared to SQL execution operations. Ensure that you use parameterized SQL statements wherever possible and always pre-prepare statements that run more than once.

The `stmt.prepares.count` column of the `SYS.SYSTEMSTATS` table indicates how often statements were prepared directly by an application. If the `stmt.prepares.count` column has a high value, ensure the following:

- Your application uses parametrized statements.
- Your application only prepares any given SQL statement once and not for every single execution.
- Your application does not connect and disconnect frequently from the database. If that is the case, consider using long active connections.

Connect and disconnect operations are resource-intensive operations that do not scale well. Ensure that your application minimizes the number of connect and disconnect operations. Use long active connections with parameterized and re-prepared SQL statements to improve performance.

If your application cannot avoid frequent connect and disconnect operations or you are unable to modify the application to use long-lived connections, consider using connection pooling. Connection pooling can reduce the frequency of connect and disconnect operations and improve the utilization of prepared statements compared to the case where an application frequently connects and disconnects from the database.

Enable 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 enable 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

When multiple connections run the same command, they access common command structures controlled by a single command lock. To avoid sharing their commands and possibly placing contention on the lock, you can set `PrivateCommands=1`.

The use of private commands increases the amount of temporary space used, but provides better scaling at the cost.

See PrivateCommands in the *Oracle TimesTen In-Memory Database Reference*.

XLA Tuning

There are tips on how you can improve 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. See LogBufMB in the *Oracle TimesTen In-Memory Database Reference*.

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

You can improve performance when using cache groups or a replication scheme.

For recommendations on improving performance when using cache groups, see Cache Performance in the *Oracle TimesTen In-Memory Database Cache Guide*.

For recommendations on improving performance for when using a replication scheme, see Improving Replication Performance in the *Oracle TimesTen In-Memory Database Replication Guide*.