Oracle® TimesTen In-Memory Database

Security Guide

Release 26.1
F54451-01
March 2026

Oracle TimesTen In-Memory Database Security Guide, Release 26.1

F54451-01

Contents

About This Content

TimesTen provides security through authentication, authorization and secure network communication.

Audience

This guide is intended for anyone who is interested in security when using TimesTen.

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.

1

Authentication in TimesTen

One aspect of TimesTen access control is authentication of each database user through the use of passwords.

This chapter discusses users and passwords in TimesTen.

> **Note**
>
> Examples in this chapter use the TimesTen `ttIsql` utility, indicated by the `Command>` prompt.

Overview of TimesTen Users

To protect access to a TimesTen database, users must be created with appropriate passwords.

There are these types of users in TimesTen:

- Administrative users: The instance administrator is the user who created the TimesTen instance. The instance administrator must be a member of the TimesTen users group and has full privileges within the instance. See Instance Administrator and Understanding the TimesTen Users Group in *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.

  Other uses can have administrative capabilities by being granted the `ADMIN` privilege. This can be granted by the instance administrator or by another user with `ADMIN` privilege.

  > **Note**
  >
  > See Administrative Privileges.

- TimesTen system users: The system users `SYSTEM` (for internal use), `SYS` (a schema for system objects), and `TTREP` (for replication) are created during TimesTen installation, for internal use only.
- Internal users: An internal user and associated password are defined within a TimesTen database. The user must authenticate with the specified password for access to that database. You can create an internal user with the `CREATE USER` statement.

- External users: An external user is created within the operating system but must be a member of the TimesTen users group. External users are assumed to have been authenticated by the operating system upon login, so there is no stored password within the database. TimesTen uses the operating system credentials of the external user to enable connection to TimesTen as that user. An external user must be identified to the database through the `CREATE USER ... IDENTIFIED EXTERNALLY` statement.

  An external user cannot connect over a TimesTen Client/Server connection unless the client and server are on the same host.

> **Note**
>
> - See Understanding the TimesTen Users Group in *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide* and CREATE USER in *Oracle TimesTen In-Memory Database SQL Reference*.
> - When an external user connects from a Linux or UNIX system, TimesTen converts the user name to upper case, rendering it case-insensitive.

Managing TimesTen Users

There are TimesTen features for managing database users.

- Creating or Identifying a Database User
- Changing the Password of an Internal User
- Providing a Client/Server User and Password
- Dropping a User From the Database

Creating or Identifying a Database User

An instance administrator or a user with the `ADMIN` privilege can create an internal user, identify an external user, or alter a user. These actions can be performed either through a TimesTen direct connection or over an encrypted client-server connection. (See Overview of TimesTen Users in this guide and CREATE USER and ALTER USER in *Oracle TimesTen In-Memory Database SQL Reference*.)
To create an internal user, provide the user name and password in the `CREATE USER` statement. The following example creates the internal user `terry` with the password `secret`:

```
Command> CREATE USER terry IDENTIFIED BY secret;
User created.
```

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

```
Command> CREATE USER pat IDENTIFIED EXTERNALLY;
User created.
```

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

```
Command> ALTER USER pat IDENTIFIED BY secret;
```

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

```
Command> ALTER USER pat IDENTIFIED EXTERNALLY;
```

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

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

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

```
Command> SELECT * FROM sys.user_users;
< PAT, 4, OPEN, <NULL>, <NULL>, USERS, TEMP, 2021-02-25 12:00:17.027100, <NULL>, <NULL> >
1 row found.
```

Note

You can run a `CREATE` or `ALTER USER … IDENTIFIED BY` SQL statement over a client/server connection only when TLS is used. The password is only encrypted when sent over a TLS connection.

See SYS.ALL_USERS, SYS.USER_USERS, and SYS.DBA_USERS in the *Oracle TimesTen In-Memory Database System Tables and Views Reference*.

Changing the Password of an Internal User

An internal user can alter their password through the `IDENTIFIED BY` clause of the `ALTER USER` statement.

A user with the `ADMIN` privilege can alter the password of any user.

For example, to change the password for internal user `TERRY` to `"12345"`:

```
Command> ALTER USER terry IDENTIFIED BY 12345;
User altered.
```

Providing a Client/Server User and Password

The preferred method of specifying a user name and password is by storing both in an Oracle Wallet. However, you can alternatively provide the user name and password in a client DSN or using connection attributes. Providing credentials in a wallet is more secure than supplying a password in a client DSN or on the connection string.

You first set or change a password through `CREATE USER` or `ALTER USER` SQL statements. See Creating or Identifying a Database User.

Once set or changed, you can provide the user and password to the TimesTen server through one of the following methods.

- Providing a User Name and Password in an Oracle Wallet
- Providing a User Name and Password in Connection Attributes

- Providing a User Name and Password in a Client DSN
- Providing a User and Password for TimesTen Utilities

Providing a User Name and Password in an Oracle Wallet

The most secure method to provide credentials when connecting is to store a user's password in an Oracle Wallet. When connecting, you provide the user name and wallet to supply credentials for the connection. Supplying the user name identifies which user's password to retrieve from within the wallet.

There are user-managed and system-managed Oracle Wallets. The system-managed wallets are those that may be created by a user, but are used internally for internal procedures. This section discusses user-managed wallets that are used for connecting to a TimesTen database.

To create a user-managed wallet for providing credentials when connecting:

1. Create a directory to contain your wallet. For example, you could create a directory such as `/wallets` in which your user-managed wallet is stored.
2. The `ttUser` utility requires a full directory path in which to create a new Oracle Wallet or to identify an existing wallet. The name of a wallet cannot be specified. Thus, the wallet is identified by a unique full directory path. Provide the name of the wallet directory created above and a unique name for a subdirectory under it in which to place a single wallet to the `ttUser` utility.

Note

- You can store credentials for multiple users within a single Oracle wallet. For example, you could create a wallet in the `/wallets/dsn1wallet` directory. Multiple users credentials can be added into a wallet identified by `/wallets/dsn1wallet`.
- The credentials from only one user can exist in a wallet. Thus, if you have a single user that has different passwords used to connect to separate DSNs, provide each credential within different wallets. For example, if user Terry has a password to connect to `dsn1` and another password to connect to `dsn2`, then you could add Terry's passwords as appropriate to a wallet in the `/wallets/dsn1wallet` directory and to a wallet in the `/wallets/dsn2wallet` directory. Each wallet would have the appropriate passwords to connect to each DSN.

The `ttUser` utility performs the following:

- If the location does not already exist, TimesTen creates the specified subdirectory and the wallet in the wallet directory location specified. The credentials are added to the Oracle Wallet.
- If the wallet does exist but the user does not exist in the wallet, the `ttUser` utility adds the user and password to the wallet.
- If the user credentials have already been added to an existing wallet, the password is changed for the user name provided.

The following example shows the user creating the `/wallets` directory to contain the wallet. The example assumes that `/wallets/dsn1wallet` does not exist. Thus, the `ttUser` utility creates the `dsn1wallet` subdirectory and then creates the Oracle Wallet in the `/wallets/`

dsn1wallet directory. The ttUser utility prompts for the password for the user terry, which is then added to the wallet.

```
% mkdir /wallets
% ttUser -setPwd -wallet /wallets/dsn1wallet -uid terry
Enter password:
```

The following example shows how to add credentials for user terry into multiple wallets to access multiple TimesTen databases. For example, you would store TimesTen credentials for DSN1 (terry, pwd1) and DSN2 (terry, pwd2) in two separate wallets that exist in separate subdirectories under the wallets directory.

```
$ ttUser -setPwd -wallet /wallets/dsn1wallet -uid terry
Enter password:
$ ttUser -setPwd -wallet /wallets/dsn2wallet -uid terry
Enter password:
```

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

When it's time to authenticate a user to connect to a database, you provide the name of the user and the location of the corresponding wallet by using the UID and PwdWallet connection attributes. The UID connection attribute identifies which user to authenticate using the PwdWallet provided.

```
connect “dsn=mydb;uid=terry;PwdWallet=/wallets/dsn1wallet”;
```

For client/server connections, the wallet must exist on the client. See PwdWallet in the *Oracle TimesTen In-Memory Database Reference*.

You are required to secure and manage all wallets on your client or server. You can move the wallet to the location from which you want to connect. Once you no longer need the user credentials, you can remove these credentials from the wallet with ttUser -removePwd.

If the wallet does not exist or the PwdWallet connection attribute is not specified, then the order of precedence is to look for credentials provided in the connection string and then in the DSN.

Providing a User Name and Password in Connection Attributes

General connection attributes are set by each connection and exist for the duration of the connection. Each concurrent connection can have different values. You can provide the user name and password with the UID, PWD or PWDCrypt general connection attributes.
TimesTen uses the following order of precedence when locating the user name and password for connection authentication:

- An Oracle Wallet with the user name and password. See Providing a User Name and Password in an Oracle Wallet.
- The UID, PWD or PWDCrypt connection attributes provided in the connection string.
- The UID, PWD or PWDCrypt connection attributes provided in the client DSN.

The UID, PWD and PWDCrypt connection attributes are as follows:

- UID: Provides 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 provide a user name. When you do not provide a user name, TimesTen assumes that the UID is the user name identified by the operating system.

- `PWD`: Provides 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`: As an alternative to `PWD`, provides an encrypted password that corresponds with the specified `UID`.

Note

For more information on the `UID`, `PWD` and `PWDCrypt` general connection attributes, see UID and PWD in the *Oracle TimesTen In-Memory Database Reference*. See Authentication in TimesTen in the *Oracle TimesTen In-Memory Database Security Guide*.

Once you have defined the user name and password for a client/server connection, through the `UID` and `PWD` connection attributes, you provide these connection attributes to connect to the database.

- In the connection string.
- In a client DSN in the `odbc.ini` file.

The following example is a connection request to `database1` that provides the user name as `Terry` and the password as `ttpwd` in the connection attributes.

```
% ttIsql "DSN=database1;UID=terry;PWD=ttpwd"
```

Providing a User Name and Password in a Client DSN

You can specify the user name and password in the client DSN.

On Windows, you provide connection attributes in the Oracle TimesTen Client DSN Setup dialog. In this dialog, you can provide the `User ID`, `Password` and `PWDCrypt` connection attributes. If providing your password on this dialog, use either `Password` or `PWDCrypt` connection attributes. See Creating a Client DSN on Windows in the *Oracle TimesTen In-Memory Database Operations Guide*.

On Linux and UNIX, you provide connection attributes in the `odbc.ini` file. In the client DSN in the `odbc.ini` file, you can provide the `UID`, `PWD`, `PWDCrypt`, or `PwdWallet` connection attributes. To provide your password, use only one of the following connection attributes: `Password`, `PWDCrypt`, or `PwdWallet`.

The following is the syntax for the client DSN in the `odbc.ini` file:

```
[ODBC Data Sources]
Client_DSN=TimesTen 26.1 Client Driver
```

See Creating a DSN on Linux and UNIX for TimesTen in the *Oracle TimesTen In-Memory Database Operations Guide*.

Providing a User and Password for TimesTen Utilities

You can provide the user name and password in an Oracle Wallet, the connection attributes, or in the `odbc.ini` file.

If a TimesTen utility takes a connection string, then you can provide the user name and password in an Oracle Wallet. Instead of providing a `UID` and `PWD` connection attribute on the command line, provide the `PwdWallet` connection attribute with the location and name of the wallet.

If the `UID` connection attribute setting is provided for a TimesTen utility but no `PWD` attribute setting is provided, either in the connection string or the `odbc.ini` file, TimesTen prompts for a password.

See UID and PWD in *Oracle TimesTen In-Memory Database Reference*.

Note

- When you enter a password at the prompt, what you type is not shown.
- It is not advisable to specify a value for `PWD` on the command line.

Dropping a User From the Database

An instance administrator or a user with the `ADMIN` privilege can use the `DROP USER` statement to remove an internal or external user from the database. See DROP USER in *Oracle TimesTen In-Memory Database SQL Reference*.
For example:

```
Command> DROP USER terry;
User dropped.
```

Note

- You cannot drop a user who is still connected to the database or before all database objects owned by the user have been deleted.
- TimesTen does not support `DROP USER CASCADE`.

Cache Group Users

There are required users when using cache.

This section covers these topics regarding cache group users:

- Required Users for Cache
- Providing Both Cache Administration Users and Passwords
- #unique_30

Required Users for Cache

To use cache, you must create administration and schema users on both the Oracle and TimesTen databases.

To use cache, you must have the following users on the Oracle Database:

- Create an Oracle cache administration user who creates, owns, and maintains Oracle Database objects that store information used to manage the cache environment for a TimesTen database and enforce predefined behaviors of particular cache group types.
- Identify one or more schema users who own the Oracle Database tables to be cached in a TimesTen database.

To use cache, you must create the following users on the TimesTen database:

- A TimesTen cache administration user who performs cache group operations. The TimesTen cache administration user must have the same user name as the Oracle cache administration user created for cache who can access the cached Oracle Database tables. The password of the TimesTen cache administration user can be different from the password of the companion Oracle cache administration user.
- One or more cache table users who own the cache tables. You must create a TimesTen cache table user with the same user name as each Oracle Database schema user who owns Oracle Database tables to be cached in the TimesTen database. The password of a cache table user can be different from the password of the Oracle Database schema user with the same name.

  The owner and name of a TimesTen cache table is the same as the owner and name of the corresponding cached Oracle Database table.

Providing Both Cache Administration Users and Passwords

If you are running a request that does not require access to the Oracle database, you can proceed without needing to provide credentials for the Oracle database. That is, you can connect with only the user name and password for connecting to the TimesTen database. However, when you want to perform an action that requires connecting to the Oracle database, then you must provide the appropriate credentials to be able to connect to both the TimesTen and Oracle databases.
You first create or change a cache administration user and its password through `CREATE USER` or `ALTER USER` SQL statements. See Creating or Identifying a Database User.

Once the cache administration users are created with their respective passwords, these credentials need to be provided with one of the following methods.

- Providing the Cache Administration User Names and Passwords in an Oracle Wallet
- Providing Cache Administration User Name and Passwords in Connection Attributes
- Providing Cache Administration User Name and Passwords in a Client DSN

Providing the Cache Administration User Names and Passwords in an Oracle Wallet

The most secure method to provide credentials when connecting is to store a user's password in an Oracle Wallet. When connecting, you provide the user name and wallet to supply credentials for the connection. Supplying the user name identifies which user's password to retrieve from within the wallet.

You can store existing credentials for both the cache user and the cache administration user and their associated passwords within an Oracle Wallet using the `ttUser` utility.

- For the cache user, you can add this user's password to a wallet in the same manner as a TimesTen user as described in Providing a User Name and Password in an Oracle Wallet.
- To connect as the cache administration users, you must provide the passwords for both the TimesTen cache administration user and the Oracle cache administration user.

See Providing a User Name and Password in an Oracle Wallet for full details on how to store credentials in an Oracle Wallet. This section describes the process to add both cache administration user passwords to an Oracle Wallet.

You can add the cache administration users passwords to a wallet used by other users, such as a wallet that contains all credentials for those connecting to a DSN. Alternatively, you could create a wallet only for the cache administration users.

Use the `ttUser -setPwd` command to store the password for the TimesTen cache administration user. Use the `ttUser -setOraclePwd` command to store the password for the Oracle cache administration user.

The following example shows how to use the `ttUser` utility to add both cache administration users to an Oracle Wallet in the `/wallets/cacheadminwallet` directory.

1. If it does not already exist, make a directory for your wallet. This example users `/wallets` as the directory for the wallet.

   ```
   % mkdir /wallets
   ```

2. Run the `ttUser -setPwd` command to store the TimesTen cache administration user credentials. Provide a subdirectory name that identifies the wallet (since you cannot change the name of an OracleWallet). This example provides `cacheadminwallet` as the subdirectory name for the wallet. If `cacheadminwallet` directory does not exist, then the `ttUser` utility creates the `cacheadminwallet` subdirectory and then creates the Oracle Wallet in the `/wallets/cacheadminwallet` directory. The `ttUser` utility prompts for the password for the TimesTen cache administration user `cacheadmin`, which is added to the wallet.

   ```
   % ttUser -setPwd -wallet /wallets/cacheadminwallet -uid cacheadmin
   Enter password:
   ```

3. Run the `ttUser -setOraclePwd` command to store the Oracle cache administration user credentials. The `ttUser` utility prompts for the password for the Oracle cache administration user `cacheadmin`, which is added to the wallet in `/wallets/cacheadminwallet`.

   ```
   % ttUser -setOraclePwd -wallet /wallets/cacheadminwallet -uid cacheadmin
   Enter password:
   ```

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

When it's time to authenticate the cache administration users when connecting to a database, provide the name of the cache administration user and the location of the corresponding wallet with the `UID` and `PwdWallet` connection attributes. The `UID` connection attribute specifies which user to authenticate using the `PwdWallet` provided.

```
connect “dsn=mydb;uid=cacheadmin;PwdWallet=/wallets/cacheadminwallet”;
```

Providing Cache Administration User Name and Passwords in Connection Attributes

General connection attributes are set by each connection and exist for the duration of the connection. Each concurrent connection can have different values. You can provide both cache administration user names and passwords with the `UID`, `PWD` and `OraclePWD` general connection attributes.
Once you have created both cache administration users and associated passwords, you can specify them on a connection string with the `UID`, `PWD`, and `OraclePWD` connection attributes when connecting to the database.

TimesTen uses the following order of precedence when locating the user name and password for connection authentication:

- An Oracle Wallet with the cache administration user name and passwords. See Providing the Cache Administration User Names and Passwords in an Oracle Wallet.
- The `UID`, `PWD` and `OraclePWD` connection attributes provided in the connection string.
- The `UID`, `PWD` and `OraclePWD` connection attributes provided in the client DSN.

The `UID`, `PWD` and `OraclePWD` connection attributes are as follows:

- `UID`: In this case, specifies the cache administration user name to be used for the connection to the database.
- `PWD`: In this case, specifies the password for the TimesTen cache administration user.
- `OraclePWD`: Specifies the password for the Oracle cache administration user.

Note

For more information on the `UID`, `PWD` and `OraclePWD` general connection attributes, see UID and PWD in the *Oracle TimesTen In-Memory Database Reference*. See Authentication in TimesTen in the *Oracle TimesTen In-Memory Database Security Guide*.

The following example is a connection request to `database1` that provides the cache administration user name as `cacheadmin`, the TimesTen cache administration user password as `ttpwd`, and the Oracle cache administration user password as `orapwd`.

```
% ttIsql "DSN=database1;UID=cacheadmin;PWD=ttpwd;OraclePWD=orapwd"
```

Providing Cache Administration User Name and Passwords in a Client DSN

You can provide both cache administration user names and passwords in the client DSN.

On Windows, you specify connection attributes in the Oracle TimesTen Client DSN Setup dialog. In this dialog, you can specify the `User ID`, and `Password` connection attributes. However, the `OraclePWD` connection attribute can only be specified on the connection string. See Creating a Client DSN on Windows in the *Oracle TimesTen In-Memory Database Operations Guide*.

On Linux and UNIX, you specify connection attributes in the `odbc.ini` file. In the client DSN in the `odbc.ini` file, you can specify the `UID`, `PWD` and `OraclePWD` connection attributes.

The following is the syntax for the client DSN in the `odbc.ini` file:

```
[ODBC Data Sources]
Client_DSN=TimesTen 26.1 Client Driver
```

See Creating a DSN on Linux and UNIX for TimesTen in the *Oracle TimesTen In-Memory Database Operations Guide*.

Registering the Oracle Cache Administration User and Password

One of the prerequisites to setting up your cache environment is registering the Oracle cache administration user and password in TimesTen. TimesTen uses these credentials to connect to the Oracle database. You can register with TimesTen Classic the Oracle cache administration user name and password.

There are cache operations that TimesTen performs for you. In order for TimesTen to connect to the Oracle database successfully to perform these cache operations, TimesTen needs to have the Oracle cache administration user and password credentials registered internally. This is accomplished when you run either the `ttCacheUidPwdSet` built-in procedure. By default, the Oracle cache administration user and password are stored in memory. You can specify that the Oracle cache administration user and passwords are saved in a system-managed Oracle Wallet (preferred) by setting the `CacheAdminWallet=1` in the DSN as a first connection attribute. Once the Oracle cache administration user and password are registered (either in memory or in a system-managed wallet), TimesTen uses the credentials to connect to the backend Oracle database for cache operations. See CacheAdminWallet in the *Oracle TimesTen In-Memory Database Reference*.

1. Ensure that the `CacheAdminWallet` first connection attribute is set to 1.
2. Start the `ttIsql` utility and connect to the `cache1` DSN (for example) as the TimesTen cache administration user. Provide the cache administration user name and passwords when connecting using one of the methods detailed in Providing Both Cache Administration Users and Passwords.

   ```
   % ttIsql "DSN=cache1;UID=cacheadmin;PwdWallet=/wallets/cacheadminwallet"
   ```
3. Use the `ttCacheUidPwdSet` built-in procedure (only once) to register the TimesTen database of the Oracle cache administration user name and password in the Oracle database. Since `CacheAdminWallet=1`, the Oracle cache administration user name and password are stored in a system-managed Oracle Wallet.

   The Oracle cache administration user name is `cacheadmin` and its password is `orapwd`.

   ```
   Command> call ttCacheUidPwdSet('cacheadmin','orapwd');
   ```

Note

You can run the `ttCacheUidPwdSet` built-in procedure over a client/server connection only when TLS is used. The password is only encrypted when sent over a TLS connection.

See Setting Up a Caching Infrastructure in *Oracle TimesTen In-Memory Database Cache Guide* and Privileges for Cache Groups in this guide.

Note

Alternatively, you can use `ttAdmin` to set the Oracle cache administration user ID and password. See Set Cache Policies in *Oracle TimesTen In-Memory Database Reference*. For example:

```
% ttAdmin -cacheUidPwdSet -cacheUid cacheadmin -cachePwd orapwd database1
```

You can use the `ttCacheUidPwdSet` built-in procedure to later change the Oracle cache administration password at any time, or change the Oracle cache administration user name (and optionally the password as well) as long as there are no existing cache groups.

Prometheus Exporter Authentication

In order to monitor database health and operation, TimesTen collects metrics from a variety of sources. TimesTen Prometheus Exporter converts these metrics into the form used by Prometheus Monitoring. This integration allows customers to add TimesTen to the systems that they monitor with Prometheus.

Prometheus includes its own time-series database and time-series query language. You can use Prometheus directly to construct near real-time graphs of metrics or to create programmable alerts.

The TimesTen exporter, implemented as the `ttExporter` utility, is not configured to run by default. It supports client certificate authentication (mutual TLS) or no authentication. While it is typical for Prometheus exporters to operate with no security, the default configuration of the TimesTen exporter is client certificate authentication. An Oracle Wallet is used to store TLS and TimesTen login credentials.

See The TimesTen Prometheus Exporter in *Oracle TimesTen In-Memory Database Monitoring and Troubleshooting Guide* and ttExporter in *Oracle TimesTen In-Memory Database Reference*.

Password Management

You can manage passwords to increase the level of security that can be implemented for authentication.

This section provides an overview of password management in TimesTen.

- Password Management Features
- Profile for Password Management

Password Management Features

Password management features can enhance the security of your TimesTen database.

- Password Lifetime and Grace Time
- Limitations on Password Reuse
- Maximum Failed Login Attempts and Password Lock Time
- Password Requirements
- Password Complexity Checker

Password Lifetime and Grace Time

You can limit how long a user can continue to use the same password before it expires, as well as a grace period after that period of time. During the grace period, the password is still allowed and recognized, but with a warning.

Limitations on Password Reuse

While limiting password lifetimes enhances system security, allowing users to frequently reuse previous passwords diminishes the effectiveness.

When a user is changing their password, you can specify:

- A minimum period of time that must pass before a previous password can be reused.
- The number of password changes that must occur before a previous password can be reused.

Both of these must be satisfied before a password can be reused. For example, if `PASSWORD_REUSE_TIME` is 30 and `PASSWORD_REUSE_MAX is 10`, the user can reuse a password after 30 days if it is not one of the last 10 passwords used.

If one or the other is set to unlimited, a password can never be reused, but if both are set to unlimited, there are no limits on how often a password can be reused.

Maximum Failed Login Attempts and Password Lock Time

Hackers may try to access TimesTen by repeatedly guessing passwords until one works. You can limit the number of failed attempts that are allowed and how long the account is locked after this maximum number is reached.

Password Requirements

The following lists the password requirements:

- Cannot exceed 30 characters.
- Is case-sensitive.
- Must start with a letter. A password cannot start with a digit or a special character unless the password is enclosed in double quotation marks.
- If a special character is used, the password must be contained in double quotation marks. The exceptions are the `#` and the `@` special characters. A password that contains the `#` or the `@` special character does not need to be enclosed in double quotation marks.
- Cannot contain a semi-colon (`;`) or a double quotation mark (").

See CREATE USER in the *Oracle TimesTen In-Memory Database SQL Reference*.

Password Complexity Checker

TimesTen offers a set of PL/SQL functions you can choose from to test for sufficient password complexity. This functionality helps ensure that user passwords are stringent enough to impose the desired level of protection for your system.

These functions are provided:

- `TT_VERIFY_FUNCTION` (basic protection)
- `TT_STRONG_VERIFY_FUNCTION` (stronger protection)
- `TT_STIG_VERIFY_FUNCTION` (protection according to the Department of Defense Database Security Technical Implementation Guide)

Checks are run against passwords newly specified through the `CREATE USER` or `ALTER USER` statement. If the password does not have sufficient complexity, the statement fails with an error.

You can specify a password complexity verification function when you create or alter a user profile with the `CREATE PROFILE` or `ALTER PROFILE` statement. Set the `PASSWORD_COMPLEXITY_CHECKER` parameter to the desired function, or to `NULL` for no complexity checking, or to `DEFAULT` to set complexity checking according to the `DEFAULT` user profile (`NULL` by default). Then specify that profile when you create or alter a user through the `CREATE USER` or `ALTER USER` statement.

Refer to CREATE PROFILE in *Oracle TimesTen In-Memory Database SQL Reference*.

Note

- TimesTen does not support user-defined password complexity functions.
- The `CREATE PROFILE` or `ALTER PROFILE` parameter `PASSWORD_VERIFY_FUNCTION` is equivalent to `PASSWORD_COMPLEXITY_CHECKER`.

Profile for Password Management

TimesTen employs profiles to specify settings of the password management parameters.

TimesTen employs profiles for the features described in the preceding section: `PASSWORD_LIFE_TIME`, `PASSWORD_GRACE_TIME`, `PASSWORD_REUSE_TIME`, `PASSWORD_REUSE_MAX`, `FAILED_LOGIN_ATTEMPTS`, and `PASSWORD_LOCK_TIME`.

The same profile can be used for multiple users, and there is a default profile. A user who is not assigned a profile will use the default profile. Also, a setting of `DEFAULT` for any parameter in a profile will result in use of the value from the default profile.

The `CREATE PROFILE` SQL statement creates a profile. Specify `PROFILE` in a `CREATE USER` statement to assign an existing profile to a user.

See CREATE PROFILE, ALTER PROFILE, CREATE USER, or ALTER USER in *Oracle TimesTen In-Memory Database SQL Reference*.

2 Authorization in TimesTen

One aspect of TimesTen access control is the use of permissions, or privileges, to authorize or limit access to database objects such as tables or views.

This chapter discusses TimesTen features for authorization, covering these topics:

- Privileges Overview
- System Privileges
- Overview of Privileges to Create, Alter, or Drop Objects
- Privileges for SQL Objects
- Privileges for PL/SQL Objects
- Privileges for Cache Groups
- User Privilege Views

Note

- For a list of object privileges, see Privileges in *Oracle TimesTen In-Memory Database SQL Reference*.
- For TimesTen SQL statements discussed in this chapter, syntax and required privileges are documented in SQL Statements in *Oracle TimesTen In-Memory Database SQL Reference*.
- Examples in this chapter use the TimesTen `ttIsql` utility, indicated by the `Command>` prompt.

Privileges Overview

TimesTen allows access to objects in the database according to authorization through the granting of privileges. These privileges determine what operations users may perform.

This section covers these topics:

- About Privileges
- Granting and Revoking Privileges
- Functionality of Privileges
- Overview of System Privileges
- Overview of Object Privileges
- Privileges for TimesTen Utilities
- Overview of the PUBLIC Role

About Privileges

When there are multiple users who could potentially access database objects, access to these objects is authorized according to the granting of privileges.

Every object has an owner. Object privileges authorize a user to access or modify an object owned by another user. Privileges are granted or revoked either by the instance administrator, a user with the `ADMIN` privilege, or, for privileges to a certain object, by the owner of the object.

There are also system level privileges to authorize actions such as connecting to the database.

Note

A user has all privileges on all objects that they own, and these privileges cannot be revoked.

Granting and Revoking Privileges

Use the SQL `GRANT` statement to grant privileges to allow a user to access a particular object, objects, or types of objects. Use the SQL `REVOKE` statement to revoke privileges.

You must have administrative privilege to grant or revoke system privileges or to grant or revoke object privileges for an object you do not own.

Examples:

```
GRANT admin TO terry;
GRANT SELECT ON pat.customers TO terry;
GRANT SELECT ON emp_details_view TO terry;

REVOKE admin, ddl FROM terry;
REVOKE update ON pat.customers FROM terry;
```

See GRANT and REVOKE in *Oracle TimesTen In-Memory Database SQL Reference*.

Functionality of Privileges

TimesTen evaluates each user's privileges when a SQL statement is executed.

For example:

```
Command> SELECT * from pat.table1;
```

If this statement is executed by `pat`, then no extra privileges are necessary because `pat` owns this object. However, before another user, such as `terry`, executes this statement, they must be granted the `SELECT` privilege for `pat.table1`:

```
Command> GRANT SELECT ON pat.table1 TO terry;
```

Privileges accomplish the following:

- They define what data users, applications, or functions can access or what operations they can perform.
- They prevent users from adversely affecting system performance or from consuming excessive system resources. For example, a privilege restricting the creation of indexes is

provided not because of an authorization concern, but because it may affect DML performance and occupies space.

Some examples of privileges include authorization to perform the following:

- Connect to the database and create a session
- Create a table
- Select rows from a table
- Perform cache group operations

There are two levels of privileges:

- System privileges enable system-wide functionality, such as access to all objects. Granting system privileges can enable a user to perform administrator tasks or access objects in other users' schemas. Grant them only to trusted users. See Overview of System Privileges.
- Object privileges enable access to a specific database object, such as a particular table or view. See Overview of Object Privileges

A subset of these privileges are automatically granted to each user upon creation through the `PUBLIC` role. See Overview of the PUBLIC Role.

Privileges are checked when a SQL statement is prepared and the first time it is executed. Subsequent executions of the statement require further privilege checks only if a `REVOKE` statement has been executed in the database.

Overview of System Privileges

A system privilege authorizes a user to perform system-level activities across the database or perform a specified type of operation for all database objects of a specified type (for example, `CREATE ANY TABLE`).

Examples of system privileges are `ADMIN`, `SELECT ANY TABLE`, `CREATE SESSION` and `CREATE ANY SEQUENCE`. See System Privileges.

Only the instance administrator or a user with the `ADMIN` (administrative) privilege can grant a system privilege to a user.

Note

A user with `ADMIN` privileges has a special set of system privileges, as discussed in Administrative Privileges. The instance administrator has an all-encompassing set of system privileges, as covered in Instance Administrator.

Overview of Object Privileges

An object privilege enables a user to perform a specific operation on a specific object. Separate object privileges are available for each object type, such as `CREATE TABLE`.

A user does not have access to objects owned by other users unless explicitly granted access by the object's owner or by a user with `ADMIN` privilege.

If the `PUBLIC` role has been granted access to a given object, then all database users have access to that object.

Object privileges are granted or revoked by the instance administrator, a user with the `ADMIN` privilege, or the user who owns the object.

See Privileges for SQL Objects.

Privileges for TimesTen Utilities

Sometimes special privileges are required to run a TimesTen utility.

Any special privilege required to run a TimesTen utility is noted under "Required privilege" in the description of the utility in Utilities in *Oracle TimesTen In-Memory Database Reference*.

Note

If any user other than the instance administrator tries to run a utility that requires special privilege when the database is not loaded into memory, they will receive an error because TimesTen cannot determine the privilege of the user.

Overview of the PUBLIC Role

A role called `PUBLIC` is automatically created in each TimesTen database and given specific privileges, and each user created in a TimesTen database inherits these privileges. Each subsequent privilege that is also granted to the `PUBLIC` role is also automatically granted to all users simultaneously.

For example, this command results in `CREATE SESSION` privilege for all users:

```
Command> GRANT CREATE SESSION TO PUBLIC;
```

Also see Privileges Through the PUBLIC Role in this document and The PUBLIC Role in *Oracle TimesTen In-Memory Database SQL Reference*.

Note

TimesTen does not support any other roles.

System Privileges

There are system privileges available in TimesTen.

- About System Privileges
- Instance Administrator
- Administrative Privileges
- Privileges to Connect to the Database
- ANY Keyword
- ALL PRIVILEGES
- Privilege Hierarchy
- Additional System Privileges

- Privileges Through the PUBLIC Role

About System Privileges

Aside from the instance administrator, the most powerful system privilege is `ADMIN`, which enables the user to perform system operations or operations on any database object. Only the instance administrator or a user with the `ADMIN` privilege can grant or revoke system privileges to other users.

An individual user can view their own system privileges in the `SYS.USER_SYS_PRIVS` system view. A user with the `ADMIN` privilege can view all system privileges for all users in the `SYS.DBA_SYS_PRIVS` system table. See User Privilege Views.

Instance Administrator

The instance administrator, a member of the TimesTen users group, is the user who creates the TimesTen installation and all TimesTen instances. This user has a number of special privileges and capabilities beyond those of other administrative users.

These are described in the following sections:

- Instance Administrator Privileges
- Instance Administrator Ownership and Privileges for Database and Log Directories

Instance Administrator Privileges

There are privileges that only the instance administrator can do.

- Remove the TimesTen installation.
- Create, modify (including upgrade), or destroy a TimesTen instance.
- Create or destroy a database.
- Load or unload a database manually (`ramPolicy manual` using `ttAdmin -ramLoad`).
- Load a database when changes to first connection attribute settings are applied.
- Open or close a database.
- Restore a database.
- Start and stop the TimesTen daemon.
- Restart the TimesTen server.

See the Instance Administrator, Understanding the TimesTen Users Group, and TimesTen Instances in *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*

-

Note

- The instance administrator cannot be the root user.
- You cannot change to a different instance administrator.

Instance Administrator Ownership and Privileges for Database and Log Directories

The instance administrator owns the database directory (indicated by the `DataStore` connection attribute), where checkpoint files are written, and the log directory (indicated by the `LogDir` connection attribute).

Proper ownership and permissions must be set for these directories. In addition to the owner being the instance administrator, the group must be the TimesTen users group and the directory permissions must be set for read/write/execute permission for owner and group with no access by anyone else.

Administrative Privileges

The `ADMIN` privilege confers system privileges and privileges on all database objects, which enables these users to perform administrative tasks and any valid database operation. Only the instance administrator or another user with `ADMIN` privilege can grant `ADMIN` privilege.

A user with the `ADMIN` privilege can do the following:

- Perform create, alter, drop, select, update, insert, or delete operations on all database objects.
- Grant or revoke all privileges.
- Perform checkpointing operations.
- Create and delete users.
- View system tables, views, and packages.
- Create, alter or drop replication schemas or active standby pairs.

Note

For more information on viewing privileges for users from system tables or views, see User Privilege Views.

To grant the `ADMIN` privilege to the user `terry`, the instance administrator or another user with `ADMIN` privilege executes this statement:

```
GRANT ADMIN TO terry;
```

To grant the `SELECT` privilege to `terry` on the `departments` table owned by `pat`:

```
GRANT SELECT ON pat.departments TO terry;
```

Note

Since `pat` is the owner of departments, `pat` may also grant the `SELECT` object privilege to `terry`.

Privileges to Connect to the Database

A user must be granted the `CREATE SESSION` system privilege by the instance administrator or a user with the `ADMIN` privilege in order to connect to the database.

The following example grants the `CREATE SESSION` privilege to `pat`:

```
Command> GRANT CREATE SESSION TO pat;
```

Note

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

ANY Keyword

Privileges used with the `ANY` keyword enable the user to perform the operation on any object of the specified type in the database.

These system privileges are `CREATE ANY` *`object_type`*, `DROP ANY` *`object_type`*, `ALTER ANY` *`object_type`*, `SELECT ANY` *`object_type`*, `UPDATE ANY TABLE`, `INSERT ANY TABLE`, `DELETE ANY TABLE`, and `EXECUTE ANY PROCEDURE`.

`ANY TABLE` also includes views and materialized views.

ALL PRIVILEGES

`ALL PRIVILEGES`, which can be granted by the instance administrator or a user with `ADMIN` privilege, grants system privileges to a user.

If you want to limit the privileges granted, you can grant `ALL PRIVILEGES` then revoke those system privileges that you do not want the user to have.

Once granted, `ALL PRIVILEGES` can subsequently be revoked.

Privilege Hierarchy

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

When a user needs a privilege for an operation, first verify whether the user already has the privilege through a higher level privilege. For example, if the user `pat` needs to have the `SELECT` privilege for `terry.table2`, you can check the following:

- Has `pat` been granted the `SELECT ANY TABLE` privilege? This privilege means `pat` would have `SELECT` on any table, view, or materialized view.
- Has `pat` been granted the `ADMIN` privilege? This would mean that `pat` can perform any valid SQL operation.

If you grant a privilege that is included in a higher level privilege, no error occurs. However, when you revoke privileges, they must be revoked in the same unit as granted (ANY level or object level).

The following series of statements is allowed, and `pat` can still update the `hr.employees` table because of the `UPDATE ANY TABLE` privilege. (The second statement of course is unnecessary, but the third statement would not be allowed without it.)

```
Command> GRANT UPDATE ANY TABLE TO pat;
Commanc> GRANT UPDATE ON hr.employees TO pat;
Command> REVOKE UPDATE ON hr.employees FROM pat;
```

This next example also leaves `pat` with the ability to update `hr.employees`, because that was granted explicitly:

```
Command> GRANT UPDATE ANY TABLE TO pat;
Commanc> GRANT UPDATE ON hr.employees TO pat;
Command> REVOKE UPDATE ANY TABLE FROM pat;
```

The following example attempts to revoke the ability to update the `hr.employees` table from the user, but is not allowed because there was no `GRANT` statement for that specific object.

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

See Privilege Hierarchy in *Oracle TimesTen In-Memory Database SQL Reference*.

Additional System Privileges

In addition to the `ADMIN` privilege, some system privileges authorize a range of operations across certain areas of database functionality.

- `XLA`: You must have the `XLA` system privilege to connect as an XLA reader, who can have global impact on the system. An XLA reader can create extra log volume and can cause long log holds if they do not advance their bookmarks.
- `CACHE_MANAGER`: The `CACHE_MANAGER` privilege is required for cache group administrator operations. See Privileges for Cache Groups.

Privileges Through the PUBLIC Role

The instance administrator or a user with the `ADMIN` privilege can grant or revoke default privileges for all users by granting or revoking privileges for the `PUBLIC` role.

Note

- If a user has been explicitly granted a privilege, it is not revoked if that privilege is revoked from `PUBLIC`.
- Any privileges that were granted to `PUBLIC` by user `SYS` cannot be revoked. These privileges, granted as part of database creation, are shown when you execute the following SQL statement:

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

In the following example, user `pat` is granted the `SELECT ANY TABLE` privilege and `PUBLIC` is granted the `SELECT ANY TABLE` privilege. Then all system privileges are displayed from the `SYS.DBA_SYS_PRIVS` view. (See User Privilege Views.) As shown, revoking `SELECT ANY TABLE` from `PUBLIC` does not revoke `SELECT ANY TABLE` from `pat`. (The second column indicates a privilege held by the user. The third column, `NO` in the example, indicates whether the user can grant that privilege to others.)

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

By default in a newly created TimesTen database, `PUBLIC` has `SELECT` and `EXECUTE` privileges on various system tables and views and PL/SQL functions, procedures and packages. You can see the list of privileges granted to `PUBLIC` by querying the `SYS.DBA_TAB_PRIVS` view. In the query below, the privilege granted to `PUBLIC` is in the fifth column, as indicated by the `DESCRIBE` statement that precedes the query.

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

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

```
...
57 rows found.
```

Overview of Privileges to Create, Alter, or Drop Objects

There are privileges that are required in order to create, alter, or drop database objects.

- Privileges to Create Database Objects
- Privileges to Alter Database Objects
- Privileges to Drop Database Objects

Privileges to Create Database Objects

To create a database object such as a table, view, materialized view, sequence, PL/SQL procedure, PL/SQL function, PL/SQL package, or synonym, you must have the appropriate `CREATE` *`object_type`* or `CREATE ANY` *`object_type`* privilege.

The following describes the `CREATE` and `CREATE ANY` privileges:

- The `CREATE` *`object_type`* privilege grants a user the ability to create an object of the specified type (such as `TABLE`), but only in the user's own schema. After creation, the user owns the object and has all privileges for the object.
- The `CREATE ANY` *`object_type`* privilege grants a user the ability to create any object of that type in any schema of the database. The `CREATE ANY` *`object_type`* privileges are `CREATE ANY TABLE`, `CREATE ANY INDEX`, `CREATE ANY VIEW`, `CREATE ANY MATERIALIZED VIEW`, `CREATE ANY SEQUENCE`, `CREATE ANY SYNONYM` and `CREATE ANY PROCEDURE`.

A user must be granted `CREATE TABLE` privilege to create a table in their schema, as in this example:

```
Command> GRANT CREATE TABLE TO terry;
```

This example grants the privilege to create any table in any schema to user `terry`:

```
Command> GRANT CREATE ANY TABLE TO terry;
```

Note

- See Object Privileges for Views and Object Privileges for Materialized Views.
- When `CREATE OR REPLACE` results in an object (such as a procedure, function, package, or synonym) being replaced, there is no effect on privileges that any users had previously been granted on that object. This is as opposed to when there is an explicit `DROP` and then `CREATE` to re-create an object, in which case all privileges on the object are revoked.

Privileges to Alter Database Objects

The `ALTER ANY` *`object_type`* privilege is necessary to modify the properties of objects that the user does not own.

For example, if a procedure `proc1` is created in the `hr` schema and `pat` is granted the `ALTER ANY PROCEDURE` privilege, `pat` can alter the procedure `hr.proc1`.

The `ALTER` privilege cannot be granted on an individual object. Instead, you must grant the `ALTER ANY` privilege for the desired object type.

Privileges to Drop Database Objects

The `DROP ANY` *`object_type`* privilege enables a user to drop any object of the specified type in the database and is necessary to drop an object of *`object_type`* that the user does not own.

For example, granting `pat` the `DROP ANY TABLE` privilege enables `pat` to drop the `employees` table that is owned by the user `hr`.

The `DROP` privilege cannot be granted on an individual object. Instead, you must grant the `DROP ANY` privilege for the desired object type.

Privileges for SQL Objects

User access to database objects is authorized by granting privileges, either for a single object or for that type of object anywhere in the database, through the `GRANT` statement. Access is removed through the `REVOKE` statement.

This section covers the following:

- Object Privileges for Tables
- Object Privileges for Views
- Object Privileges for Sequences
- Object Privileges for Materialized Views
- Object Privileges for Synonyms
- ALL Object Privileges

Note

Also, see Privileges for PL/SQL Objects.

Object Privileges for Tables

For a user to create a table, that user must be granted the `CREATE TABLE` or `CREATE ANY TABLE` privilege.

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

For example:

```
Command> GRANT SELECT ON hr.employees TO pat;
Command> GRANT UPDATE ON hr.employees TO pat;
```

The `INDEX` privilege enables a user to create an index on the table.

The REFERENCES privilege enables use of the REFERENCES clause in the CREATE TABLE or ALTER TABLE statement. This clause creates a foreign key dependency from a child table column (in the following example, table1.col1) to a parent table column (in the example, table2.pk).

```
Command> ALTER TABLE pat.table1 ADD CONSTRAINT fk1 FOREIGN KEY (col1)
REFERENCES pat.table2 (pk);
```

If pat, owner of the tables, executes the statement, no additional privileges are needed. Any other user executing the statement would need ALTER ANY TABLE privilege.

In addition, if the user executing an ALTER TABLE ... REFERENCES statement does not own the table referenced by the REFERENCES clause, then REFERENCES object privilege on the applicable table column is required. For example, for pat to execute this statement:

```
Command> ALTER TABLE pat.table1 ADD CONSTRAINT fk1
FOREIGN KEY (col1) REFERENCES terry.table2 (pk);
```

Pat would need the following privilege grant:

```
Command> GRANT REFERENCES (pk) ON terry.table2 TO pat;
```

Note that the REFERENCES privilege implicitly grants SELECT privilege for a user creating a foreign key from the parent table. However, this implicit grant does not mean that the user has the SELECT privilege on the parent table, so any SELECT statements fail if the only privilege on the parent table is REFERENCES.

Note

If you have tables related by foreign key constraints, these notes apply:

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

Object Privileges for Views

For a user to select from a view that they do not own, they need to be granted the SELECT object privilege for that view. Furthermore, the owner of the view must have the SELECT object privilege for all of the objects referenced by the view.

For user pat to create a view that references only objects owned by pat, as in the statement that follows, then pat needs only the CREATE VIEW privilege.

```
Command> CREATE VIEW pat.view1 AS SELECT * FROM pat.table1;
```

For pat to create a view that references a table owned by terry, as in the statement that follows, then pat also needs the SELECT object privilege on that table. The owner of a view must be granted the SELECT object privilege on each object referenced by the view.

```
Command> CREATE VIEW pat.view2 AS SELECT * FROM terry.table2;
```

For a third user, joe, to execute the preceding statement, joe needs the CREATE ANY VIEW privilege. And pat, as the owner of the view, still must have been granted the SELECT object privilege in order to perform the select on the table that terry owns.

When you select from a view, TimesTen validates the view at execution time, as well as any views referenced by that view, for the required underlying privileges.

Now consider the following example:

```
Command> CREATE VIEW pat.view2 AS SELECT * from terry.table2;
Command> CREATE VIEW joe.view4 AS SELECT * from pat.view2, terry.table4;
```

For pat to execute these statements, the following privileges must be granted:

- User pat must be granted the CREATE ANY VIEW privilege so pat can create a view in the schema owned by joe.
- User joe must be granted the SELECT object privilege on terry.table4.
- User joe must be granted the SELECT object privilege on pat.view2
- User pat must be granted the SELECT object privilege on terry.table2

Object Privileges for Sequences

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

For example, to grant SELECT privilege on the employees_seq sequence in the hr schema to the user pat:

```
Command> GRANT SELECT ON hr.employees_seq TO pat;
```

User pat can subsequently generate the next value of the sequence with the following statement:

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

Object Privileges for Materialized Views

To create a materialized view, a user needs at least the CREATE MATERIALIZED VIEW privilege. To create a materialized view in another user's schema, the CREATE ANY MATERIALIZED VIEW privilege is required.

Additionally, the owner of the materialized view needs to have CREATE TABLE privilege as well as SELECT privilege on every detail table in that materialized view. If the owner of an existing materialized view loses the SELECT privilege on any detail table on which the materialized view is based, the materialized view becomes invalid.

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

Note

The status of a materialized view is indicated in the `STATUS` column of the `SYS.DBA_OBJECTS`, `SYS.ALL_OBJECTS`, and `SYS.USER_OBJECTS` views. The owner of the materialized view can see its status in the `USER_OBJECTS` view.

Also, if a materialized view is invalid, the `ttIsql describe` output appends `INVALID` for the materialized view.

Furthermore, regarding materialized views:

- Users that have the privilege to do so can still update the detail tables of the materialized view. However, an invalid materialized view does not reflect these changes.
- In order to re-validate an invalid materialized view, you must grant the appropriate privileges to the owner of the materialized view and then drop and re-create the materialized view.

Object Privileges for 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 names and object owners. In addition, you can use a synonym to simplify SQL statements.

Synonyms provide independence by permitting applications to function without modification regardless of which object a synonym refers to. Synonyms can be used in DML statements, some DDL statements and cache statements.

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

Table 2-1 Privileges for Synonyms

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

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

ALL Object Privileges

You can grant all privileges for an object to a user with the `ALL` keyword. This grants a user the right to perform any operation on the object. The object owner and any user with the `ADMIN` privilege can execute the `GRANT ALL` and `REVOKE ALL` statements.

For example, `GRANT ALL ON hr.employees TO pat` grants all privileges for the `employees` table to user `pat`. It is possible to revoke individual privileges after granting all object privileges:

```
Command> GRANT ALL ON hr.employees TO pat;
Command> REVOKE DELETE ON hr.employees FROM pat;
```

You may also `REVOKE ALL` object privileges that were granted to a user for the object, as demonstrated here for user `pat`:

```
Command> REVOKE ALL ON hr.employees FROM pat;
```

Privileges for PL/SQL Objects

Authorization in PL/SQL requires certain privileges.

- Privileges for PL/SQL Statements and Operations
- Invalidated Objects
- Definer's Rights and Invoker's Rights (AUTHID Clause)

Note

Also, see Privileges for SQL Objects.

Privileges for PL/SQL Statements and Operations

There are required privileges for PL/SQL statements and operations.

- Overview of Privileges for PL/SQL Statements and Operations
- Privileges Reference for PL/SQL Statements and Operations
- Granting Privileges for PL/SQL Statements and Operations

Overview of Privileges for PL/SQL Statements and Operations

For PL/SQL users, authorization through the granting of privileges is necessary to enable a user to create, alter, drop, or execute PL/SQL procedures and functions, including packages and their member procedures and functions.

You need the `CREATE PROCEDURE` privilege to create a procedure, function, package definition, or package body if it is being created in your own schema, or `CREATE ANY PROCEDURE` if it is being created in any other schema. To alter or drop a procedure, function, package definition, or package body, you must be the owner or have the `ALTER ANY PROCEDURE` privilege or `DROP ANY PROCEDURE` privilege, respectively.

For a user to execute PL/SQL functions, PL/SQL procedures or PL/SQL packages that they do not own, they must be granted the `EXECUTE` object privilege for the procedure or function or for the package to which it belong, or granted EXECUTE ANY PROCEDURE. When you grant a user EXECUTE privilege on a package, this automatically grants EXECUTE privilege on its component procedures and functions.

EXECUTE privilege authorizes the following:

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

To explicitly compile using ALTER PROCEDURE or ALTER FUNCTION, the user must be granted the ALTER ANY PROCEDURE system privilege.

Privileges Reference for PL/SQL Statements and Operations

There are required privileges for PL/SQL statements and operations.

Required privileges for PL/SQL statements and operations are summarized in Table 2-2.

Table 2-2 Privileges for Using PL/SQL Procedures and Functions

Action	SQL Statement or Operation	Required Privilege

Create a procedure, function, package definition, or package body.	CREATE [OR REPLACE] PROCEDURE
	CREATE [OR REPLACE] FUNCTION
	CREATE [OR REPLACE] PACKAGE
	CREATE [OR REPLACE] PACKAGE BODY
		CREATE PROCEDURE in user's schema
		Or:
		CREATE ANY PROCEDURE in any other schema

Alter a procedure, function, or package.	ALTER PROCEDURE ALTER FUNCTION ALTER PACKAGE	Ownership of the procedure, function, or package Or: ALTER ANY PROCEDURE
Drop a procedure, function, package definition, or package body.	DROP PROCEDURE DROP FUNCTION DROP PACKAGE DROP PACKAGE BODY	Ownership of the procedure, function, or package Or: DROP ANY PROCEDURE
Execute a procedure or function.	Invoke the procedure or function.	Ownership of the procedure or function, or of the package to which it belongs (if applicable) Or: EXECUTE for the procedure or function, or for the package to which it belongs (if applicable) Or: EXECUTE ANY PROCEDURE

Granting Privileges for PL/SQL Statements and Operations

You can grant and then revoke EXECUTE privilege to user2 for a procedure and a package that user1 owns.

```
Command> grant execute on user1.myproc to user2;
Command> grant execute on user1.mypkg to user2;
...
Command> revoke execute on user1.myproc from user2;
Command> revoke execute on user1.mypkg from user2;
```

Note

- A user who has been granted privilege to execute a procedure (or function) can execute the procedure even without privileges on other procedures that the procedure calls. For example, consider a stored procedure `user2.proc1` that executes procedure `user2.proc2`. If `user1` is granted privilege to execute `proc1` but is not granted privilege to execute `proc2`, the user could not run `proc2` directly but could still run `proc1`.
- Privilege to execute a procedure or function allows implicit compilation of the procedure or function if it is invalid or not compiled at the time of execution.
- To invoke a procedure or function through a synonym, a user must have privilege to execute the underlying procedure or function.
- A SQL statement executed in PL/SQL requires the same privilege as when executed directly.
- `EXECUTE ANY PROCEDURE` does not apply to TimesTen supplied packages; however, most are accessible through the `PUBLIC` role.

The following example shows a series of attempted operations by a user, `user1`, as follows:

1. The user attempts each operation before having the necessary privilege. The resulting error is shown.
2. The instance administrator grants the necessary privilege.
3. The user successfully performs the operation.

The `ttIsql` utility is used by `user1` to perform (or attempt) the operations and by the instance administrator to grant privileges.

user1:

Initially the user does not have permission to create a procedure. That must be granted even in the user's own schema.

```
Command> create procedure testproc is
         begin
         dbms_output.put_line('user1.testproc called');
         end;
         /
15100: User USER1 lacks privilege CREATE PROCEDURE
The command failed.
```

Instance administrator:

```
Command> grant create procedure to user1;
```

user1:

Once `user1` can create a procedure in the `user1` schema, that user owns it and can execute it.

```
Command> create procedure testproc is
         begin
         dbms_output.put_line('user1.testproc called');
         end;
         /

Procedure created.

Command> begin
         testproc();
         end;
         /
user1.testproc called

PL/SQL procedure successfully completed.
```

The user cannot yet create a procedure in another schema, though.

```
Command> create procedure user2.testproc is
         begin
         dbms_output.put_line('user2.testproc called');
         end;
         /
15100: User USER1 lacks privilege CREATE ANY PROCEDURE
The command failed.
```

user1:

Now `user1` can create a procedure in another schema, but cannot execute it without owning it or having necessary privilege.

Instance administrator:

```
Command> grant create any procedure to user1;

Command> create procedure user2.testproc is
         begin
         dbms_output.put_line('user2.testproc called');
         end;
         /

Procedure created.

Command> begin
         user2.testproc();
         end;
         /
 8503: ORA-06550: line 2, column 7:
PLS-00904: insufficient privilege to access object USER2.TESTPROC
 8503: ORA-06550: line 2, column 1:
PL/SQL: Statement ignored
The command failed.
```

Instance administrator:

```
Command> grant execute any procedure to user1;
```

user1:

Now `user1` can execute a procedure in another schema.

```
Command> begin
         user2.testproc();
         end;
         /
user2.testproc called

PL/SQL procedure successfully completed.
```

Invalidated Objects

When a privilege on an object is revoked from a user, all of that user's PL/SQL objects that refer to that object are temporarily invalidated. Once the privilege has been restored, a user can explicitly recompile and revalidate an object by executing `ALTER PROCEDURE`, `ALTER FUNCTION`, or `ALTER PACKAGE`, as applicable, on the object. Alternatively, each object is recompiled and revalidated automatically the next time it is executed.
For example, if `user1` has a procedure `user1.proc0` that calls `user2.proc1`, `proc0` becomes invalid if `EXECUTE` privilege for `proc1` is revoked from `user1`.

Use the following to see if any of your objects are invalid:

```
select * from user_objects where status='INVALID';
```

This example shows a series of actions resulting in an invalidated PL/SQL procedure:

1. A user is granted `CREATE ANY PROCEDURE` privilege, creates a procedure in another user's schema, then creates a procedure in their own schema that calls the procedure in the other user's schema.
2. The user is granted `EXECUTE` privilege to execute the procedure in the other user's schema.
3. The user executes the procedure in their schema that calls the procedure in the other user's schema.
4. `EXECUTE` privilege for the procedure in the other user's schema is revoked from the user, invalidating the user's own procedure.
5. `EXECUTE` privilege for the procedure in the other user's schema is granted to the user again. When the user executes their own procedure, it is implicitly recompiled and revalidated.

Administrative user:

```
Command> grant create any procedure to user1;
```

user1:

```
Command> create procedure user2.proc1 is
         begin
         dbms_output.put_line('user2.proc1 is called');
         end;
         /

Procedure created.

Command> create procedure user1.proc0 is
         begin
         dbms_output.put_line('user1.proc0 is called');
         user2.proc1;
         end;
         /

Procedure created.
```

Administrative user:

```
Command> grant execute on user2.proc1 to user1;
```

user1:

```
Command> begin
         user1.proc0;
         end;
         /
user1.proc0 is called
user2.proc1 is called

PL/SQL procedure successfully completed.
```

And to confirm `user1` has no invalid objects:

```
Command> select * from user_objects where status='INVALID';
0 rows found.
```

Administrative user:

Now revoke the `EXECUTE` privilege from `user1`.

```
Command> revoke execute on user2.proc1 from user1;
```

user1:

Immediately, `user1.proc0` becomes invalid because `user1` no longer has privilege to execute `user2.proc1`.

```
Command> select * from user_objects where status='INVALID';
< PROC0, <NULL>, 273, <NULL>, PROCEDURE, 2021-06-04 14:51:34, 2021-06-04 14:58:23,
2021-06-04:14:58:23, INVALID, N, N, N, 1, <NULL> >
1 row found.
```

So `user1` can no longer execute the procedure.

```
Command> begin
         user1.proc0;
         end;
         /
 8503: ORA-06550: line 2, column 7:
PLS-00905: object USER1.PROC0 is invalid
 8503: ORA-06550: line 2, column 1:
PL/SQL: Statement ignored
The command failed.
```

Administrative user:

Again grant `EXECUTE` privilege on `user2.proc1` to `user1`.

```
Command> grant execute on user2.proc1 to user1;
```

user1:

The procedure `user1.proc0` is still invalid until it is either explicitly or implicitly recompiled. It is implicitly recompiled when it is executed, as shown here. Or `ALTER PROCEDURE` could be used to explicitly recompile it.

```
Command> select * from user_objects where status='INVALID';
< PROC0, <NULL>, 273, <NULL>, PROCEDURE, 2021-06-04 14:51:34, 2021-06-04 16:13:00,
2021-06-04:16:13:00, INVALID, N, N, N, 1, <NULL> >
1 row found.
Command> begin
         user1.proc0;
         end;
         /
user1.proc0 is called
user2.proc1 is called

PL/SQL procedure successfully completed.

Command> select * from user_objects where status='INVALID';
0 rows found.
```

Definer's Rights and Invoker's Rights (AUTHID Clause)

When a PL/SQL procedure or function is defined, the optional `AUTHID` clause of the `CREATE FUNCTION` or `CREATE PROCEDURE` statement specifies whether the function or procedure executes with *definer's rights* (`AUTHID DEFINER`, the default) or *invoker's rights* (`AUTHID CURRENT_USER`).

The `AUTHID` setting affects the name resolution and privilege checking of SQL statements that a procedure or function issues at runtime. With definer's rights, SQL name resolution and privilege checking operate as though the owner of the procedure or function (the definer, in whose schema it resides) is running it. With invoker's rights, SQL name resolution and privilege checking simply operate as though the current user (the invoker) is running it.

For procedures or functions in a package, the `AUTHID` clause of the `CREATE PACKAGE` statement specifies whether each member function or procedure of the package executes with definer's rights or invoker's rights. The `AUTHID` clause is shown in the syntax documentation for these statements, under SQL Statements in *Oracle TimesTen In-Memory Database SQL Reference*.

Invoker's rights would be useful in a scenario where you might want to grant broad privileges for a body of code, but would want that code to affect only each user's own objects in the user's own schema.

Definer's rights would be useful in a situation where you want all users to have access to the same centralized tables or other SQL objects, but only for the specific and limited actions that are executed by the procedure. The users would not have access to the SQL objects otherwise.

Refer to Invoker's Rights and Definer's Rights (AUTHID Property) in *Oracle Database PL/SQL Language Reference*.

The following example runs a script twice in `ttIsql` with just one change, first defining a PL/SQL procedure with `AUTHID CURRENT_USER` for invoker's rights, then with `AUTHID DEFINER` for definer's rights.

Script for AUTHID examples:

The script assumes three users have been created: a tool vendor and two tool users (`brandX` and `brandY`). Each has been granted `CREATE SESSION`, `CREATE PROCEDURE`, and `CREATE TABLE` privileges as necessary. The following setup is also assumed, to allow "`use` *`username`*`;`" syntax to connect to the database as *`username`*.

```
connect adding "uid=toolVendor;pwd=pw" as toolVendor;
connect adding "uid=brandX;pwd=pw" as brandX;
connect adding "uid=brandY;pwd=pw" as brandY;
```

The script does the following:

- Creates the procedure, `printInventoryStatistics`, as the tool vendor.
- Creates a table with the same name, `myInventory`, in each of the three user schemas, populating it with unique data in each case.

- Runs the procedure as each of the tool users.

The different results between the two executions of the script show the difference between invoker's rights and definer's rights.

Following is the script for the invoker's rights execution.

```
use toolVendor;
create table myInventory (name varchar2(100), inventoryCount tt_integer);
insert into myInventory values('butter', 1);

create or replace procedure printInventoryStatistics authid current_user is
 inventoryCount pls_integer;
begin
 select count(*) into inventoryCount from myInventory;
 dbms_output.put_line('Total items in inventory: ' || inventoryCount);
 for currentItem in (select * from myInventory) loop
   dbms_output.put_line(currentItem.name || ' ' || currentItem.inventoryCount);
 end loop;
end;
/
grant execute on printInventoryStatistics to brandX;
grant execute on printInventoryStatistics to brandY;

use brandX;
create table myInventory (name varchar2(100), inventoryCount tt_integer);
insert into myInventory values('toothpaste', 100);
set serveroutput on
execute toolVendor.printInventoryStatistics;

use brandY;
create table myInventory (name varchar2(100), inventoryCount tt_integer);
insert into myInventory values('shampoo', 10);
set serveroutput on
execute toolVendor.printInventoryStatistics;
```

The only difference for the definer's rights script is the change in the `AUTHID` clause for the procedure definition.

```
...
create or replace procedure printInventoryStatistics authid definer is
 inventoryCount pls_integer;
begin
 select count(*) into inventoryCount from myInventory;
 dbms_output.put_line('Total items in inventory: ' || inventoryCount);
 for currentItem in (select * from myInventory) loop
   dbms_output.put_line(currentItem.name || ' ' || currentItem.inventoryCount);
 end loop;
end;
/
...
```

Using AUTHID CURRENT_USER:

This part shows the results when the procedure is defined with invoker's rights. Note that when the tool users `brandX` and `brandY` run the `printInventoryStatistics` procedure, each sees the data in the `myInventory` table as the invoker.

```
Command> run invoker.sql

use toolVendor;
create table myInventory (name varchar2(100), inventoryCount tt_integer);
insert into myInventory values('butter', 1);
```

```
1 row inserted.

create or replace procedure printInventoryStatistics authid current_user is
 inventoryCount pls_integer;
begin
 select count(*) into inventoryCount from myInventory;
 dbms_output.put_line('Total items in inventory: ' || inventoryCount);
 for currentItem in (select * from myInventory) loop
   dbms_output.put_line(currentItem.name || ' ' || currentItem.inventoryCount);
 end loop;
end;
/

Procedure created.

grant execute on printInventoryStatistics to brandX;
grant execute on printInventoryStatistics to brandY;

use brandX;
create table myInventory (name varchar2(100), inventoryCount tt_integer);
insert into myInventory values('toothpaste', 100);
1 row inserted.
set serveroutput on;

execute toolVendor.printInventoryStatistics;
Total items in inventory: 1
toothpaste 100

PL/SQL procedure successfully completed.

use brandY;
create table myInventory (name varchar2(100), inventoryCount tt_integer);
insert into myInventory values('shampoo', 10);
1 row inserted.
set serveroutput on;

execute toolVendor.printInventoryStatistics;
Total items in inventory: 1
shampoo 10

PL/SQL procedure successfully completed.
```

Use the following to terminate all the connections:

```
Command> disconnect all;
```

Using AUTHID DEFINER:

This part shows the results when the procedure is defined with definer's rights. Note that when the tool users `brandX` and `brandY` run `printInventoryStatistics`, each sees the data in `myInventory` belonging to the tool vendor (the definer).

```
Command> run definer.sql

use toolVendor;

create table myInventory (name varchar2(100), inventoryCount tt_integer);
insert into myInventory values('butter', 1);
1 row inserted.

create or replace procedure printInventoryStatistics authid definer is
 inventoryCount pls_integer;
```

```
begin
 select count(*) into inventoryCount from myInventory;
 dbms_output.put_line('Total items in inventory: ' || inventoryCount);
 for currentItem in (select * from myInventory) loop
   dbms_output.put_line(currentItem.name || ' ' || currentItem.inventoryCount);
 end loop;
end;
/

Procedure created.

grant execute on printInventoryStatistics to brandX;
grant execute on printInventoryStatistics to brandY;

use brandX;
create table myInventory (name varchar2(100), inventoryCount tt_integer);
insert into myInventory values('toothpaste', 100);
1 row inserted.
set serveroutput on;

execute toolVendor.printInventoryStatistics;
Total items in inventory: 1
butter 1

PL/SQL procedure successfully completed.

use brandY;
create table myInventory (name varchar2(100), inventoryCount tt_integer);
insert into myInventory values('shampoo', 10);
1 row inserted.
set serveroutput on;

execute toolVendor.printInventoryStatistics;
Total items in inventory: 1
butter 1

PL/SQL procedure successfully completed.
```

In this case, it is also instructive to see that although `brandX` and `brandY` can each access the `toolVendor.myInventory` table through the procedure, they cannot access it directly. That is a key use of definer's rights, to enable specific and restricted access to a table or other SQL object through the actions of a procedure.

```
Command> use brandX;
brandx: Command> select * from toolVendor.myInventory;
15100: User BRANDX lacks privilege SELECT on TOOLVENDOR.MYINVENTORY
The command failed.

brandx: Command> use brandY;
brandy: Command> select * from toolVendor.myInventory;
15100: User BRANDY lacks privilege SELECT on TOOLVENDOR.MYINVENTORY
The command failed.
```

When finished, terminate all the connections:

```
Command> disconnect all;
```

Privileges for Cache Groups

Cache groups require certain users and privileges.

- About Cache Group Users and Privileges
- Oracle Cache Administration User Privilege
- TimesTen Cache Administration User Privilege
- Privileges for Other Cache Users

About Cache Group Users and Privileges

There are system and object privileges for cache groups. In order for a user to perform operations involving any cache group, the user must have the appropriate cache group privileges.

In addition, there are administrative users, namely the Oracle cache administration user and the TimesTen cache administration user, who require special privileges. See Cache Group Users.

For a complete list of system and object privileges for cache group operations, see Privileges in *Oracle TimesTen In-Memory Database SQL Reference*.

> **Note**
>
> Passthrough does not require any cache group privileges, because privileges are checked by the Oracle database with the user credentials.

Oracle Cache Administration User Privilege

To grant the Oracle cache administration user the minimum set of privileges required to perform cache operations: On Oracle Database, run the SQL*Plus script `grantCacheAdminPrivileges.sql` in the `*timesten_home*/install/oraclescripts` directory as the `SYS` user.

See Grant Privileges to the Oracle Cache Administration User in *Oracle TimesTen In-Memory Database Cache Guide*.

TimesTen Cache Administration User Privilege

The required privilege for the TimesTen cache administration user is the `CACHE_MANAGER` system privilege, enabling the user to perform necessary cache group operations.

A TimesTen cache administration user must have the `CACHE_MANAGER` privilege to perform the initial load of a read-only cache group or to change the state of autorefresh on a read-only cache group. (The initial load implicitly alters the state of the cache group autorefresh from paused to on.)

For a complete list of individual cache group operation privileges, see Required Privileges for Cache Administration User for Cache Operations in *Oracle TimesTen In-Memory Database Cache Guide*.

This grants the `CACHE_MANAGER` privilege to `pat`:

```
Command> GRANT CACHE_MANAGER TO pat;
```

Note

An asynchronous writethrough (AWT) cache group combines both cache groups and replication. The `CACHE_MANAGER` privilege provides all of the privileges needed for creating AWT cache groups.

Privileges for Other Cache Users

Certain cache group privileges are requires for other users (non-administrative users).

- Non-Administrative Cache Users
- Cache Group System Privileges
- Cache Group Object Privileges

Non-Administrative Cache Users

Operations on a cache group or a cache table, such as loading a cache group or updating a cache table, can be performed by any TimesTen user who has sufficient privileges.

Note that for these users, there must also be a corresponding Oracle Database user with the same name who has privilege to select from and update the cached Oracle Database tables.

See Create the TimesTen Users and Grant Privileges to the TimesTen Users in *Oracle TimesTen In-Memory Database Cache Guide*.

Cache Group System Privileges

Cache group system privileges enable a user to operate on cache group objects across the database.

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

For example, the following confers the privilege for a user to alter any cache group in the database:

```
Command> GRANT ALTER ANY CACHE GROUP TO pat;
```

These cache group system privileges are for operations on objects not owned by the user:

- `FLUSH ANY CACHE GROUP`
- `LOAD ANY CACHE GROUP`
- `UNLOAD ANY CACHE GROUP`
- `REFRESH ANY CACHE GROUP`

Cache Group Object Privileges

Object privileges for cache group operations enable a user to perform a particular operation on a particular cache group that the user does not own.

These are the available cache group object privileges:

- `FLUSH`
- `LOAD`
- `UNLOAD`
- `REFRESH`

This example grants `pat` the cache group object privilege to perform a `FLUSH` on the cache group `cachegrp` that is owned by `terry`:

```
Command> GRANT FLUSH ON terry.cachegrp TO pat;
```

See Methods for Transmitting Changes Between TimesTen and Oracle Databases in *Oracle TimesTen In-Memory Database Cache Guide*.

User Privilege Views

You can view the privileges granted to each user through certain views.

Table 2-3 System Views for User Privileges

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

This example displays the system privileges granted to all users:

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

Note

See System Tables and Views in *Oracle TimesTen In-Memory Database System Tables and Views Reference*.

3

Encryption at Rest in TimesTen

TimesTen supports encryption at rest (encryption of data stored in the file system) for database files, such as checkpoint and transaction log files.
These topics are discussed here:

- Configuring Encryption at Rest
- Managing Encryption Keys
- Replicating an Encrypted Database
- Backing Up and Restoring an Encrypted Database

Note

From this point forward, a database configured for encryption at rest of database files is referred as an encrypted database.

Configuring Encryption at Rest

You can optionally enable encryption at rest and specify the encryption algorithm used to encrypt the data stored in the checkpoint and transaction log files.

Note

Encryption at rest has a measurable performance impact to XLA and other database operations. See XLA and TimesTen Event Management in *Oracle TimesTen In-Memory Database C Developer's Guide*.

TimesTen supports these encryption algorithms:

- AES-128
- AES-192
- AES-256

TimesTen provides these data store attributes to set up encryption at rest upon database creation:

- `DbEncrypted`: Enables encryption at rest and specifies the encryption algorithm to be used. Valid values are:
  - `none` (default)
  - `AES128`
  - `AES192`
  - `AES256`

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

- `DbKeyWallet`: Specifies the path for the Oracle wallet used to store the master key for encryption at rest. Default value is `*instance_home*/info/wallets`. The Oracle wallet is an auto-login wallet. See DbKeyWallet in *Oracle TimesTen In-Memory Database Reference*.

> **Warning**
>
> If the wallet with the master key for encryption at rest is lost or deleted, the database becomes unreadable and cannot be recovered.

Encryption at rest in TimesTen supports FIPS 140-2 mode. While it is not a requirement to encrypt a database, FIPS mode consists of a set of tests performed on the cryptographic engine to validate that it has not been tampered with or modified. You can enable FIPS mode during the creation of the TimesTen instance or by restarting the instance after enabling FIPS mode (`db_fips_140=1`) in the TimesTen instance configuration file (`timesten.conf`). See TimesTen Instance Configuration File in *Oracle TimesTen In-Memory Database Reference*.

You can switch databases files from non-encrypted to encrypted on an existing database by using a rekey operation (see Re-Keying an Encrypted Database). However, for security, you cannot switch encrypted database files to non-encrypted.

Example 3-1 DSN Specification for an Encrypted Database

This is an example of a DSN specification for a database, `database1`, configured for encryption at rest. Encryption at rest is enabled and set to use the AES-256 encryption algorithm upon database creation. The wallet for the master key is set to be stored within the `/disk1/wallets` directory.

```
[database1]
DataStore=/disk1/databases/database1
LogDir=/disk1/logs
DatabaseCharacterSet=AL32UTF8
DbEncrypted=AES256
DbKeyWallet=/disk1/wallets
```

Managing Encryption Keys

TimesTen creates two types of keys for encryption at rest: a master key and a data encryption key (DEK) specific to each checkpoint or transaction log file.
During the creation of an encrypted database, TimesTen creates a randomly-generated master key and stores it in a wallet. Every time TimesTen creates a new checkpoint or transaction log file for the database, it also creates a randomly-generated DEK. The DEK is exclusively for the encryption and decryption of that specific checkpoint or transaction log file. TimesTen uses the master key to encrypt and decrypt DEKs.

- Checkpoint Files
- Transaction Log Files
- Re-Keying an Encrypted Database

Checkpoint Files

During a checkpoint operation, TimesTen encrypts all blocks of a checkpoint file using its corresponding DEK. Then, TimesTen encrypts the DEK with the master key and stores the encrypted DEK and the index to identify the master key in an unencrypted section of the checkpoint file.
A checkpoint file is basically an image of the data stored in memory. To ensure checkpoint files remain as such, for encrypted databases, a new encryption data file (edata file) is created for each checkpoint file. The edata file contains the encryption metadata for each block in the checkpoint file (TimesTen uses 12 byte initialization vectors to encrypt each block in a checkpoint file).

The edata files are named `*dsname*.edata0` or `*dsname*.edata1`, where `*dsname*` is the database path name and file name prefix specified in the `DataStore` attribute, and the `0` or `1` suffix corresponds to the checkpoint file the edata file complements.

Edata files have the same permissions as checkpoint files—read and write privileges for owner and group (660)—and are approximately 50 bytes per block in size.

Transaction Log Files

For encrypted databases, TimesTen encrypts all blocks of a transaction log file using its corresponding DEK. Then, TimesTen encrypts the DEK with the master key and stores the encrypted DEK and the index to identify the master key in an unencrypted section of the log file.
TimesTen uses 12 byte initialization vectors to encrypt each block in a log file. TimesTen stores the integrity data for each block at the end of the block.

Re-Keying an Encrypted Database

Re-keying an encrypted database consists of rotating the master key and DEKs used to encrypt the checkpoint and transaction log files. The database files are then re-encrypted using the new keys. Master keys are associated with an index number—the index consists of a monotonically increasing sequence of numbers. The wallet generally only stores one master key. It may store up to two master keys during an ongoing rekey operation: the new and previous master keys. Therefore, if we assume—prior to a re-keying request—that the last master key is associated with an index value of `3` (meaning that the encryption keys have already been rotated twice), then during a new rekey operation, TimesTen does the following:

1. Creates a new randomly-generated master key and assigns the next number in the sequence, `4` in this case, as its index. Then, it stores the new master key in the wallet.
2. Creates two new checkpoint files and their corresponding randomly-generated DEKs.
3. Encrypts the checkpoint files using their corresponding DEKs.
4. Encrypts the DEKs using the new master key and stores the encrypted DEKs and master key index, `4`, in the header of the checkpoint files.
5. Searches for any transaction log files using the previous master key, which in this case is the master key with index `3`, and creates a new randomly-generated DEK for each of these log files.
6. Re-encrytps the log files using their corresponding new DEKs.
7. Encrypts the DEKs using the new master key and stores the encrypted DEKs and master key index, `4`, in the header of the corresponding log files.

8. Removes the previous master key from the wallet. In this case, this leaves the wallet with only the master key with index `4`, since the master key with index `3` is removed.

You can also use a rekey operation to change the encryption details, such as changing the encryption algorithm or encrypting a non-encrypted database.

Tip

- It is highly recommended that you implement a time-based key rotation strategy as best practice based on your own risk assessment, contractual requirements, and security policies of your organization.

  Use the `ttRekeyStatus` built-in utility to determine when the database was last rekeyed.

  ```
  Command> CALL ttRekeyStatus();
  < Rekey done, 2025-09-04 18:51:13.072758, 2025-09-04
  18:51:14.786877, 3 >
  1 row found.
  ```

- Since there is a performance cost to a rekey operation, it is recommended that any rekey operation is performed during low-demand periods and that a checkpoint is performed first.

Note

You cannot rekey an encrypted database while any of these operations is in progress:

- Rekey operation
- Database backup
- Database duplicate (for replication)

You can perform a rekey operation using the `-rekey` option of the `ttAdmin` utility.

```
ttAdmin -rekey dsname
```

If you want to change the encryption algorithm or want to encrypt an unencrypted database, you must include the `-encrAlg` option and specify encryption algorithm TimesTen should use.

```
ttAdmin -rekey -encrAlg AES256 dsname
```

Note

A rekey operation can only be performed by the instance administrator and the database must be loaded into memory.

You can use the `ttRekeyStatus` built-in procedure to determine the status of the rekey operation.

```
Command> CALL ttRekeyStatus();
< Rekey done, 2025-09-11 22:19:39.044642, 2025-09-11 22:19:39.997542, 4 >
1 row found.
```

For more information on the `-rekey` option of the `ttAdmin` utility or the `ttRekeyStatus` built-in procedure, see Encryption at Rest or ttRekeyStatus, respectively, in *Oracle TimesTen In-Memory Database Reference*.

Replicating an Encrypted Database

TimesTen supports replication for encrypted databases. During a database duplicate operation (`ttRepAdmin -duplicate`), TimesTen reads the required keys to decrypt the checkpoint files and copies the wallet to the path specified by the `DbKeyWallet` attribute of the database targeted as a duplicate.
To replicate an encrypted database, it is mandatory to have consistent TLS configuration between TimesTen instances (replication peers), that is, the cipher suites and replication wallet settings between replication peers must match and be explicitly required to match. See Task 3: Configure TLS for Replication.

Warning

Encryption attempts for an unencrypted database fail if the replication agent is already running. Ensure that you stop the replication agent before attempting to encrypt an unencrypted database.

TimesTen records rekey requests in the transaction log file. The replication agent replicates the rekey operation on standby or subscriber databases once it reads the transaction log file.

- For active standby pair replication, you can only issue a rekey request on the active database.
- For classic replication, you can only issue a rekey request on a master database. However, unlike with a single active database, it is possible for two separate masters to issue concurrent rekey requests. This may happen if there is some kind of delay due to network partition or one of the replication agents being down. In this case, to avoid conflicts with concurrent rekey requests that change the encryption algorithm, master database A only needs to replicate a rekey request from master database B if the request from B has a newer time stamp than the last request from A and vice versa.

For an example on how to set up replication for an encrypted database, see Configuring an Active Standby Pair with One Subscriber or Configuring a Classic Replication Scheme with One Master and One Subscriber in *Oracle TimesTen In-Memory Database Replication Guide* while taking into account Example 3-2 and Example 3-3 for step 1 and step 2, respectively.

Example 3-2 DSN Specification for an Active Standby Pair Replication Scheme for Encrypted Databases

This is an example of a DSN specification for a set of databases in an active standby pair with one subscriber configuration. Encryption at rest is enabled and set to use the AES-256 encryption algorithm upon database creation. The wallet for the master key is set to be stored within the `/disk1/wallets` directory.

```
[master1]
DataStore=/disk1/databases/master1
DatabaseCharacterSet=AL32UTF8
ConnectionCharacterSet=AL32UTF8
DbEncrypted=AES256
DbKeyWallet=/disk1/wallets

[master2]
DataStore=/disk1/databases/master2
DatabaseCharacterSet=AL32UTF8
ConnectionCharacterSet=AL32UTF8
DbEncrypted=AES256
DbKeyWallet=/disk1/wallets

[subscriber1]
DataStore=/disk1/databases/subscriber1
DatabaseCharacterSet=AL32UTF8
ConnectionCharacterSet=AL32UTF8
DbEncrypted=AES256
DbKeyWallet=/disk1/wallets
```

Example 3-3 Required Parameters in the TimesTen Instance Configuration File for Replication of Encrypted Databases

This is an example of the required parameters in the TimesTen instance configuration file (`timesten.conf`) for replication of encrypted databases.

```
replication_wallet=/disk1/wallets/serverWallet
replication_cipher_suite=TLS_AES_256_GCM_SHA384
replication_ssl_mandatory=1
```

Where:

- `replication_wallet` is the absolute path to the server wallet.
- `replication_cipher_suite` is the cipher suite to be used for encrypted communication between replication agents.
- `replication_ssl_mandatory` specifies whether it is mandatory to have consistent TLS configuration between replication peers. `1` indicates that replication cannot proceed unless TLS settings match on all TimesTen instances.

Backing Up and Restoring an Encrypted Database

TimesTen supports back up and restore operations for encrypted databases.
A backup file for an encrypted database includes the wallet with the master key required to decrypt the data from the checkpoint and transaction log files stored in the backup file. This is to avoid issues of a mismatching master key in a restore operation due to any rekey operations performed after the backup.

The master key and data encryption keys (DEKs) stored in a database backup are from the time the backup was created, and they may have already expired—based on the security policies established by your organization—once the backup is used for a restore operation.

Given this, as best practice, it is highly recommended that you perform a rekey operation after restoring an encrypted database. See Re-Keying an Encrypted Database

For more information on how to either back up or restore a database, see Backing Up and Restoring a Database in *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.

4 Secure Network Communication in TimesTen

Some features in TimesTen, such as client/server and replication, use Transport Layer Security (TLS) to secure network communication between TimesTen instances. TLS is a secure and standard protocol that encrypts connections. It ensures communication is authenticated, encrypted, and protected from tampering.

These topics are discussed here:

Note

TLS configuration in TimesTen does not apply to communication between TimesTen and Oracle Database, such as when cache is used. Secure TimesTen-Oracle communications can be configured through settings in the `sqlnet.ora` file used for connections to Oracle Database. See Configuring Transport Layer Security Authentication in *Oracle Database Security Guide*.

Overview of TLS in TimesTen

TLS is an industry-standard protocol used to secure network connections. In TimesTen, TLS helps ensure that:

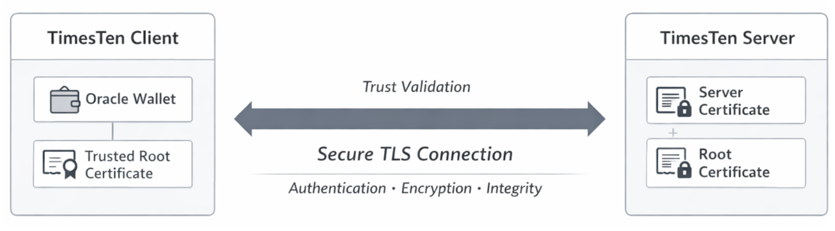
- Clients and servers can verify each other’s identities
- Data exchanged over the network is encrypted
- Tampering or modifications to data in transit are detected

TLS uses **digital certificates** that are based on public key infrastructure (PKI) to support trusted authentication between systems. Certificates and the cryptographic keys they contain are stored in **Oracle Wallets**, which are secure containers that TimesTen components use when initiating or accepting TLS connections.

When a TimesTen client connects to a TimesTen server (or when replication agents communicate), the following occurs:

1. Access Oracle Wallets. Each TimesTen component (client, server, or replication agent) reads its Oracle Wallet, which contains TLS certificates.
2. Verify identity. The server presents its certificate to the client. The client uses a trusted root certificate to verify that the server’s certificate is valid and that the server is authentic. If there is a requirement, the server can verify the client authenticity as well with the `SSLClientAuthentication=1` connection attribute setting.

3. Perform a TLS handshake and establish session. TLS negotiates cryptographic parameters and establishes a secure session.
4. Encrypt communication. Once the session is established, all data exchanged over the network is encrypted and integrity protected.



Cipher Suites in TimesTen

A cipher suite is a set of authentication, encryption, and data integrity algorithms used for exchanging messages between network entities.

Each cipher suite defines the algorithms used for:

- **Authentication** (digital signatures)
- **Encryption** (data confidentiality)
- **Data integrity** (message authentication)

During the TLS handshake, the client and server negotiate a cipher suite from the list of cipher suites that both sides support. The selected cipher suite is then used for all communication during the TLS session.

These topics are discussed here:

- Certificate Key Type and Cipher Suite Compatibility
- Supported TLS Protocol Versions
- Cipher Suite Configuration and Negotiation
- TLS 1.3 and 1.2 Cipher Suites Supported by TimesTen

Certificate Key Type and Cipher Suite Compatibility

A TLS cipher suite specifies, among other components, the authentication (digital signature) algorithm used during the TLS handshake. Common authentication algorithms include RSA and ECDSA.

Certificates stored in Oracle Wallets and used for TLS client or server authentication contain a public key based on either RSA or ECDSA. The authentication algorithm defined in the selected cipher suite must be compatible with the certificate's key algorithm.

If the certificate key type and the cipher suite authentication algorithm do not match, the TLS handshake fails.

RSA Support

RSA (Rivest–Shamir–Adleman) is a public-key cryptographic algorithm based on the mathematical difficulty of factoring large prime numbers. TimesTen supports:

- Key sizes: 2048 and 4096 bits (default)

- Signing Algorithms: sha256, sha384, sha512 (default)

Certificates that contain an RSA key require cipher suites that use RSA authentication (for example, TLS_ECDHE_RSA_*).

ECDSA Support

ECDSA (Elliptic Curve Digital Signature Algorithm) is a public-key asymmetric cryptographic algorithm that uses elliptic curve mathematics to generate strong digital signatures with smaller key sizes than RSA. TimesTen supports:

- Elliptic Curves: p256, p384, p521 (default)
- Signing Algorithms: ecdsasha256, ecdsasha384, ecdsasha512 (default)

Certificates that contain an ECC (ECDSA) key require cipher suites that use ECDSA authentication (for example, TLS_ECDHE_ECDSA_*).

To determine whether a certificate uses RSA or ECC, if you have OpenSSL installed, navigate to your server certificate and run the following command:

```
openssl x509 -in server1.cert -text -noout
```

where `server1.cert` is the name of the server certificate.

Look for output similar to the following:

- For RSA: `Public Key Algorithm: rsaEncryption`
- For ECC: `Public Key Algorithm: id-ecPublicKey`

In summary, when configuring TLS:

1. Determine whether your certificate uses RSA or ECC.
2. Configure at least one cipher suite that matches the certificate's key type.
3. Ensure that the selected signing algorithm and key size/curve are supported by TimesTen.

Using compatible certificate key types and cipher suites ensures successful TLS negotiation and strong cryptographic protection.

Supported TLS Protocol Versions

TimesTen supports the following TLS protocol versions:

- TLS 1.3 provides improved security, performance, and simpler cipher suite negotiation.
- TLS 1.2 supports compatibility with older TimesTen clients.

Cipher Suite Configuration and Negotiation

Cipher Suite Configuration

Since there is no default cipher suite setting, you must specify the `CipherSuites` connection attribute in both your client and server configuration. TimesTen provides the `CipherSuites` connection attribute to specify one or more cipher suites. (The exception is if you specified the cipher suite setting at the instance level. In this case, the setting is in the `timesten.conf` file.)

You can specify one or more cipher suites for the TimesTen client-specific `CipherSuites` connection attribute and for the TimesTen server-specific `CipherSuites` connection attribute. If you specify more than one cipher suite for the connection attribute, ensure to separate each by a comma. TimesTen recommends that you list the cipher suites from strongest to weakest. Ensure that the client and the server have at least one common cipher suite. If the client and server cannot agree on a common cipher suite, the TLS connection fails.

When configuring client/server connections between two release-environments, configure cipher suites recognized by both releases:

- For client/server connections between TimesTen Releases 26.1 and 22.1, specify at least one of the following cipher suites on both the client and server sides:
  - `TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`
  - `TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384`
- For client/server connections between TimesTen Releases 26.1 and 18.1, specify at least one of the following cipher suites on both the client and server sides:
  - `SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`
  - `SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384`

Cipher Suite Negotiation

- Protocol Version Selection
  - If both client and server list at least one TLS 1.3 cipher suite, TLS 1.3 is negotiated and TLS 1.2 cipher suites are ignored.
  - If no mutually configured TLS 1.3 cipher suite exists, TimesTen falls back to TLS 1.2 cipher suites.
- Cipher Suite Selection
  - TLS 1.3: When TLS 1.3 is negotiated, the client and server select the first TLS 1.3 cipher suite that appears in both `CipherSuites` lists. The order of cipher suites in the client's `CipherSuites` list determines the selection.
  - TLS 1.2: When TLS 1.2 is negotiated, the client proposes its ordered list of TLS 1.2 cipher suites. The server selects the first cipher suite from the client's list that is also configured on the server.

CipherSuites Length Limitation

- The `CipherSuites` connection attribute is limited to 256 characters. Any cipher suites listed beyond this limit are silently truncated and ignored.
- When configuring mixed TLS 1.3 and TLS 1.2 cipher suites, ensure that all required cipher suites appear within the first 256 characters to avoid negotiation failures.

TLS 1.3 and 1.2 Cipher Suites Supported by TimesTen

The following table lists the supported cipher suites (including the SSL-named cipher suites) and the authentication, encryption, and data integrity algorithm each uses. The last two columns indicate if the cipher suite is valid for the ECC or the RSA public key algorithm. Ensure that you choose a cipher suite that is valid for the key (ECC or RSA) that was used to sign the certificates.

Table 4-1 TLS 1.3 Cipher Suites

Cipher Suite	Authentication	Encryption	Data Integrity	Valid for ECC?	Valid for RSA?
`TLS_AES_128_CCM_SHA256`	CDHE_RSA, DHE_RSA, ECDHE_ECDSA	AES 128 CCM	SHA256 (SHA-2)	Yes	Yes
`TLS_AES_128_GCM_SHA256`	CDHE_RSA, DHE_RSA, ECDHE_ECDSA	AES 128 GCM	SHA256 (SHA-2)	Yes	Yes
`TLS_AES_256_GCM_SHA384`	CDHE_RSA, DHE_RSA, ECDHE_ECDSA	AES 256 GCM	SHA384 (SHA-2)	Yes	Yes
`TLS_CHACHA20_POLY1305_SHA256` (non-FIPS only)	CDHE_RSA, DHE_RSA, ECDHE_ECDSA	CHACHA20 POLY1305	SHA256 (SHA-2)	Yes	Yes

Note

`TLS_AES_256_GCM_SHA384` is the strongest cipher suite.

Table 4-2 TLS 1.2 Cipher Suites

Cipher Suite	Authentication	Encryption	Data Integrity	Valid for ECC?	Valid for RSA?
`TLS_DHE_RSA_WITH_AES_128_GCM_SHA256`	DHE_RSA	AES 128 GCM	SHA256 (SHA-2)	No	Yes
`TLS_DHE_RSA_WITH_AES_256_GCM_SHA384`	DHE_RSA	AES 256 GCM	SHA384 (SHA-2)	No	Yes
`TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`	ECDHE_ECDSA	AES 128 GCM	SHA256 (SHA-2)	Yes	No
`TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384`	ECDHE_ECDSA	AES 256 GCM	SHA384 (SHA-2)	Yes	No
`TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`	ECDHE_RSA	AES 128 GCM	SHA256 (SHA-2)	No	Yes
`TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`	ECDHE_RSA	AES 256 GCM	SHA384 (SHA-2)	No	Yes

Note

You can use the `SSL` prefix rather than `TLS` for the supported cipher suites for cross release client/server configurations or backward compatibility. For example, you can specify either `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384` or `SSL_ECDHE_RSA_WITH_AES_256_GCM_SHA384`.

About Certificates and Oracle Wallets

In TimesTen, digital certificates are used to establish trust between systems during a TLS connection. These certificates follow the X.509 v3 standard and bind an identity to a public key.

An Oracle Wallet stores certificates and private keys required for TLS communication. Each TimesTen component that participates in TLS must have access to a suitable wallet. For example:

- The **server wallet** contains both the server certificate and a trusted root certificate.
- The **client wallet** contains the trusted root certificate needed to validate the server certificate.

TimesTen provides the `ttCreateCerts` utility to create wallets and certificates for use with TLS. You can use this utility to generate self-signed certificates or prepare for certificates issued by a certificate authority.

Transport Layer Security for TimesTen Client/Server

When using a client/server connection, you can optionally configure and use Transport Layer Security (TLS) for encrypted communication between clients and the server.

This section discusses TimesTen support for TLS for client/server, covering these topics:

- About Using Certificates with Client/Server
- Configuration for TLS for Client/Server
- Using TLS for Client/Server in TimesTen

About Using Certificates with Client/Server

For self-signed certificates, TimesTen provides the `ttCreateCerts` utility for the generation of certificates for TLS.

TimesTen uses Oracle Wallets to store certificates, which you can use separately after the instance has been created. For general information about these wallets, also referred to as "keystores", refer to How the Keystore for the Storage of TDE Master Encryption Keys Works in *Oracle Database Advanced Security Guide*.

Note

The server validates the client identity using client certificate that is stored in the client wallet. This is a way for the server to verify that the connecting client is a legitimate client, but the user still must provide user ID and password credentials.

In TimesTen:

- TimesTen can generate certificates and place them on an instance when it is created.
- The TimesTen server has its own self-signed root certificate.
- The user typically imports, or optionally copies, the client wallet to each client.

Regarding certificates generated by TimesTen:

- Certificates produced are self-signed and stored in an Oracle Wallet.
- The root certificate has a default expiration time. It is your responsibility to track this. When the root certificate expires, regenerate all certificates. When the TimesTen regenerates the root certificate, all clients must re-import it.
- Clients store the root certificate in a local wallet, that is accessible to the client.
- Wallets produced are auto-login or single-sign-on (SSO) wallets, without a password. Access to wallets is controlled by file system permissions.
- The wallets are platform-independent.
- Because the certificates are self-signed, they cannot be revoked. But certificates can be regenerated as needed.

TimesTen also supports the use of certificates signed by a third-party CA. The wallet of each server has its own public/private key identity signed by the root certificate. Each client that connects to a server must have a wallet containing the root certificate of that server. (A client may optionally have multiple wallets for connections to multiple database servers.)

See Task 2A: Create Self-Signed Certificates and Task 2B: Create CA-Signed Certificates for configuration details.

Configuration for TLS for Client/Server

There is both server-side and the client-side configuration for TLS for client/server.

- Server Attributes for TLS
- Client Attributes for TLS

Server Attributes for TLS

These are the server connection attributes that determine settings for TLS for client/server.

For configuration details, see also Task 3: Configure the Server for TLS.

- `Wallet`: Specify the wallet location, as an absolute path, where certificates were placed. Assuming `ttCreateCerts` was used, this is the full path of the `serverWallet` directory.
- `Encryption` (encryption flag): Use one of the following settings. These descriptions assume matching cipher suite settings between the server and client, where applicable.
  - `accepted`: Enable an encrypted session if required or requested by the client; use an unencrypted session otherwise. This is the default.
  - `rejected`: Demand an unencrypted session. (If the server does not support encryption, TimesTen behaves as if this is the setting on the server.) The connection is rejected if the client requires encryption.
  - `requested`: Request an encrypted session if the client allows it (if the client has any setting other than `rejected`); use an unencrypted session otherwise.
  - `required`: Demand an encrypted session. Reject the connection if the client rejects encryption.

  See Table 4-3 later in this section for a summary of the results of each possible combination of settings of this attribute between the server and client, with consideration of the cipher suite settings.
- `CipherSuites`: This lists the cipher suite or suites that can be used, depending also on the client setting. Specify the desired cipher suites, comma-separated and in order of

preference. See Cipher Suites in TimesTen for the list of supported cipher suites for TLS 1.3 and 1.2. For TLS to be used, the server and client settings must include at least one common cipher suite.

- `SSLClientAuthentication`: Specifies whether TLS client authentication is required (setting of 1) or not (setting of 0, the default). With client authentication, the server validates an identity presented by the client, and requires an identity (public/private key) in the client wallet. Note that regardless of the client authentication setting, server authentication is performed, where the client validates the server.

  The server and client must have the same `SSLClientAuthentication` setting.

Note

As an alternative to the preceding server connection attributes, these equivalent attributes are available in the instance-level TimesTen configuration file, *`timesten_home`*`/conf/timesten.conf`, on the server. Connection attribute settings take precedence.

- `server_wallet`
- `server_encryption`
- `server_cipher_suites`
- `ssl_client_authentication`

If you have more than one database in a TimesTen instance, these settings apply to all, but can be overridden for each database through the server DSN definition.

TimesTen supports TLS session renegotiation, where new session keys are generated during an active TLS session for more robust security. Session renegotiations are performed according to either how much data has been transferred or how much time has passed. If you want to enable this feature, use one these attributes in the server DSN definition:

- `SSLRenegotiationSize`: Specifies a number of megabytes of data transfer in either direction between the client and server, after which session renegotiation is performed. The default setting is 0, meaning do not renegotiate based on megabytes transferred.
- `SSLRenegotiationPeriod`: Specifies a period of time, in minutes, after which session renegotiation is performed. The default setting is 0, meaning do not renegotiate based on a time period.

If both attributes are set to nonzero values, whichever situation occurs first will result in renegotiation.

The following table shows the results of all possible combinations of encryption flag settings between client and server, with consideration of the cipher suite settings.

Table 4-3 Results of Combinations of Server and Client Encryption Settings

Server Encryption Flag Setting	Client Encryption Flag Setting	Result
`accepted`	`accepted`	Connection accepted, encryption off.
`accepted`	`rejected`	Connection accepted, encryption off.
`accepted`	`requested`	Connection accepted. Encryption on if there is overlap between the cipher suite settings, off if there is not.

Table 4-3 (Cont.) Results of Combinations of Server and Client Encryption Settings

Server Encryption Flag Setting	Client Encryption Flag Setting	Result
`accepted`	`required`	Connection accepted with encryption on if there is overlap between cipher suite settings. Connection rejected if there is not.
`rejected`	`accepted`	Connection accepted, encryption off.
`rejected`	`rejected`	Connection accepted, encryption off.
`rejected`	`requested`	Connection accepted, encryption off.
`rejected`	`required`	Connection rejected.
`requested`	`accepted`	Connection accepted. Encryption on if there is overlap between the cipher suite settings, off if there is not.
`requested`	`rejected`	Connection accepted, encryption off.
`requested`	`requested`	Connection accepted. Encryption on if there is overlap between the cipher suite settings, off if there is not.
`requested`	`required`	Connection accepted with encryption on if there is overlap between cipher suite settings. Connection rejected if there is not.
`required`	`accepted`	Connection accepted with encryption on if there is overlap between cipher suite settings. Connection rejected if there is not.
`required`	`rejected`	Connection rejected.
`required`	`requested`	Connection accepted with encryption on if there is overlap between cipher suite settings. Connection rejected if there is not.
`required`	`required`	Connection accepted with encryption on if there is overlap between cipher suite settings. Connection rejected if there is not.

Note

If automatic client failover is enabled and a failover occurs, encryption attribute settings from the original connection will continue to be used. The failover server must have the same encryption settings as the original server. (See Using Automatic Client Failover in *Oracle TimesTen In-Memory Database Operations Guide* for information about automatic client failover.)

Client Attributes for TLS

These are the client connection attributes to determine settings for TLS for client/server.

For configuration details, see also Task 4: Configure the Client for TLS.

> **Note**
>
> If an attribute is set in both the client DSN definition and the connect string, the connect string setting takes precedence.

- `Wallet`: Specify the wallet directory, as an absolute path, where certificates were placed. If `ttCreateCerts` was used, this is the full path of the `clientWallet` directory.
- `Encryption` (encryption flag): Use one of the following settings. These descriptions assume matching cipher suite settings between the server and client, where applicable.
  - `accepted`: Enable an encrypted session if required or requested by the server; use an unencrypted session otherwise. This is the default.
  - `rejected`: Demand an unencrypted session. (If the client does not support encryption, TimesTen behaves as if this is the setting on the client.) The connection is rejected if the server requires encryption.
  - `requested`: Request an encrypted session if the server allows it (if the server has any setting other than `rejected`); use an unencrypted session otherwise.
  - `required`: Demand an encrypted session. The connection is rejected if the server rejects encryption.

  See Table 4-3 for a summary of the results of each possible combination of settings of this attribute between the server and client, with consideration of the cipher suite settings.
- `CipherSuites`: This lists the cipher suite or suites that can be used, depending also on the server setting. Specify the desired cipher suites, comma-separated and in order of preference. See Cipher Suites in TimesTen for the list of supported cipher suites for TLS 1.3 and 1.2. For TLS to be used, the server and client settings must include at least one common cipher suite.
- `SSLClientAuthentication`: Specifies whether TLS client authentication is required (setting of 1) or not (setting of 0, the default). With client authentication, the server validates an identity presented by the client, and requires an identity (public/private key) in the client wallet. Note that regardless of the client authentication setting, server authentication is performed, where the client validates the server.

  The server and client must have the same `SSLClientAuthentication` setting.

> **Note**
>
> As an alternative to the preceding client connection attributes, these equivalent attributes are available in the instance-level TimesTen configuration file, *`timesten_home`*`/conf/timesten.conf`, on the client. Connection attribute settings take precedence.
>
> - `client_wallet`
> - `client_cipher_suites`
> - `client_encryption`
> - `ssl_client_authentication`
>
> If you have more than one client DSN in a TimesTen instance, these settings apply to all, but can be overridden for each client through the client DSN definition.

Using TLS for Client/Server in TimesTen

This section describes how to configure Transport Layer Security (TLS) for TimesTen client/ server communication

- Task 1: Decide Between Self-Signed Certificates and CA-signed Certificates
- Task 2A: Create Self-Signed Certificates
- Task 2B: Create CA-Signed Certificates
- Task 3: Configure the Server for TLS
- Task 4: Configure the Client for TLS
- (Optional) Task 5: Export Certificates and Configuration in TimesTen
- (Optional) Task 6: Import Certificates and Configuration in TimesTen
- Task 7: Verify Operation of TLS for Client/Server

Tasks 5 and **6** are optional and are provided as a convenience for exporting and importing client TLS configuration; equivalent results can be achieved by manually configuring the client wallet and DSN.

Task 1: Decide Between Self-Signed Certificates and CA-signed Certificates

With a self-signed certificate, the certificate signs itself. TimesTen components trust the certificate because it is explicitly placed in an Oracle wallet. Trust is established locally and must be configured on each participating system. With a Certificate Authority (CA)-signed certificate, the certificate is signed by a trusted CA. Trust is established through the CA certificate chain stored in an Oracle wallet, allowing TimesTen components to validate certificates issued by that CA according to its trust policies. See About Using Certificates with Client/Server for more information.
Determine whether your deployment requires certificates signed by a trusted CA:

- Use self-signed certificates for development or internal environments where manual trust configuration is acceptable.
- Use CA-signed certificates for production environments that require integration with an enterprise PKI.

Proceed to **Task 2A** if you decided to use self-signed certificates, or **Task 2B** if you decide to use CA-signed certificates.

Task 2A: Create Self-Signed Certificates

To generate self-signed certificates, use the TimesTen instance creation utilities to automatically generate the server and client certificates and store them in Oracle wallets.If you prefer to create CA-signed certificates, go to **Task 2B** instead.
You can arrange for certificates to be created when you run the `ttInstanceCreate` utility (from the installation `bin` directory).

Note

- Certificates generated by `ttInstanceCreate` can be used for replication as well as for client/server.
- You can also use the TimesTen `ttCreateCerts` utility manually to generate certificates. This is useful, for example, if you need to regenerate certificates for any reason, such as expiration, or if you have multiple databases on a single TimesTen instance and want to use different certificates for each database. See ttCreateCerts in the *Oracle TimesTen In-Memory Database Reference* .

Set `-serverEncryption` (the encryption flag) and `-serverCipherSuites` (the cipher suite or suites to use) on the `ttInstanceCreate` command line. See Server Attributes for TLS for descriptions of encryption and cipher suites attributes. See Transport Layer Security for TimesTen Client/Server for a list of cipher suites you can use in TimesTen.

This command, to create an instance named `tt261`, will generate certificates in the instance `conf` directory, *`timesten_home`*`/conf`.

```
% installation_dir/bin/ttInstanceCreate -name tt261 -location instances_dir -
serverEncryption required -serverCipherSuites TLS_AES_128_CCM_SHA256
Creating instance in instances_dir/tt261 ...
INFO: Creating certificates, this may take some time ...
ttCreateCerts : certificates created in instances_dir/tt221/conf
...
Instance created successfully.
```

This generates wallets with a root certificate, server certificate, and client certificate and adds the following entries to the instance `timesten.conf` file (the latter two by default):

```
server_encryption=required
server_cipher_suites=TLS_AES_128_CCM_SHA256
client_wallet=timesten_home/conf/clientWallet
server_wallet=timesten_home/conf/serverWallet
```

From the *`timesten_home`*`/conf` directory, what follows shows the `serverWallet` and `clientWallet` directories that are created when you run `ttInstanceCreate`. Each contains a wallet, `cwallet.sso`. (Ignore the `.cert` files and `rootWallet` directory.)

```
% ls
client1.cert  root.cert   server1.cert  sys.odbc.ini      timesten.conf
clientWallet  rootWallet  serverWallet  sys.ttconnect.ini
```

Note

If you want to change the `-serverEncryption` and `-serverCipherSuites` settings for the instance at a later time, you can do so using the `ttInstanceModify` utility, which also has those options. You can copy or move the wallet to a different location and specify the new location of the server wallet using the `ttInstanceModify -serverWallet` option.

After you have generated certificates, you can list information about them using the `-certificateList` option of the TimesTen `ttAdmin` utility, but to use `ttAdmin` you must specify a database on the command line that is defined in the `sys.odbc.ini` file in the *`timesten_home/`*`conf` directory.

The utility looks in the *timesten_home*/conf directory unless the wallets were moved elsewhere, as would be indicated by the Wallet connection attribute in sys.odbc.ini or by server_wallet in the instance-level timesten.conf configuration file.

This example is for a database mydb. Start the TimesTen daemon before you run ttAdmin.

```
% ttDaemonAdmin -start
TimesTen Daemon (PID: 733500, port: 6624) startup OK.
% ttAdmin -certificateList mydb
NAME                                            HOLDER                EXPIRATION
timesten_home/conf/serverWallet/cwallet.sso     CN=server1,C=US       Fri Jul 30
23:08:02 UTC 2032
```

Task 2B: Create CA-Signed Certificates

To use certificates signed by a certificate authority (CA) with TimesTen, you must create and populate a server wallet and a client wallet.If you prefer to create self-signed certificates, go to **Task 2A** instead.
It is assumed that you have obtained a private key and a certificate request (.csr file), typically using openssl; you have sent the certificate request to a certificate authority (CA); and the CA has returned a signed certificate with signing chain.

Starting with .pem files that contain x509 certificates, concatenate the certificates in order from the certificate provided by the CA to the root. (If your certificates are not in .pem format, use openssl or some other appropriate utility to convert them.) In the discussion in the sections that follow, assume the result is called complete.pem and the private key is in privkey.pem. Then you will package the certificates into a pkcs12 file.

At the end of this process, the server wallet will contain the following:

- The entire certificate chain. This consists of the certificate from the CA, the intermediate certificates, and the root certificate.
- The private key of the certificate from the CA.

And the client wallet will contain the following:

- The certificate chain excluding the CA certificate. This consists of the intermediate certificates and the root certificate.

Create the Server Wallet

To use CA-signed certificates with TimesTen, you must manually import certificates into the server wallet and the client wallet. Complete the following steps for the client wallet. These steps assume intermediate certificates certsigner1.pem and certsigner2.pem.

1. Concatenate the certificates in the correct order. Starting with .pem files that contain x509 certificates, concatenate the certificates in order from the certificate received from the CA to the root. For example:

   ```
   % cat cert.pem certsigner1.pem certsigner2.pem root.pem > complete.pem
   ```

2. Package the certificates into a pkcs12 file, as in the following example. Use any password in the openssl command. It will not be in the Oracle Wallet. In this example, the concatenated certificates are in complete.pem and the private key is in privkey.pem.

   ```
   % openssl pkcs12 -export -in complete.pem -inkey privkey.pem -out
   server.p12 -password pass:mypwd
   ```

3. Create an empty `auto_login_only` Oracle Wallet using the TimesTen `ttCreateCerts` utility. See ttCreateCerts in the *Oracle TimesTen In-Memory Database Reference* for information about `ttCreateCerts` and the `-run` option.

   ```
   % ttCreateCerts -run "wallet create -wallet servertWallet -auto_login_only"
   ```

4. Put the certificates and private key into the wallet. For example:

   ```
   % ttCreateCerts -run "wallet import_pkcs12 -wallet serverWallet -
   auto_login_only -pkcs12file server.p12 -pkcs12pwd mypwd"
   ```

5. Verify the server wallet. This includes confirming that the intermediate and root certificates are “Trusted Certificates” and the new certificate is a “User Certificate”. (A “User Certificate” means that TimesTen has the private key for it.) Output should be of the basic form shown.

   ```
   % ttCreateCerts -run "wallet display -wallet path/serverWallet"
   Requested Certificates:
   User Certificates:
   Subject:
   CN=www.example.com,O=xxxxxxxxxxxxxx,L=xxxxxxxxxxxxxx,ST=xxxxxxxxxxxxxx,C=US
   Trusted Certificates:
   Subject:        CN=xxxxxxxxxxxxxx SHA-256 Private
   Root,O=xxxxxxxxxxxxxx,C=US
   Subject:        CN=xxxxxxxxxxxxxx SHA-256 Private
   Intermediate,O=xxxxxxxxxxxxxx,C=US
   ```

Create the Client Wallet

To use CA-signed certificates with TimesTen, you must manually import certificates into the server wallet and the client wallet. Complete the following steps for the client wallet. These steps assume intermediate certificates `certsigner1.pem` and `certsigner2.pem`.

1. Concatenate the certificates in the correct order. Starting with `.pem` files that contain x509 certificates, concatenate the certificates in order from the certificate received from the CA to the root. For example:

   ```
   % cat cert.pem certsigner1.pem certsigner2.pem root.pem > complete.pem
   ```

2. Package the certificates into a `pkcs12` file, as in the following example. Use any password in the `openssl` command. It will not be in the Oracle Wallet. In this example, the concatenated certificates are in `complete.pem` and the private key is in `privkey.pem`.

   ```
   % openssl pkcs12 -export -in complete.pem -inkey privkey.pem -out
   server.p12 -password pass:mypwd
   ```

3. Create an empty `auto_login_only` Oracle Wallet using the TimesTen `ttCreateCerts` utility. See ttCreateCerts in the *Oracle TimesTen In-Memory Database Reference* for information about `ttCreateCerts` and the `-run` option.

   ```
   % ttCreateCerts -run "wallet create -wallet clientWallet -auto_login_only"
   ```

4. Import the certificates and private key into the wallet.

   ```
   % ttCreateCerts -run "wallet import_pkcs12 -wallet clientWallet -
   auto_login_only -pkcs12file server.p12 -pkcs12pwd mypwd"
   ```

5. Verify the client wallet. This includes confirming that the intermediate and root certificates are “Trusted Certificates” and the new certificate is a “User Certificate”. (A “User Certificate” means that TimesTen has the private key for it.) Output should be of the basic form shown.
6. Proceed to **Task 3**.

Task 3: Configure the Server for TLS

You can configure TLS for the server in the following ways.

- The encryption flag and cipher suite(s) are specified in the `ttInstanceCreate` command as shown earlier. Later, there are additional TLS configuration attributes you can set as well. In particular, set the wallet location.
- At the instance level, you can add or update TLS attributes in the `timesten.conf` file in the *`timesten_home`*`/conf` directory, such as through the `server_wallet`, `server_cipher_suites`, and `server_encryption` attributes. (Recall that initial values for `server_cipher_suites` and `server_encryption` were set through `ttInstanceCreate`.) Values in `timesten.conf` serve as default values for any database on the instance.
- At the database level, server attributes for TLS can be set in the server DSN definition in *`timesten_home`*`/conf/sys.odbc.ini`. For a given database, these settings override the instance-level settings in `timesten.conf`.

This excerpt from a server DSN definition specifies where `ttInstanceCreate` placed the server wallet directory:

```
[mydb]
Driver=timesten_home/install/lib/libtten.so
DataStore=databases_dir/mydb
...
Wallet=timesten_home/conf/serverWallet
```

Make sure that you also update the new wallet directory in `timesten.conf` file using `ttInstanceModify -serverWallet` utility or modifying the `server_wallet` attribute in the `timesten.conf` file. Alternatively, if you're using the `sys.odbc.ini` file to specify the TLS attributes, modify the `Wallet` attribute.

See Server Attributes for TLS for information about available configuration attributes.

Task 4: Configure the Client for TLS

Configure TLS for the client in the client DSN definition. Manually set `Encryption` (the encryption flag), `CipherSuites` (the cipher suite(s) to use), and `Wallet` (pointing to the client wallet directory) in the client DSN definition in `sys.odbc.ini` on the server.

For example, for a database `mydb`:

```
[mydbCS]
TTC_SERVER=mydb_CS
TTC_SERVER_DSN=mydb
Wallet=timesten_home/conf/clientWallet
Encryption=required
CipherSuites=TLS_AES_128_CCM_SHA256
```

Also be aware of these alternatives:

- You can manually copy the client wallet from the server and edit the `odbc.ini` file on the client to set the correct attribute values. The `ttAdmin` utility has a `-clientExportAll` option that includes all the required files to copy to your client. See Transport Layer Security in *Oracle TimesTen In-Memory Database Reference*.
- You can set attribute values in the connect string for a particular connection, such as from the `ttIsql` utility:

  ```
  Command> connect
  "Driver=...;DataStore=...;Encryption=required;CipherSuites=TLS_AES_128_CCM_SHA256";
  ```

- You can set equivalent attributes at the instance level in the TimesTen configuration file, `*timesten_home*/conf/timesten.conf`, on the client. These settings, including `client_encryption`, `client_cipher_suites`, and `client_wallet`, would be used as default values. Any connection attribute settings for a particular connection take precedence.

See Client Attributes for TLS for information about available configuration attributes.

(Optional) Task 5: Export Certificates and Configuration in TimesTen

The `ttAdmin` utility has a `-clientExportAll` option that outputs a ZIP file containing the client wallet, a sample`sys.odbc.ini` file that can be used in accessing the database, and other files (such as `tnsnames.ora` file) as applicable.

The `-clientExportAll` option requires:

- A server DSN name (specified on the command line)
- The server DSN to be defined in the server `sys.odbc.ini` file.
- The server DSN to be configured for TLS, including a wallet, encryption, cipher suites.

The following example shows a TLS-enabled server DSN named `mydbCS` in the server `sys.odbc.ini` file.

```
[mydbCS]
TTC_SERVER=mydb_CS
TTC_SERVER_DSN=mydb
Encryption=required
CipherSuites=TLS_AES_128_CCM_SHA256
Wallet=mydbCSWallet
```

Run the `ttAdmin` utility from the `*timesten_home*/bin` directory on the server. On the command line, specify the output ZIP file path and name, followed by the server DSN name. specify the desired ZIP file path and name and the client DSN.

The wallet in the exported ZIP file contains the CA public key used to verify server certificates and, when client authentication is enabled, a client certificate.

The following command exports the client configuration for the database `mydbCS` and creates a file named `exports.zip` in the `*timesten_home*/info` directory:

```
% ttAdmin -clientExportAll timesten_home/info/exports.zip mydbCS
Client definitions exported to timesten_home/info/exports.zip
```

The `exports.zip` file contains a sample `sys.odbc.ini` file and a directory named after the server DSN, `mydbCSWallet` in this example. That directory contains the client wallet, `cwallet.sso`, that was created during instance creation. The exported ZIP file does not contain the server wallet.

(Optional) Task 6: Import Certificates and Configuration in TimesTen

Run the `ttClientImport` utility from the client to import the contents of the ZIP file created by the `ttAdmin -clientExportAll` command.

Typically, copy the exported ZIP file from the server to a desired location on the client system, and then specify that location on the `ttClientImport` command line. The `ttClientImport` utility imports the client wallet, the `sys.odbc.ini` file, and other configuration files, as applicable.

The utility creates a wallet directory named after the server DSN, `mydbCSWallet` in this example, under the `wallets` directory under *`timesten_home`*`/conf`.

```
% ttClientImport path/exports.zip
Client definitions imported.
```

The client `sys.odbc.ini` is updated to add a client DSN with the TLS settings from the exported configuration. If a client DSN with the same name already exists, it is replaced. The wallet path in the imported `sys.odbc.ini` file is updated to reflect the actual wallet location under the `wallets/mydbCSWallet`directory path.

```
[mydbCS]
TTC_SERVER=mydb_CS
Encryption=Required
CipherSuites=TLS_AES_128_CCM_SHA256
Wallet=timesten_home/conf/wallets/mydbCSWallet
```

The following commands show the resulting directory structure on the client system:

```
% ls
sys.odbc.ini  sys.ttconnect.ini  timesten.conf  wallets
% ls wallets
mydbCSWallet
% ls wallets/mydbCSWallet
cwallet.sso
```

After the import is completed, and assuming the client and server are configured with compatible encryption and cipher suite settings, clients can connect to the server using TLS. See Task 7: Verify Operation of TLS for Client/Server.

See ttAdmin and ttClientImport in *Oracle TimesTen In-Memory Database Reference* for additional information about these utilities.

Task 7: Verify Operation of TLS for Client/Server

If TLS is configured on both the server and the client with sufficiently matching settings of `Encryption` and `CipherSuite`, TLS is used as soon as the connection is established.

You can confirm this by calling `sqlgetconnectattr tt_tls_session` from `ttIsqlCS` on the client. A return value of 1 indicates TLS is being used.

The following set of examples shows the results of several combinations of encryption settings on the server and client.

Scenario 1: Encryption is requested on the server and on the client with the same cipher suite settings. The connection is successful and TLS is used.

Server DSN definition:

```
[mydb]
Driver=timesten_home/install/lib/libtten.so
DataStore=/db/databases/mydb
PermSize=512
TempSize=128
LogBufMB=256
LogFileSize=256
LogDir=/db/logs
DatabaseCharacterSet=AL32UTF8
OracleNetServiceName=ttorcl
Wallet=timesten_home/conf/mywallets/serverWallet
Encryption=requested
CipherSuites=TLS_AES_128_CCM_SHA256
```

Client DSN definition:

```
[mydbCS]
TTC_SERVER=myserverhost.example.com
TTC_SERVER_DSN=mydb
UID=myuser
PWD=welcome
Wallet=timesten_home/conf/mywallets/clientWallet
Encryption=requested
CipherSuites=TLS_AES_128_CCM_SHA256
```

Connect, executing `ttIsqlCS` from the *`timesten_home`*`/bin` directory (output formatted for readability):

```
% ttIsqlCS mydbCS

Copyright (c) 1996, 2025, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

connect "DSN=mydbCS";
Connection successful: DSN=mydbCS;TTC_SERVER=myserverhost.example.com;
TTC_SERVER_DSN=mydb;UID=myuser;DATASTORE=/db/databases/mydb;
DATABASECHARACTERSET=AL32UTF8;CONNECTIONCHARACTERSET=US7ASCII;LOGFILESIZE=256;
LOGBUFMB=256;LOGDIR=/db/logs;PERMSIZE=512;TEMPSIZE=128;
ORACLENETSERVICENAME=ttorcl;(SERVER)ENCRYPTION=Requested;
(SERVER)WALLET=file:timesten_home/conf/mywallets/serverWallet;
(client)Encryption=Requested;
(client)Wallet=/timesten_home/conf/mywallets/clientWallet;
(client)CipherSuites=TLS_AES_128_CCM_SHA256;
(Default setting AutoCommit=1)
```

Confirm TLS is enabled:

```
Command> sqlgetconnectattr tt_tls_session;
TT_TLS_SESSION = 1 (SQL_TRUE)
```

Scenario 2: Encryption is requested on the server and on the client but with mismatched cipher suite settings. The connection is successful but a warning message indicates that TLS is not used. (Except for what is shown here, settings are the same as in Scenario 1.)

From the server DSN definition:

```
CipherSuites=TLS_AES_128_CCM_SHA256
```

From the client DSN definition:

```
CipherSuites=TLS_CHACHA20_POLY1305_SHA256
```

Connect:

```
% ttIsqlCS mydbCS

Copyright (c) 1996, 2025, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

connect "DSN=mydbCS";

Warning 01000: Unable to create requested TLS session; unencrypted session
created. Check Wallet and CipherSuites on client and server. SSL error: SSL
Fatal Alert

Connection successful:
...
```

Scenario 3: Encryption is accepted on the server and on the client. With these encryption values, the TLS encryption is not enforced. The connection is successful but TLS is not used. (Except for what is shown here, settings are the same as in Scenario 1.) See also Table 4-3.

From the server DSN definition:

```
Encryption=accepted
```

From the client DSN definition:

```
Encryption=accepted
```

Connect:

```
% ttIsqlCS mydbCS

Copyright (c) 1996, 2025, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

connect "DSN=mydbCS";
Connection successful:
...
Command> sqlgetconnectattr tt_tls_session;
TT_TLS_SESSION = 0 (SQL_FALSE)
```

Scenario 4: Encryption is required on the client but rejected on the server. The connection attempt is unsuccessful. (Except for what is shown here, settings are the same as in Scenario 1.)

From the server DSN definition:

```
Encryption=rejected
```

From the client DSN definition:

```
Encryption=required
```

Attempt to connect:

```
% ttIsqlCS mydbCS

Copyright (c) 1996, 2022, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

connect "DSN=mydbCS";
HY000: Connection rejected: inconsistent encryption attributes
The command failed.
Done.
```

Transport Layer Security for TimesTen Replication

When you use TimesTen replication in TimesTen Classic, you can optionally configure and use Transport Layer Security (TLS) for secure, encrypted network communication between replication agents or between TimesTen utilities (such as `ttRepAdmin`) and replication agents. Mutual authentication is used for all connections.

These are the main steps for using TLS with TimesTen replication:

- Task 1: Generate Certificates for Replication
- Task 2: Copy Certificates for Replication
- Task 3: Configure TLS for Replication
- Task 4: Activate TLS for Replication
- Task 5: Check Operation of TLS for Replication

Task 1: Generate Certificates for Replication

You can create certificates for replication with the `ttInstanceCreate` utility when you create a TimesTen instance, or by using the `ttCreateCerts` utility directly.

Using `ttInstanceCreate` would be essentially the same as shown earlier in this chapter for client/server, in Task 2A: Create Self-Signed Certificates. Note that `ttInstanceCreate` uses the `ttCreateCerts` utility to generate certificates. Or see ttCreateCerts in the *Oracle TimesTen In-Memory Database Reference* for information about `ttCreateCerts` syntax, options, and usage in order to use it directly.

If you will be using certificates for both replication and client/server, it is preferable to use separate certificates for the two features. You can use the `ttCreateCerts` utility to generate additional certificates as needed.

Note the following regarding certificates generated by TimesTen:

- Certificates produced are self-signed and stored in an Oracle Wallet.
- Because the certificates are self-signed, they cannot be revoked. But certificates can be regenerated as needed.
- The root CA has a default expiration time. It is the user’s responsibility to track this. When the root CA expires, all certificates must be regenerated. When the root CA is regenerated, it must be copied to each TimesTen instance.
- Instances will store the root certificate (the public key) in a local wallet.
- Wallets produced are auto-login or single-sign-on (SSO) wallets, without a password. Access to wallets is controlled by file system permissions.
- The wallets are platform-independent.

TimesTen uses Oracle Wallets to store certificates. For general information about these wallets, also referred to as "keystores", refer to How the Keystore for the Storage of TDE Master Encryption Keys Works in *Oracle Database Advanced Security Guide*.

Task 2: Copy Certificates for Replication

After you generate certificates for replication, copy them to the other TimesTen instances. Recall the resulting wallets from the example in ttCreateCerts in the *Oracle TimesTen In-Memory Database Reference*.

```
% ls timesten_home/conf/wallets
client1.cert  clientWallet  root.cert  rootWallet  server1.cert  serverWallet
% ls timesten_home/conf/wallets/serverWallet
cwallet.sso
```

For TLS for replication, only `serverWallet` is used. Copy the `serverWallet` directory, which includes the root certificate, to the desired location. This is preferably the same location on each TimesTen instance.

On each instance:

```
% mkdir timesten_home/conf/wallets
[...Copy serverWallet from the instance where it was created...]
% cd timesten_home/conf/wallets
% ls
serverWallet
% ls serverWallet
cwallet.sso
```

Task 3: Configure TLS for Replication

To use TLS for replication, set TLS attributes in the `timesten.conf` file on each TimesTen instance. The settings are read on each instance by the replication agent and by utilities that may communicate with the agent.

Tip

Generate and copy certificates before you configure TLS for replication. Otherwise, configuration may trigger an error condition where replication agents start up and try to access certificates that do not exist yet.

- `replication_cipher_suite`: Specify the cipher suite or suites that can be used for encrypting communications to and from the replication agent, depending also on the other replication peer's setting. Specify the desired cipher suites, comma-separated. See Cipher Suites in TimesTen for the list of supported cipher suites for TLS 1.3 and 1.2. This setting is required. There is no default.
- `replication_wallet`: Specify the path to the wallet directory—the directory where you placed the certificates that you generated. This setting is required. There is no default location. It is suggested, but not required, to use the same location and directory name on each TimesTen instance. (In the example in the preceding section, Task 2: Copy Certificates for Replication, this would be *`timesten_home`*`/conf/wallets/serverWallet`.)
- `replication_ssl_mandatory`: Specifies whether it is mandatory to have consistent TLS configuration between TimesTen instances—specifically, whether TLS is configured through `replication_cipher_suite` and `replication_wallet` settings, and what cipher suite is specified. If there is a mismatch between the current instance and the replication peer, then TimesTen behavior is determined as follows:

- On an instance with a setting of `replication_ssl_mandatory=0` (not mandatory, the default), replication proceeds between that instance and the replication peer, but TLS is not used for communications between the replication agents as long as the settings are inconsistent.
- On an instance with a setting of `replication_ssl_mandatory=1` (mandatory), replication cannot proceed between this instance and the replication peer until the settings are made consistent.

Note

- For these configuration changes to take effect on any given instance, you must restart the replication agent. (It is not necessary to restart the TimesTen daemon.)
- If the `replication_cipher_suite` value is invalid or the suite is not supported by TimesTen, an error is reported and replication cannot function until the problem is resolved.
- If `replication_cipher_suite` is set but `replication_wallet` is not, or no certificates are found in the specified location, an error is reported and replication cannot function until the problem is resolved.

Task 4: Activate TLS for Replication

Once TLS is configured on all TimesTen instances, with certificates located in the specified `replication_wallet` directories and the desired cipher suite specified in the `replication_cipher_suite` settings, restarting the replication agents will activate TLS, resulting in it being used for communication to and from the replication agents.

There are two ways to activate TLS:

- Switch Online to TLS for Replication
- Switch All Instances Simultaneously to TLS for Replication (Offline)

Switch Online to TLS for Replication

If you have an existing replication scheme that is not using TLS, you can perform an online switchover to TLS by restarting the replication agents one at a time as replication continues to function.

1. On each instance, confirm `replication_wallet` is set to indicate where the certificates are located. (In the example in Task 2: Copy Certificates for Replication, this would be `*timesten_home*/conf/wallets/serverWallet`.)
2. On each instance, confirm `replication_cipher_suite` is set to indicate the cipher suite you are using.
3. On each instance, confirm `replication_ssl_mandatory=0`. This allows you to update the TimesTen instances to start using TLS one at a time.
4. On each instance (one at a time, in succession), stop and restart the replication agent:

```
% ttAdmin -repStop DSN
% ttAdmin -repStart DSN
```

For example, assume the following:

- There is an active standby pair with databases `rep1` on `host1` and `rep2` on `host2`, with subscriber `rep3` on `host3`.
- Certificates were generated on `rep1` and placed in `/swdir/mywalletloc`, then copied to the same location on `rep2` and `rep3`.

Complete these steps, as replication continues to function, to use TLS for communications to and from each of the replication agents:

1. Use these TLS settings in the `timesten.conf` file on each instance:

   ```
   replication_wallet=/swdir/mywalletloc
   replication_cipher_suite=TLS_AES_256_GCM_SHA38
   replication_ssl_mandatory=0
   ```

2. Restart the replication agent on each instance, one at a time.

   On `host1`:

   ```
   % ttAdmin -repStop rep1
   % ttAdmin -repStart repl
   ```

   On `host2`:

   ```
   % ttAdmin -repStop rep2
   % ttAdmin -repStart rep2
   ```

   On `host3`:

   ```
   % ttAdmin -repStop rep3
   % ttAdmin -repStart rep3
   ```

Switch All Instances Simultaneously to TLS for Replication (Offline)

If you want TLS to start and be enforced on all instances immediately and simultaneously, you must shut down all replication agents, stopping replication, before setting `replication_ssl_mandatory=1` on each instance.

1. On all instances, stop the replication agent:

   ```
   % ttAdmin -repStop DSN
   ```

   > **Note**
   >
   > If you are using Oracle Clusterware, you can accomplish this for all instances with a single command using the `ttCWAdmin` utility from any instance in the cluster:
   >
   > ```
   > % ttCWAdmin -stop -dsn DSN
   > ```

2. On all instances, confirm `replication_wallet` is set to indicate where the certificates are located.
3. On all instances, confirm `replication_cipher_suite` is set to indicate the cipher suite you are using.
4. On all instances, confirm `replication_ssl_mandatory=1`.

   This requires all replication agents to be shut down at once, and all `timesten.conf` files to be updated while all the replication agents are down.
5. On all instances, restart the replication agent:

```
% ttAdmin -repStart DSN
```

> **Note**
>
> If you are using Oracle Clusterware, you can accomplish this for all instances with a single command using the `ttCWAdmin` utility from any instance in the cluster:
>
> ```
> %  ttCWAdmin -start -dsn DSN
> ```

Task 5: Check Operation of TLS for Replication

The `ttRepAdmin` utility `-showstatus -detail` option indicates whether the replication agent transmitters and receivers are using TLS (indicated as "SSL").

For example:

```
TRANSMITTER thread(s) (TRANSMITTER(M):140427924887296):
For                       : REP1 (track 0) (SSL)
   Start/Restart count    : 1
   Current state          : STATE_META_PEER_INFO

RECEIVER thread(s) (RECEIVER:140427327059712):
For                       : REP1 (track 0) (SSL)
   Start/Restart count    : 1
   Current state          : STATE_RCVR_READ_NETWORK_LOOP
   Current DB context     : 0x7fb7bc4a41e0
```

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

> **Note**
>
> In order for you to see this output, the replication agents on the master and subscribing systems must be running and connected to each other.

5

Use Restrict Mode to Secure TimesTen Utilities

TimesTen utilities such as `ttBulkCp`, `ttIsql`, and `ttMigrate` support a restrict mode (with the `-restrict` option) that enhances security by preventing operations that might overwrite files or expose the system to vulnerabilities.

This chapter includes the following topics:

About Using sudo

UNIX system administrators can use the sudoers configuration to grant specific operating system users the ability to run certain commands as a different user, without requiring a password.

This mechanism helps to limit user privileges while still allowing them to execute well-defined commands that require elevated privileges. The `/etc/sudoers` file defines which users can run which commands as target users. For details on configuring sudo, run the `man sudoers` and `man sudo` commands on your UNIX system.

If you enter a command that creates a file, a malicious user could overwrite critical system files. Similarly, if the command spawns a shell, an attacker might be able to run any arbitrary command as the target user. To mitigate these risks, three TimesTen utilities such as `ttBulkCp`, `ttIsql`, and `ttMigrate` have a restricted mode (`-restrict`) that limits specific capabilities of the programs.

This example shows the use of `-restrict` with the TimesTen `ttIsql` utility (see Use ttIsql with Restrict Mode). In the `sudoers` file, add the following line to allow `scott` (existing user) run a TimesTen utility `ttIsql` as the `ttadmin` user without entering a password with restrict mode:

```
scott ALL=(ttadmin) NOPASSWD: /TimesTen/instance1/bin/ttIsql -restrict
```

Allows the `scott` user to run the following command:

```
$ sudo -u ttadmin /TimesTen/instance1/bin/ttIsql -restrict
```

Overview of Restrict Mode in TimesTen Utilities

When you run the TimesTen utilities (`ttBulkCp`, `ttIsql`, and `ttMigrate`) with restrict mode, the utilities block certain operations, particularly those that could overwrite existing files or pose security vulnerabilities.

The utilities, along with their client-server versions such as `ttBulkCpCS`, `ttIsqlCS`, and `ttMigrateCS`, add the `-restrict` command-line option. See ttBulkCp, ttIsql, and ttMigrate in the *Oracle TimesTen In-Memory Database Reference*.

Instead of using sudo to run the `ttBulkCp`, `ttIsql`, and `ttMigrate` utilities with the `-restrict` option, users can use their client-server variants (`ttBulkCpCS`, `ttIsqlCS`, and `ttMigrateCS`). These client-server variants can be run by operating system users other than the instance administrator operating system user. There is no need to use the `-restrict` option, as operating system's file permissions will prevent the overwriting of critical TimesTen files. Since these client-server variants prevent database connections as the instance administrator, users must connect using database account credentials with the appropriate privileges. The only limitation of the client-server variants is that they do not support database creation.

Use ttBulkCp with Restrict Mode

When you run the `ttBulkCp` (or `ttBulkCpCS`) utility with the `-restrict` option, the command-line options listed in the table below prevents you from overwriting existing files.

The system displays an error message if you try to overwrite an existing file. If the file does not exist, then `ttBulkCp` creates it.

Command-line Option	**Reason to Block**	**Error Message**
`-o <datafile>`	Overwrites an existing data file.	cannot overwrite the existing file *dataFile* in the restrict mode
`-e <errfile>`	Overwrites an existing error file.	cannot overwrite the existing error file *errorFile* in the restrict mode

If the command does not include the `-o` or `-e` option, no output file is present, and there is nothing to restrict. The `-restrict` option prevents overwriting only if there is an existing data file or an error file.

This example shows how the `-restrict` option prevents `ttBulkCp` from overwriting an existing file called `sales.dat` when exporting the contents of the table `mytbl` to that file:

```
$ ttBulkCp -o -restrict mystore mytbl sales.dat
ttBulkCp: cannot overwrite existing file sales.dat in restrict mode.0/0 rows
copied.
```

This example shows how the `-restrict` option prevents `ttBulkCp` from importing data if the specified error file `errors.log` already exists:

```
$ ttBulkCp -i -restrict -e errors.log mystore mytbl sales.dat
ttBulkCp: cannot overwrite existing error file errors.log in restrict mode.
```

Use ttIsql with Restrict Mode

When you invoke `ttIsql` (or `ttIsqlCS`) with the `-restrict` option, the commands listed in the table below are blocked and displays an error if you attempt to run them.

Command	**Reason to Block**	**Error Message**
`cachesqlget`	Overwrites critical files.	cachesqlget commands are restricted.

Command	Reason to Block	Error Message
`connect`	Allow one connection. Otherwise, you can connect as the instance admin to any database managed by that instance.	Restricted: A connection was already established in restrict mode.
`edit`	Overwrites critical files. Many editors allow to run host commands.	Edit commands are restricted.
`host`	Spawns a shell or run operating system command.	Host commands are restricted.
`putenv`	Manipulates `PATH`, `LD_LIBRARY_PATH`, `TIMESTEN_HOME` and other critical environment variables.	PutEnv commands are restricted.
`savehistory`	Overwrites critical files.	SaveHistory commands are restricted.
`set signalbehavior 1` `set signalbehavior waitfordebugger`	If enabled and `ttIsql` crashes, the operating system runs the `ptrace` command on the `PATH`, which may pose a security risk.	'Set signalbehavior waitfordebugger' commands are restricted.
`spool`	Overwrites critical files.	Spool commands are restricted.

When the `ttIsql` utility is invoked with `-restrict` option, this utility blocks certain `ttIsql` commands to enhance security. The following example shows the error messages that appears when a user tries to run these restricted commands:

```
$ ttIsql -restrict
Copyright (c) 1996, 2025, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

Command> spool $TIMESTEN_HOME/conf/timesten.conf;
Spool commands are restricted.

Command> savehistory -f $TIMESTEN_HOME/conf/timesten.conf ;
SaveHistory commands are restricted.

Command> cachesqlget scott.mycachegroup install $TIMESTEN_HOME/conf/
timesten.conf;
cachesqlget commands are restricted.

Command> putenv TIMESTEN_HOME /tmp/i2;
PutEnv commands are restricted.
```

Restricting the `connect` command in ttIsql

When `ttIsql` runs in restrict mode (using the `-restrict` option), it enforces strict control on database connections to improve security. This initial connection must be made either by specifying a connect string or `DSN` as a command-line option or by running the `connect` command within `ttIsql`.

In restrict mode, only one connection can be made in a `ttIsql` session.

```
Command> connect database1;
Connection successful:
```

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

When you attempt to create a second connection, `ttIsql` shows the following error:

```
Command> connect database2;
Restricted: A connection was already established in restrict mode.
The command failed.
```

Even if the initial connection is closed, if you attempt to establish another connection within the same session, it fails. To open a new connection, you must start a new `ttIsql` session. With restrict mode, you can now control which database the user is allowed to connect by specifying the connect string in the `/etc/sudoers` file.

The reason for this restriction is that there could be multiple databases managed by the same TimesTen instance. In restrict mode, administrators can ensure that the user can only connect to a designated database instance. To accomplish this, add the connect string in the `/etc/sudoers` file:

```
scott ALL=(ttadmin) NOPASSWD: /TimesTen/instance1/bin/ttIsql -restrict -
connstr DSN=database1 *
```

This example demonstrates how the `ttadmin` user gives the `scott` user permission to specify the same connect string. By restricting `ttIsql` to have only one connection, the user `scott` can only connect to the database (`database1`) mentioned in the `/etc/sudoers` configuration file.

Since additional connections are blocked in restrict mode, the user cannot change databases or modify connection attributes during the session.

Use ttMigrate with Restrict Mode

When you invoke `ttMigrate` (or `ttMigrateCS`) with the `-restrict` option, the command-line option listed in the table below prevents you from overwriting existing files.

The table shows the error message when `ttMigrate` fails. If the file does not exist, `ttMigrate` creates it.

Command-line Option	**Reason to Block**	**Error Message**
`-c <datafile>`	Overwrites an existing data file.	not overwriting the existing file *dataFile* in restrict mode

This example shows how the `-restrict` option prevents `ttMigrate` from overwriting an existing file.

```
$ ttMigrate -c -restrict sampledb sales.ttm
ttMigrate: not overwriting the existing file sales.ttm in restrict mode.
```