Oracle® TimesTen In-Memory Database

Security Guide

Release 18.1

E79756-05

November 2020

ORACLE®

Oracle TimesTen In-Memory Database Security Guide, Release 18.1

E79756-05

Contents

Preface vii

Audience vii
Related documents vii
Conventions viii
Documentation Accessibility ix

What's New xi

New features in Release 18.1.4.4.0 xi
New features in Release 18.1.4.1.0 xi

1 Authentication in TimesTen

Overview of TimesTen users 1-1
Managing TimesTen users 1-2
Creating or identifying a database user 1-2
Changing the password of an internal user 1-3
Specifying a Client/Server user and password 1-3
Specifying a user and password for TimesTen utilities 1-4
Dropping a user from the database 1-4
Cache group users 1-4
Password management 1-5
Password management features 1-5
Password lifetime and grace time 1-6
Limitations on password reuse 1-6
Maximum failed login attempts and password lock time 1-6
Profile for password management 1-6

2 Authorization in TimesTen

Privileges overview 2-1
Granting and revoking privileges 2-2
Functionality of privileges 2-2
Overview of system privileges 2-3
Overview of object privileges 2-3
Privileges for TimesTen utilities 2-3
Overview of the PUBLIC role 2-4
System privileges 2-4

Instance administrator privileges 2-4
Ownership and privileges for database and log directories 2-5
Administrative privileges 2-5
Privileges to connect to the database 2-6
ANY keyword 2-6
ALL PRIVILEGES 2-6
Privilege hierarchy 2-7
Additional system privileges 2-7
Privileges through the PUBLIC role 2-8
Overview of privileges to create, alter, or drop objects 2-9
Privileges to create database objects 2-9
Privileges to alter database objects 2-10
Privileges to drop database objects 2-10
Privileges for SQL objects 2-10
Object privileges for tables 2-11
Object privileges for views 2-12
Object privileges for sequences 2-13
Object privileges for materialized views 2-13
Object privileges for synonyms 2-13
ALL object privileges 2-14
Privileges for PL/SQL objects 2-14
Privileges for PL/SQL statements and operations 2-14
Invalidated objects 2-18
Definer's rights and invoker's rights (AUTHID clause) 2-20
Privileges for cache groups 2-24
Cache administration user privilege 2-24
Cache manager user privilege 2-24
Cache user privilege 2-25
Cache group system privileges 2-25
Cache group object privileges 2-25
User privilege views 2-26

3 Encryption in TimesTen

Certificates for Transport Layer Security 3-1
About using certificates with TimesTen 3-1
Generation of certificates for TimesTen 3-2
Features of the ttCreateCerts utility 3-2
Using ttCreateCerts 3-3
Transport Layer Security for TimesTen Client/Server 3-4
Introduction to TLS for Client/Server 3-5
Certificates for TimesTen Client/Server 3-5
Configuration for TLS for Client/Server 3-6
Configuration on the server 3-6
Configuration on the client 3-8
Operation of TLS for Client/Server 3-9
Transport Layer Security for TimesTen replication 3-12
Introduction to TLS for replication 3-12

Certificates for TimesTen replication 3-12
Configuration for TLS for replication 3-13
Activation of TLS for replication 3-14
Switching online to TLS for replication 3-14
Switching all instances simultaneously to TLS for replication (offline) 3-15
Operation of TLS for replication 3-16

4 Security for the TimesTen Kubernetes Operator

Privileges for the TimesTen Kubernetes Operator 4-1
Authorization for users of the TimesTen Kubernetes Operator 4-1
Encryption for the TimesTen Kubernetes Operator 4-2

Index

Preface

Oracle TimesTen In-Memory Database (TimesTen) is a relational database that is memory-optimized for fast response and throughput. The database resides entirely in memory at runtime and is also persisted to the file system.

- Oracle TimesTen In-Memory Database in classic mode, or TimesTen Classic, refers to single-instance and replicated databases (as in previous releases).
- Oracle TimesTen In-Memory Database in grid mode, or TimesTen Scaleout, refers to a multiple-instance distributed database. TimesTen Scaleout is a grid of interconnected hosts running instances that work together to provide fast access, fault tolerance, and high availability for in-memory data.
- TimesTen alone refers to both classic and grid modes (such as in references to TimesTen utilities, releases, distributions, installations, actions taken by the database, and functionality within the database).
- TimesTen Application-Tier Database Cache, or TimesTen Cache, is an Oracle Database Enterprise Edition option. TimesTen Cache is ideal for caching performance-critical subsets of an Oracle database into cache tables within a TimesTen database for improved response time in the application tier. Cache tables can be read-only or updatable. Applications read and update the cache tables using standard Structured Query Language (SQL) while data synchronization between the TimesTen database and the Oracle database is performed automatically. TimesTen Cache offers all of the functionality and performance of TimesTen Classic, plus the additional functionality for caching Oracle Database tables.
- TimesTen Replication features, available with TimesTen Classic or TimesTen Cache, enable high availability.

TimesTen supports standard application interfaces JDBC, ODBC, and ODP.NET; Oracle interfaces PL/SQL, OCI, and Pro*C/C++; and the TimesTen TTClasses library for C++.

Audience

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

Related documents

TimesTen documentation is available at
https://docs.oracle.com/database/timesten-18.1.

Oracle Database documentation is also available on the Oracle documentation website. This may be especially useful for Oracle Database features that TimesTen supports but does not attempt to fully document, such as OCI and Pro*C/C++.

Conventions

TimesTen supports multiple platforms. Unless otherwise indicated, the information in this guide applies to all supported platforms. The term Windows applies to all supported Windows platforms. The term UNIX applies to all supported UNIX platforms. The term Linux is used separately. Refer to "Platforms and compilers" in *Oracle TimesTen In-Memory Database Release Notes* (`README.html`) in your installation directory for specific platform versions supported by TimesTen.

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

This document uses the following text conventions:

Convention	Meaning
italic	Italic type indicates terms defined in text, book titles, or emphasis.
`monospace`	Monospace type indicates code, commands, URLs, function names, attribute names, directory names, file names, text that appears on the screen, or text that you enter.
`italic monospace`	Italic monospace type indicates a placeholder or a variable in a code example for which you specify or use a particular value. For example: `LIBS = -Lt`*`imesten_home`*`/install/lib -ltten` Replace *`timesten_home`* with the path to the TimesTen instance home directory.
[ ]	Square brackets indicate that an item in a command line is optional.
{ }	Curly braces indicated that you must choose one of the items separated by a vertical bar (|) in a command line.
|	A vertical bar (or pipe) separates alternative arguments.
. . .	An ellipsis (. . .) after an argument indicates that you may use more than one argument on a single command line. An ellipsis in a code example indicates that what is shown is only a partial example.
% or $	The percent sign or dollar sign indicates the UNIX shell prompt, depending on the shell that is used.
#	The number (or pound) sign indicates the UNIX root prompt.

In addition, TimesTen documentation uses the following special conventions.

Convention	Meaning
`installation_dir`	The path that represents the directory where TimesTen is installed.
`timesten_home`	The path that represents the home directory of a TimesTen instance.
`release` or *`rr`*	The first two parts in a release number, with or without the dot. The first two parts of a release number represent a major TimesTen release. For example, 181 or 18.1 represents TimesTen Release 18.1.
`DSN`	TimesTen data source name (for the TimesTen database).

Note: TimesTen release numbers are reflected in items such as TimesTen utility output, file names, and directory names, all of which are subject to change with every minor or patch release. The documentation cannot always be up to date. It seeks primarily to show the basic form of output, file names, directory names, and other code that may include release numbers. You can confirm the current release number by looking at *Oracle TimesTen In-Memory Database Release Notes* or executing the `ttVersion` utility.

Documentation Accessibility

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

Access to Oracle Support

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

What's New

This section summarizes new features and functionality of TimesTen Release 18.1 that are documented in this guide, providing links into the guide for more information.

New features in Release 18.1.4.4.0

- Enhanced features for password management are fully supported and documented. See "Password management" on page 1-5.
- For use with TLS (Transport Layer Security), new options for the `ttCreateCerts` utility allow you to specify the elliptical curve signing algorithm, size of the elliptical curve, and duration for which the certificates are valid. See "Features of the ttCreateCerts utility" on page 3-2.

New features in Release 18.1.4.1.0

- Previous content in this document has been reorganized into the two major topics of authentication and authorization, each with its own chapter.
- In this release, TimesTen formally supports the use of TLS (Transport Layer Security) to encrypt communication between TimesTen instances. See Chapter 3, "Encryption in TimesTen".
- In this release, TimesTen provides a Kubernetes Operator. Security considerations are discussed in Chapter 4, "Security for the TimesTen Kubernetes Operator". For general information, see *Oracle TimesTen In-Memory Database Kubernetes Operator User's Guide*.

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, covering these topics:

- Overview of TimesTen users
- Managing TimesTen users
- Cache group users
- Password management

> **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. For additional information, 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:** For information about the `ADMIN` privilege, see "Administrative privileges" on page 2-5.

- 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 be used for TimesTen Client/Server unless the client and server are on the same host.

Notes:

- For additional information, 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

This section discusses TimesTen features for managing database users, covering the following:

- Creating or identifying a database user
- Changing the password of an internal user
- Specifying a Client/Server user and password
- Specifying a user and password for TimesTen utilities
- 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 only through a TimesTen direct connection. (See "CREATE USER" and "ALTER USER" in *Oracle TimesTen In-Memory Database SQL Reference* for details about these statements.)

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, 2020-02-25 12:00:17.027100, <NULL>,
<NULL> >
1 row found.
```

For details on these views, 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.
```

Specifying a Client/Server user and password

Once you have defined the user name and password for a TimesTen Client/Server connection, through the `UID` and `PWD` connection attributes, you can specify them in the following ways in order to connect to the database:

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

For additional information, see "UID and PWD" in *Oracle TimesTen In-Memory Database Reference*.

Note: Client/Server supports setting or changing a password (through `CREATE USER`, `ALTER USER`, or the `ttCacheUidPwdSet` built-in procedure) only for connections using TLS.

Specifying a user and password for TimesTen utilities

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

For additional information, see "UID and PWD" in *Oracle TimesTen In-Memory Database Reference*.

Notes:

- 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 information about this statement.

For example:

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

Notes:

- 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

To use TimesTen Cache, you must create the following users on the Oracle Database:

- A 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.
- One or more schema users who own the Oracle Database tables to be cached in a TimesTen database. These may be existing users or new users.

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

- A cache manager user who performs cache group operations. The TimesTen cache manager user must have the same user name as one of the Oracle Database users created for cache who can access the cached Oracle Database tables. This Oracle Database user, identified as the "companion" Oracle Database user, can be the cache administration user, a schema user, or some other existing user. For ease of use, it is preferred to have the Oracle Database cache administration user be the companion user for the TimesTen cache manager user. The password of the cache manager user can be different from the password of the companion Oracle Database 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 or will own 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.

One of the prerequisites to setting up your cache environment is informing the TimesTen database of the cache administration user name and password in the Oracle database.

1. Start the `ttIsql` utility and connect to the `cache1` DSN (for example) as the cache manager user, and set:

   - The `UID` connection attribute to specify the cache manager user name.
   - The `PWD` connection attribute to specify the cache manager user password.
   - The `OraclePWD` connection attribute to specify the password of the companion Oracle database user.

   This example uses `ttIsql` to connect, where `cacheuser` is the cache manager user with password `timesten`. In this example, the cache administration user, whose password is `oracle`, is the companion user to the cache manager user, so that password is provided.

   ```
   % ttIsql "DSN=cache1;UID=cacheuser;PWD=timesten;OraclePWD=oracle"
   ```

2. Use the `ttCacheUidPwdSet` built-in procedure (only once) to inform the TimesTen database of the cache administration user name and password in the Oracle database.

   The cache administration user name is `cacheuser` and its password is `oracle`.

   ```
   Command> call ttCacheUidPwdSet('cacheuser','oracle');
   ```

For full details on creating users for TimesTen Cache, see "Setting up the Oracle Database and TimesTen Classic systems" and "Setting Up a Caching Infrastructure" in *Oracle TimesTen Application-Tier Database Cache User's Guide*.

For information about required privileges for the cache administration user, cache manager user, and cache users, see "Privileges for cache groups" on page 2-24.

Password management

This section provides an overview of password management features and mechanisms in TimesTen that increase the level of security that can be implemented for authentication, covering these topics:

- Password management features
- Profile for password management

Password management features

This section provides an overview of password management features to 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 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.

Profile for password management

TimesTen employs profiles to specify settings of the password management parameters 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" or "ALTER PROFILE" in *Oracle TimesTen In-Memory Database SQL Reference* for details about the password management parameters. See "CREATE USER" or "ALTER USER" for information about specifying a profile when creating or altering a user.

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. 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. 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.

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:

- Granting and revoking privileges
- Functionality of privileges

- Overview of system privileges
- Overview of object privileges
- Privileges for TimesTen utilities
- Overview of the PUBLIC role

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* for syntax and usage details.

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" on page 2-3.
- Object privileges enable access to a specific database object, such as a particular table or view. See "Overview of object privileges" on page 2-3

A subset of these privileges are automatically granted to each user upon creation through the `PUBLIC` role. See "Overview of the PUBLIC role" on page 2-4.

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`. For details on granting or revoking system privileges, see "System privileges" on page 2-4.

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" on page 2-5. The instance administrator has an all-encompassing set of system privileges, as covered in "Instance administrator privileges" on page 2-4.

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.

For details on granting or revoking object privileges, see "Privileges for SQL objects" on page 2-10.

Privileges for TimesTen utilities

Any special privilege required to run a TimesTen utility is noted under "Required privilege" in the description of the utility in "Utilities" or "TimesTen Scaleout 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" on page 2-8 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

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. These system views are described in "User privilege views" on page 2-26.

The following sections describe system privileges available in TimesTen:

- Instance administrator privileges
- Administrative privileges
- Privileges to connect to the database
- ANY keyword
- ALL PRIVILEGES
- Privilege hierarchy
- Additional system privileges
- Privileges through the PUBLIC role

Instance administrator privileges

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.

Only the instance administrator can:

- 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.

In addition, for TimesTen Scaleout, only the instance administrator can execute any commands of the `ttGridAdmin` utility. Among many other functions, including those listed above, only the TimesTen Scaleout instance administrator can create a grid, create database definitions and connectables, change the distribution map of an existing database, create repositories, and perform backups, restores, exports, and imports.

See the following for related information:

- "Instance administrator", "Understanding the TimesTen users group", and "TimesTen instances" in *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*
- "TimesTen Scaleout architecture" and "The operating system user" in *Oracle TimesTen In-Memory Database Scaleout User's Guide*

Notes:

- The instance administrator cannot be the root user.
- You cannot change to a different instance administrator.
- In TimesTen Scaleout, the instance administrator's user name, user ID, group name, and group ID must all be the same on all hosts of the grid.

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" on page 2-26.

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

```
Command> GRANT ADMIN TO terry;
```

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

```
Command> 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. For a complete description of first connection attributes, including their required privileges, see "Connection Attributes" in *Oracle TimesTen In-Memory Database Reference*.

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 legal, 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 legal 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 illegal 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* for additional information.

Additional system privileges

In addition to the ADMIN privilege, the following 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" on page 2-24 for details.

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.

Notes:

- 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. (For more information on this view, see "User privilege views" on page 2-26.) 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

This section provides an overview of privileges required to create, alter, or 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;
```

Notes:

- See "Object privileges for views" on page 2-12 and "Object privileges for materialized views" on page 2-13 for additional considerations in creating views and 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" on page 2-14.

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);
```

She 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.

Notes: 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, he 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 revalidate 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 and some DDL and TimesTen 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

This section covers the following topics for authorization in PL/SQL:

- 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" on page 2-10.

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.

This is all 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

The following statements 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;
```

Notes:

- 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.

Example 2–1 Granting of required privileges

This 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 his or her 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.
```

Instance administrator:

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

user1:

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

```
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';
```

Example 2–2 Invalidated object

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, 2019-06-04 14:51:34, 2019-06-04 14:58:23,
2019-06-04:14:58:23, 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, 2019-06-04 14:51:34, 2019-06-04 16:13:00,
2019-06-04:16:13:00, 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 his or her 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* for additional information.

Example 2–3 AUTHID clause for definer's or invoker's rights

This 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 his own (the invoker's) `myInventory` table.

```
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

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 cache administration user (an Oracle Database user) and the cache manager user (a TimesTen user), who require special privileges.

These sections discuss cache group users and privileges:

- Cache administration user privilege
- Cache manager user privilege
- Cache user privilege

See "Cache group users" on page 1-4 for an introduction to these 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.

Cache administration user privilege

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

See "Grant privileges to the Oracle database users" in *Oracle TimesTen Application-Tier Database Cache User's Guide* for details.

Cache manager user privilege

The required privilege for the TimesTen cache manager user is the `CACHE_MANAGER` system privilege, enabling the user to perform necessary cache group operations. A 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 the cache administration user and the cache manager user" in *Oracle TimesTen Application-Tier Database Cache User's 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.

Cache user privilege

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.

For related information, see "Create the TimesTen users" and "Grant privileges to the TimesTen users" in *Oracle TimesTen Application-Tier Database Cache User's Guide*.

Individual cache group privileges are discussed in these sections:

- Cache group system privileges
- Cache group object privileges

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;
```

For details on cache group operations, see "Cache Group Operations" in *Oracle TimesTen Application-Tier Database Cache User's Guide*.

User privilege views

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

Table 2–3 System privilege views

View name	Description
SYS.USER_SYS_PRIVS	Returns all of the system privileges granted to the current user.
SYS.DBA_SYS_PRIVS	Returns the list of system privileges granted to all users and inherited from the PUBLIC role. 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: For details on these views, see "System Tables and Views" in *Oracle TimesTen In-Memory Database System Tables and Views Reference*.

3

Encryption in TimesTen

This chapter discusses the use of encryption in TimesTen.

Some features in TimesTen, such as Client/Server and replication, support use of Transport Layer Security (TLS) for communication between TimesTen instances.

These topics are discussed here:

- Certificates for Transport Layer Security
- Transport Layer Security for TimesTen Client/Server
- Transport Layer Security for TimesTen replication

Notes:

- In TimesTen Release 18.1, TLS is not supported in TimesTen Scaleout.
- TimesTen uses TLS version 1.2.
- TLS configuration in TimesTen does not apply to communication between TimesTen and Oracle Database, such as when TimesTen Cache is used. For encrypted TimesTen-Oracle communication, configure TLS through settings in the `sqlnet.ora` file used for connections to Oracle Database. Refer to "Configuring Secure Sockets Layer Authentication" in *Oracle Database Security Guide*.

Certificates for Transport Layer Security

This section describes the use of certificates for TLS for TimesTen, covering these topics:

- About using certificates with TimesTen
- Generation of certificates for TimesTen

About using certificates with TimesTen

TimesTen Client/Server and replication support use of Transport Layer Security for communication between TimesTen instances. TimesTen provides the `ttCreateCerts` utility to generate certificates for TLS for TimesTen.

The procedure shown in "Using ttCreateCerts" on page 3-3 results in the TimesTen server having its own self-signed root certificate; however, this does not preclude using a certificate signed by a third party commercial certificate authority. TimesTen

supports both self-signed and CA-signed certificates. Certificates produced by `ttCreateCerts` are self-signed. The wallets are platform-independent.

The wallet of each TimesTen server must have its own public/private key identity signed by the root certificate. Each client that connects to a TimesTen server must have a wallet containing the root certificate of that server. (A client may optionally have multiple wallets for connections to multiple database services.) By default, TimesTen uses TLS server authentication, so that the client can verify that the server it is connecting to is valid.

The server uses the existing TimesTen user ID and password mechanism to authenticate a user, but TimesTen also supports a form of client authentication where the server validates an identity in the client wallet. This is a way for the server to verify that the connecting client is a legitimate TimesTen client, but the user still must provide user ID and password credentials.

The wallet is an auto-login or single-sign-on (SSO) wallet, without a password. Access to the wallet is controlled by file system permissions.

TimesTen supports NSA Suite B cipher suites `SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256` and `SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384`. See "Configuring Secure Sockets Layer Authentication" in *Oracle Database Security Guide* for information about these cipher suites.

Generation of certificates for TimesTen

The following sections describe the TimesTen `ttCreateCerts` utility and show how to use it to create certificates. The certificates can be used with TLS for either TimesTen Client/Server or TimesTen replication.

- Features of the ttCreateCerts utility
- Using ttCreateCerts

Features of the ttCreateCerts utility

The `ttCreateCerts` utility is located in the `bin` directory of a TimesTen instance. You must set `TIMESTEN_HOME` before you run `ttCreateCerts`, which you can accomplish by sourcing the `ttenv.sh` or `ttenv.csh` script from the `bin` directory.

The utility creates three Oracle Wallets: `rootWallet`, `clientWallet`, and `serverWallet`.

> **Note:** You must have Java on your system to use `ttCreateCerts`. The utility searches for it according to the `JRE_HOME`, `JAVA_HOME`, and `PATH` settings.

This is the syntax for `ttCreateCerts`:

```
% ttCreateCerts -h
usage: ttCreateCerts [-dir WALLETDIR] [options...]
       ttCreateCerts [-h | -help]
       ttCreateCerts [-V | -version]
options:
  -f | -force
  -verbose
  -validity DAYS | -valid_from mm/dd/yyyy -valid_until mm/dd/yyyy
  -dryrun
  -sign_alg ALGORITHM (ecdsasha256 ecdsasha384 ecdsasha512)
  -eccurve TYPE  (p256 p384 p521)
```

Input and option descriptions:

- `-dir`: Specifies a directory where the wallets are placed, as an absolute path. The specified directory must already exist and cannot already contain wallets produced by `ttCreateCerts`, unless you use the `-force` option. The default is *timesten_home*`/conf`.
- `-h` or `-help`: Shows help (showing the above syntax).
- `-V` or `-version`: Displays the TimesTen release number.
- `-f` or `-force`: Overwrites any previous wallets in the specified directory.
- `-verbose`: Shows additional output from execution of the utility.
- `-validity`: One of two ways to specify the lifetime of the wallets that are created, expressed as a number of days from creation. The default is 3650 days, which can be overridden by setting either `-validity` or `-valid_from` and `-valid_until`.
- `-valid_from` and `-valid_until`: The other way to specify the lifetime of the wallets that are created, expressed as a start and an end date in *mm/dd/yyyy* format.
- `-dryrun`: Echoes all the commands to be executed by `ttCreateCerts` to create the certificates as you specified, but without executing them. For options you do not set, you can use this to confirm what the default values are.
- `-sign_alg`: Specifies the elliptical curve signing algorithm. Supported algorithms are `ecdsasha256`, `ecdsasha384` (default), and `ecdsasha512`.
- `-eccurve`: Specifies the size of the elliptical curve. Supported values are `p256`, `p384` (default), and `p521`.

Using ttCreateCerts

This section provides `ttCreateCerts` examples that place the wallets in a `wallets` subdirectory under *timesten_home*`/conf`, where *timesten_home* is the full path to the TimesTen instance home directory.

The following example includes verbose output. (Without the `-verbose` option, only the last line is shown.)

```
% ttCreateCerts -verbose -dir timesten_home/conf/wallets
Requested Certificates:
User Certificates:
Subject:        CN=server1,C=US
Trusted Certificates:
Subject:        CN=ecRoot,C=US
Requested Certificates:
User Certificates:
Subject:        CN=client1,C=US
Trusted Certificates:
Subject:        CN=ecRoot,C=US
ttCreateCerts : certificates created in timesten_home/conf/wallets
```

Here are the results:

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

```
cwallet.sso
```

> **Note:** Any `.lck` files can be ignored.

See "Certificates for TimesTen Client/Server" on page 3-5 and "Certificates for TimesTen replication" on page 3-12 for where to copy certificates for TimesTen Client/Server and replication.

The next example is a dry run. No certificates are created (despite the last line). This shows only a snippet of the output:

```
% ttCreateCerts -dir timesten_home/conf/wallets -dryrun
...
+ /bin/java -Djava.security.egd=file:///dev/./urandom -Xms64m -Xmx512m -cp
/scratch/classic181430/instances/tt181/install/lib/cryptoj_5_0.jar:
/scratch/classic181430/instances/tt181/install/lib/oraclepki.jar:
/scratch/classic181430/instances/tt181/install/lib/osdt_cert.jar:
/scratch/classic181430/instances/tt181/install/lib/osdt_core.jar
oracle.security.pki.textui.OraclePKITextUI wallet add -wallet rootWallet -dn
CN=ecRoot,C=US -sign_alg ecdsasha384 -self_signed -asym_alg ECC -eccurve p384
-jsafe -validity 3650 -auto_login_only -nologo
...
ttCreateCerts : certificates created in timesten_home/conf/wallets
```

From this, you can see that the default settings are `-sign_alg ecdsasha384`, `-eccurve p384`, and `-validity 3650` (days).

Here is another example that sets signing algorithm and size of the elliptical curve:

```
% ttCreateCerts -dir timesten_home/conf/wallets -sign_alg ecdsasha256
-eccurve p256
ttCreateCerts : certificates created in timesten_home/conf/wallets
```

This example specifies that the certificates will expire one year from when they were created:

```
% ttCreateCerts -dir timesten_home/conf/wallets -validity 365
ttCreateCerts : certificates created in timesten_home/conf/wallets
```

Or, equivalently:

```
% ttCreateCerts -dir timesten_home/conf/wallets -valid_from 10/28/2020
-valid_until 10/28/2021
ttCreateCerts : certificates created in timesten_home/conf/wallets
```

The next example tries to create wallets in a location where wallets already exist:

```
% ttCreateCerts -dir timesten_home/conf/wallets
ttCreateCerts: rootWallet is not empty, use -force to overwrite
```

This example tries again, using the `-force` option:

```
% ttCreateCerts -dir timesten_home/conf/wallets -f
ttCreateCerts : certificates created in timesten_home/conf/wallets
```

Transport Layer Security for TimesTen Client/Server

When you use TimesTen Client/Server, you can optionally configure and use TLS for encrypted communication between the TimesTen client (or clients) and TimesTen server. This feature is supported with TimesTen release 18.1.2.1.0 and higher.

This section discusses TimesTen support for TLS for Client/Server, covering these topics:

- Introduction to TLS for Client/Server
- Certificates for TimesTen Client/Server
- Configuration for TLS for Client/Server
- Operation of TLS for Client/Server

Note: Using a server or client prior to TimesTen release 18.1.2.1.0 results in the connection being unencrypted, unless encryption is specified as required at the other end, in which case the connection will fail.

Introduction to TLS for Client/Server

This feature enables secure TCP/IP network connections, using TLS, between TimesTen clients and servers. The certificates for a TLS session are contained in an Oracle Wallet. Without enabling this feature, and in earlier TimesTen releases, network communication is unencrypted.

These are the high-level steps to use TLS for Client/Server:

1. Create certificates on the TimesTen server, placing them in a desired wallet directory.
2. Copy the wallet directory to the TimesTen clients.
3. Configure TLS on the server and on the clients, as appropriate, including specifying the wallet directory and cipher suite.
4. Confirm operation of TLS communication. (TLS is used as soon as the configuration is complete.)

Certificates for TimesTen Client/Server

This section discusses where to copy the certificates you generated for TimesTen Client/Server. Recall the resulting wallets from the example in "Using ttCreateCerts" on page 3-3:

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

For use of TLS for Client/Server, ignore `client1.cert`, `root.cert`, `rootWallet`, `server1.cert`, and `serverWallet`. Copy the `clientWallet` directory, which includes the root certificate, to the desired location. This is preferably the same location on each TimesTen client instance.

On each client:

```
% mkdir timesten_home/conf/wallets
[...Copy clientWallet from the server...]
% cd timesten_home/conf/wallets
% ls
```

```
clientWallet
% ls clientWallet
cwallet.sso
```

Note: Be careful to preserve the file and directory permissions of the wallet.

Configuration for TLS for Client/Server

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

- Configuration on the server
- Configuration on the client

Configuration on the server

These server-side attributes for TLS for Client/Server can be set in the server DSN definition in *timesten_home*/conf/sys.odbc.ini, where *timesten_home* is the TimesTen instance directory.

- Wallet: Specify the wallet location, as an absolute path, where you placed the certificates that you generated (preferably the same directory path as on the client). There is no default location. If you used ttCreateCerts, this is the full path of the serverWallet directory.
- Encryption: Use one of these 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 3–1 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 SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256, SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384, or both, comma-separated and in order of preference. There is no default setting. For TLS to be used, the server and client settings must include at least one common 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 attributes in `sys.odbc.ini`, you can set these equivalent attributes in the TimesTen configuration file, *timesten_home*`/conf/timesten.conf`:

- `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 also 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.

Table 3–1 shows the results of all possible combinations of encryption settings between client and server, with consideration of the cipher suite settings.

Table 3–1 Results of combinations of server and client encryption settings

Server encryption setting	Client encryption 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.
`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.

Table 3–1 (Cont.) Results of combinations of server and client encryption settings

Server encryption setting	Client encryption setting	Result
`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.

Configuration on the client

These client-side attributes for TLS for Client/Server can be set in the client DSN definition in *timesten_home*`/conf/sys.odbc.ini` or a client `odbc.ini` file, or in a connect string.

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 you placed the certificates that you generated (preferably the same directory path as on the server). There is no default location. If you used `ttCreateCerts`, this is the full path of the `clientWallet` directory.
- `Encryption`: Use one of these 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 3–1 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 `SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`, `SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384`, or both, comma-separated and in order of preference. There is no default setting. For TLS to be used, the server and client settings must include at least one common 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 some of the preceding attributes in `sys.odbc.ini` or client `odbc.ini`, you can set these equivalent attributes in the TimesTen configuration file, *`timesten_home`*`/conf/timesten.conf`:

  - `client_wallet`
  - `client_cipher_suites`

  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.

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`, as described in Table 3–1, 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.

Example 3–1 shows the results of several combinations of encryption settings on the server and client.

Example 3–1 Connecting with TLS

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:

```
[sampledb]
Driver=timesten_home/install/lib/libtten.so
DataStore=/db/databases/sampledb
PermSize=512
TempSize=128
LogBufMB=256
LogFileSize=256
```

```
LogDir=/db/logs
DatabaseCharacterSet=AL32UTF8
OracleNetServiceName=ttorcl
Wallet=timesten_home/conf/mywallets/serverWallet
Encryption=requested
CipherSuites=SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
```

Client DSN definition:

```
[sampledbCS]
TTC_SERVER=myserverhost.example.com
TTC_SERVER_DSN=sampledb
UID=myuser
PWD=welcome
Wallet=timesten_home/conf/mywallets/clientWallet
Encryption=requested
CipherSuites=SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
```

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

```
% ttIsqlCS sampledbCS

Copyright (c) 1996, 2020, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

connect "DSN=sampledbCS";
Connection successful: DSN=sampledbCS;TTC_SERVER=myserverhost.example.com;
TTC_SERVER_DSN=sampledb;UID=myuser;DATASTORE=/db/databases/sampledb;
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=SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384;
(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=SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
```

From the client DSN definition:

```
CipherSuites=SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
```

Connect:

```
% ttIsqlCS sampledbCS

Copyright (c) 1996, 2020, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.
```

```
connect "DSN=sampledbCS";

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. This is not sufficient to result in TLS usage, as noted in Table 3–1. The connection is successful but TLS is not used. (Except for what is shown here, settings are the same as in Scenario 1.)

From the server DSN definition:

```
Encryption=accepted
```

From the client DSN definition:

```
Encryption=accepted
```

Connect:

```
% ttIsqlCS sampledbCS

Copyright (c) 1996, 2020, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

connect "DSN=sampledbCS";
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 sampledbCS

Copyright (c) 1996, 2020, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

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

Transport Layer Security for TimesTen replication

When you use TimesTen replication, you can optionally configure and use Transport Layer Security (TLS) for secure network communication to and from TimesTen replication agents.

This section discusses TimesTen support for TLS for replication, covering these topics:

- Introduction to TLS for replication
- Certificates for TimesTen replication
- Configuration for TLS for replication
- Activation of TLS for replication
- Operation of TLS for replication

Introduction to TLS for replication

This feature enables secure TCP/IP network connections, using TLS, for TimesTen replication agents. Mutual authentication is used for all connections. This applies to communications between TimesTen instances, either between replication agents or between TimesTen utilities (such as `ttRepAdmin`) and replication agents. Without enabling this feature, and in earlier TimesTen releases, such communications are unencrypted.

These are the high-level steps to use TLS for replication:

1. Create certificates on one of the TimesTen instances, placing them in a desired wallet directory.
2. Copy the contents of the wallet directory to the other TimesTen instances.
3. Configure TLS in TimesTen, specifying the wallet directory and cipher suite.
4. Enable TLS communications (restart the replication agents to begin using TLS).

Certificates for TimesTen replication

This section discusses where to copy the certificates you generated for TimesTen replication. Recall the resulting wallets from the example in "Generation of certificates for TimesTen" on page 3-2:

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

For use of TLS for replication, ignore `client1.cert`, `clientWallet`, `root.cert`, `rootWallet`, and `server1.cert`. 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
```

Configuration for TLS for replication

To use TLS for replication, set these 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:

> **Important:** Generate and place certificates first. If you configure TLS for replication before you generate and place the certificates, and replication agents restart in between, there will be an error condition when replication agents try to access non-existent certificates.

- `replication_cipher_suite`: The cipher suite to be used in encrypting communications to and from the replication agent. This setting is required. There is no default.

  TimesTen supports NSA Suite B cipher suites `SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256` and `SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384`. These suites require elliptical signing.

- `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.

- `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 this instance and the replication peer, but TLS is not used for communications between the replication agents. Use this setting to ease the transition to TLS, if secure communication is not critical (at least initially).

  – 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.

Notes:

- 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.

Activation of 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 result in TLS being used for communication to and from the replication agents.

There are two ways to accomplish this:

- Enable TLS for replication online. Restart the replication agents one at a time as replication continues to function.

Or:

- Enable TLS for replication simultaneously on all instances, stopping replication until you restart all the replication agents at once.

Note: This assumes consistent settings between instances or is subject to `replication_ssl_mandatory` settings, as discussed in "Configuration for TLS for replication" on page 3-13.

This section covers these topics:

- Switching online to TLS for replication
- Switching all instances simultaneously to TLS for replication (offline)

Switching online to TLS for replication

If you have an existing replication scheme that is not using TLS, use these steps for an online switchover to TLS as replication continues to function:

1. Generate certificates as discussed in "Generation of certificates for TimesTen" on page 3-2, then copy them to the desired location on each TimesTen instance as discussed in "Certificates for TimesTen replication" on page 3-12.
2. On each instance, set `replication_wallet` to indicate where the certificates are located.
3. On each instance, set `replication_cipher_suite` to indicate the cipher suite you are using.
4. On each instance, set `replication_ssl_mandatory=0`.

   This allows you to update the TimesTen instances to start using TLS one at a time.

5. On each instance (one at a time, in succession), stop and restart the replication agent:

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

For example assume there is an active standby pair with databases `rep1` on `host1` and `rep2` on `host2`, with subscriber `rep3` on `host3`. Complete these steps, as replication continues to function, to use TLS for communications to and from each of the replication agents:

1. Create the certificates on `rep1` and place them in the desired wallet directory: `/swdir/mywalletloc` for example.
2. Copy the wallet directory to the appropriate location (preferably the same location) on `rep2` and `rep3`.
3. Configure TLS on all three instances. For example, in each `timesten.conf` file:

```
replication_wallet=/swdir/mywalletloc
replication_cipher_suite=SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
replication_ssl_mandatory=0
```

4. 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
```

Switching 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. Generate certificates as discussed in "Generation of certificates for TimesTen" on page 3-2, then copy them to the desired location on each TimesTen instance as discussed in "Certificates for TimesTen replication" on page 3-12.
2. 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
> ```

3. On all instances, set `replication_wallet` to indicate where the certificates are located.

4. On all instances, set `replication_cipher_suite` to indicate the cipher suite you are using.
5. On all instances, set `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.
6. 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
> ```

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* for information about this utility.

> **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.

4

Security for the TimesTen Kubernetes Operator

This chapter discusses security features and requirements for the TimesTen Kubernetes Operator, covering these topics:

- Privileges for the TimesTen Kubernetes Operator
- Authorization for users of the TimesTen Kubernetes Operator
- Encryption for the TimesTen Kubernetes Operator

Kubernetes is an open-source platform for managing containerized workloads and services. Kubernetes manages the resources of multiple hosts in a cluster and runs containers as required across these hosts.

TimesTen provides a Kubernetes Operator that manages Kubernetes objects of type `TimesTenClassic`. TimesTen can be deployed, monitored, managed, and controlled in an automated manner with no required human intervention.

See *Oracle TimesTen In-Memory Database Kubernetes Operator User's Guide* for information about the TimesTen Kubernetes Operator.

Privileges for the TimesTen Kubernetes Operator

The TimesTen Operator creates and manages Pods and containers running TimesTen on behalf of the user. It monitors and controls TimesTen in those containers through the TimesTen agent.

The Operator requires privileges in the Kubernetes cluster. The `service_account.yaml` file is a file included with the TimesTen Operator. When you run the `service_account.yaml` file, a Kubernetes role with a particular set of permissions is defined. The Operator runs with the permissions you granted to this role. See "Configuring Kubernetes" in *Oracle TimesTen In-Memory Database Kubernetes Operator User's Guide* for information on the `service_account.yaml` file.

The TimesTen Operator must be given suitable permissions to create and monitor the other objects in the Kubernetes cluster. The `service_account.yaml` file contains the default permissions.

Authorization for users of the TimesTen Kubernetes Operator

The set of Kubernetes users who can create, modify, and delete `TimesTenClassic` objects in a Kubernetes cluster is under the control of the Role Based Access Control (RBAC) configuration of the cluster.

In order to provide a secure installation, you should restrict the set of users who have Kubernetes RBAC permissions to GET Secret objects in the Kubernetes namespace.

(See the next section, "Encryption for the TimesTen Kubernetes Operator", regarding Secrets.)

The TimesTen agent creates the TimesTen instance, runs as the `oracle` user, and starts TimesTen. The `oracle` user is the instance administrator of the TimesTen instance. The Operator limits the set of open ports in containers that are running TimesTen to those ports that TimesTen uses.

Encryption for the TimesTen Kubernetes Operator

To ensure that only the TimesTen Operator can communicate with the TimesTen agents:

- Communication between the TimesTen Operator and the TimesTen agents is secured through TLS using self-signed certificates that are created by the Operator. These certificates, inside an Oracle Wallet, are transmitted to the agents through Kubernetes Secrets that the Operator creates. The TimesTen Operator runs in a customer-specified Kubernetes namespace. These Secrets are created in that namespace.
- Containers that run the TimesTen agent (and TimesTen itself) have access to the Secrets, and therefore to the certificates included in them. This insures that only the Operator and the agents have access to these certificates, preventing other users from using the agent to control TimesTen.
- The Operator creates a different self-signed certificate for each `TimesTenClassic` object when the object is created. These certificates are created by the standard Java `keytool` command and are stored in a `pkcs12` keystore.
- The Operator stores each keystore in a different Kubernetes Secret. When the Operator instructs Kubernetes to create Pods and containers (that run the TimesTen agents), the contents of the Secret are mounted as files in the file system of the TimesTen agent. This ensures that the certificate is securely communicated between the Operator and the TimesTen agents.
- The TimesTen agent is configured to accept only HTTPS connections and to authenticate those connections using the self-signed certificate. The agent is configured to listen on port `8443` and to not accept any other form of communication.

Index

A

access control
  authorization, 2-1
  encryption, 3-1
  user authentication, 1-1
administrative privileges, 2-5
ALL keyword, 2-14
ALL PRIVILEGES, 2-6
ALL_TAB_PRIVS view, 2-26
ALL_USERS view, 1-3
ALTER (ANY) PROCEDURE, 2-14
ALTER ANY privilege, 2-6
alter objects--privileges, 2-10
ANY keyword, 2-6
authentication
  cache users, 1-4
  changing user password, 1-3
  Client/Server users, 1-3
  creating users, 1-2
  dropping users, 1-4
  managing users, 1-2
  users overview, 1-1
  utilities users, 1-4
AUTHID clause, 2-20
authorization--see privileges

C

cache group privileges
  AWT cache groups, 2-25
  cache administration user, 2-24
  cache manager user, 2-24
  cache user, 2-25
  object privileges, 2-25
  overview, 2-24
  system privileges, 2-25
cache users, 1-4
  Oracle cache admin user, 1-4
  Oracle schema users, 1-4
  Oracle timesten user, 1-4
  TimesTen cache manager, 1-4
  TimesTen cache table users, 1-4
certificates
  Client/Server, 3-5
  replication, 3-12
  TLS, 3-1
changing user password, 1-3
Client/Server TLS
  certificates, 3-5
  configuration, 3-6
  example, 3-9
  introduction, 3-5
Client/Server user, password, 1-3
connection privileges, 2-6
CREATE (ANY) PROCEDURE, 2-14
CREATE ANY privilege, 2-6
create objects--privileges, 2-9
CREATE SESSION privilege, 2-6
creating users, 1-2

D

database directory privileges, 2-5
DBA_SYS_PRIVS view, 2-26
DBA_TAB_PRIVS view, 2-26
DBA_USERS view, 1-3
definer's rights, 2-20
DELETE ANY privilege, 2-6
DELETE privilege, 2-11
DROP (ANY) PROCEDURE, 2-14
DROP ANY privilege, 2-6
drop objects--privileges, 2-10
dropping users, 1-4

E

encryption
  Kubernetes, 4-2
  TLS (also see TLS), 3-1
EXECUTE (ANY) PROCEDURE, 2-14
EXECUTE ANY privilege, 2-6

F

FLUSH ANY CACHE GROUP privilege, 2-25
FLUSH privilege, 2-25
foreign key constraints, 2-12

G

GRANT ALL, 2-14

granting privileges, 2-2

I

IDENTIFIED BY clause, password, 1-2
INDEX privilege, 2-11
INSERT ANY privilege, 2-6
INSERT privilege, 2-11
instance administrator privileges, 2-4
invalidated PL/SQL objects, 2-18
invoker's rights, 2-20

K

Kubernetes
  authorization, 4-1
  encryption, 4-2
  privileges, 4-1

L

LOAD ANY CACHE GROUP privilege, 2-25
LOAD privilege, 2-26
log directory privileges, 2-5

M

managing users, 1-2
materialized view privileges, 2-13

O

object privileges
  ALL keyword, 2-14
  cache groups, 2-25
  materialized views, 2-13
  overview, 2-3
  PL/SQL statements, 2-14
  sequences, 2-13
  synonyms, 2-13
  tables, 2-11
  views, 2-12

P

parent-child tables, 2-12
passwords
  for cache manager user, 1-5
  for Client/Server user, 1-3
  for internal user, 1-3
  for utilities users, 1-4
  password management features, 1-5
  profile, 1-6
  setting with IDENTIFIED BY, 1-2
permissions--see privileges
PL/SQL
  AUTHID clause, 2-20
  definer's rights, 2-20
  invalidated objects, 2-18
  invoker's rights, 2-20
  objects, privileges, 2-14
  statements, 2-14
privileges
  administrative privileges, 2-5
  ALL keyword, 2-14
  ALL PRIVILEGES, 2-6
  alter objects, 2-10
  ANY keyword, 2-6
  cache administration user, 2-24
  cache groups, 2-24
  cache manager user, 2-24
  cache user, 2-25
  connect to database, 2-6
  create objects, 2-9
  database directory, 2-5
  drop objects, 2-10
  foreign key constraints, 2-12
  functionality, 2-2
  granting, 2-2
  hierarchy, 2-7
  instance administrator, 2-4
  Kubernetes, 4-1
  levels, 2-3
  log directory, 2-5
  materialized views, 2-13
  object privileges, 2-3
  overview, 2-1
  parent-child tables, 2-12
  PL/SQL objects, 2-14
  PL/SQL statements, 2-14
  PUBLIC ROLE, 2-8
  revoking, 2-2
  sequences, 2-13
  SQL objects, 2-10
  synonyms, 2-13
  system privileges, 2-3, 2-4
  tables, 2-11
  user privilege views, 2-26
  utilities, 2-3
  views, 2-12
  XLA, 2-7
profile for passwords, 1-6
PUBLIC role
  overview, 2-4
  privileges, 2-8

R

REFERENCES privilege, 2-11
REFRESH ANY CACHE GROUP privilege, 2-25
REFRESH privilege, 2-26
replication TLS
  activation, 3-14
  certificates, 3-12
  configuration, 3-13
  introduction, 3-12
  operation, 3-16
REVOKE ALL, 2-14
revoking privileges, 2-2

S

SELECT ANY privilege, 2-6
SELECT privilege, 2-11
sequences privileges, 2-13
SQL objects, privileges, 2-10
synonym privileges, 2-13
system privileges
  administrative privileges, 2-5
  ALL PRIVILEGES, 2-6
  ANY keyword, 2-6
  cache groups, 2-25
  connect to database, 2-6
  instance administrator, 2-4
  overview, 2-3
  PUBLIC ROLE, 2-8
  XLA, 2-7

T

table privileges, 2-11
TLS
  certificates, 3-1
  for Client/Server, 3-4
  for replication, 3-12
Transport Layer Security--see TLS

U

UNLOAD ANY CACHE GROUP privilege, 2-25
UNLOAD privilege, 2-26
UPDATE ANY privilege, 2-6
UPDATE privilege, 2-11
USER_SYS_PRIVS view, 2-26
USER_TAB_PRIVS view, 2-26
USER_USERS view, 1-3
users
  cache users, 1-4
  changing password, 1-3
  Client/Server, 1-3
  creating, 1-2
  dropping, 1-4
  managing, 1-2
  overview, 1-1
  user privilege views, 2-26
  utilities, 1-4
utilities
  privileges, 2-3
  user, password, 1-4

V

view privileges, 2-12

W

wallets--see certificates

X

XLA privilege, 2-7