Contents

Preface ................................................................................................................................................................ v
  Audience ...................................................................................................................................................... vii
  Related documents ................................................................................................................................ vii
  Conventions .............................................................................................................................................. viii
  Documentation Accessibility ..................................................................................................................... ix

What's New .................................................................................................................................................... xi
  New features in Release 11.2.2.0.0 ......................................................................................................... xi

1 Introduction to PL/SQL in the TimesTen Database
  Features of PL/SQL in TimesTen ............................................................................................................... 1-1
  TimesTen PL/SQL components and operations ......................................................................................... 1-2
    Application interaction with TimesTen and PL/SQL ............................................................................... 1-2
    PL/SQL in TimesTen versus PL/SQL in Oracle Database ...................................................................... 1-3
    SQL statements in PL/SQL blocks ........................................................................................................ 1-3
    Execution of PL/SQL from SQL ........................................................................................................... 1-4
  Audiences for this document .................................................................................................................... 1-4
    Developers experienced with Oracle Database and Oracle Database PL/SQL ..................................... 1-4
    Developers experienced with TimesTen .............................................................................................. 1-5
  About the TimesTen PL/SQL demos ........................................................................................................ 1-5

2 Programming Features in PL/SQL in TimesTen
  PL/SQL blocks ......................................................................................................................................... 2-1
  PL/SQL variables and constants ............................................................................................................... 2-2
  SQL function calls from PL/SQL ............................................................................................................. 2-5
  PL/SQL control structures ....................................................................................................................... 2-6
    Conditional control .............................................................................................................................. 2-6
    Iterative control .................................................................................................................................. 2-7
    CONTINUE statement .......................................................................................................................... 2-7
  How to execute PL/SQL procedures and functions ................................................................................... 2-8
  How to pass data between an application and PL/SQL .......................................................................... 2-9
    Using bind variables from an application ............................................................................................ 2-9
    IN, OUT, and IN OUT parameter modes ............................................................................................ 2-10
  Use of SQL in PL/SQL programs ............................................................................................................. 2-11
    Static SQL in PL/SQL for queries and DML statements ...................................................................... 2-11
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic SQL in PL/SQL (EXECUTE IMMEDIATE statement)</td>
<td>2-12</td>
</tr>
<tr>
<td>FORALL and BULK COLLECT operations</td>
<td>2-14</td>
</tr>
<tr>
<td>RETURNING INTO clause</td>
<td>2-15</td>
</tr>
<tr>
<td>TimesTen PL/SQL with TimesTen Cache</td>
<td>2-16</td>
</tr>
<tr>
<td>Large objects (LOBs)</td>
<td>2-17</td>
</tr>
<tr>
<td><strong>About LOBs</strong></td>
<td>2-18</td>
</tr>
<tr>
<td><strong>LOB locators</strong></td>
<td>2-18</td>
</tr>
<tr>
<td><strong>Temporary LOBs</strong></td>
<td>2-19</td>
</tr>
<tr>
<td>Differences between TimesTen LOBs and Oracle Database LOBs</td>
<td>2-19</td>
</tr>
<tr>
<td>Using LOBs</td>
<td>2-19</td>
</tr>
<tr>
<td>PL/SQL package support for LOBs</td>
<td>2-20</td>
</tr>
<tr>
<td>Passthrough LOBs</td>
<td>2-20</td>
</tr>
<tr>
<td>Use of cursors in PL/SQL programs</td>
<td>2-21</td>
</tr>
<tr>
<td>PL/SQL procedures and functions</td>
<td>2-21</td>
</tr>
<tr>
<td>Creating and using procedures and functions</td>
<td>2-21</td>
</tr>
<tr>
<td>Using synonyms for procedures and functions</td>
<td>2-24</td>
</tr>
<tr>
<td>PL/SQL packages</td>
<td>2-25</td>
</tr>
<tr>
<td>Package concepts</td>
<td>2-25</td>
</tr>
<tr>
<td>Creating and using packages</td>
<td>2-25</td>
</tr>
<tr>
<td>Using synonyms for packages</td>
<td>2-28</td>
</tr>
<tr>
<td>Wrapping PL/SQL source code</td>
<td>2-29</td>
</tr>
<tr>
<td>PL/SQL composite data types</td>
<td>3-1</td>
</tr>
<tr>
<td>Using collections in PL/SQL</td>
<td>3-2</td>
</tr>
<tr>
<td>Using records in PL/SQL</td>
<td>3-4</td>
</tr>
<tr>
<td>Using associative arrays from applications</td>
<td>3-4</td>
</tr>
<tr>
<td>PL/SQL composite data types</td>
<td>3-4</td>
</tr>
<tr>
<td>PL/SQL REF CURSORs</td>
<td>3-7</td>
</tr>
<tr>
<td>Data type conversion</td>
<td>3-9</td>
</tr>
<tr>
<td>Conversion between PL/SQL data types</td>
<td>3-9</td>
</tr>
<tr>
<td>Conversion between application data types and PL/SQL or SQL data types</td>
<td>3-9</td>
</tr>
<tr>
<td>Differences in TimesTen: data type considerations</td>
<td>3-10</td>
</tr>
<tr>
<td>Conversion between PL/SQL and TimesTen SQL data types</td>
<td>3-11</td>
</tr>
<tr>
<td>Date and timestamp formats: NLS_DATE_FORMAT and NLS_TIMESTAMP_FORMAT</td>
<td>3-12</td>
</tr>
<tr>
<td>Non-supported data types</td>
<td>3-13</td>
</tr>
<tr>
<td>Zero-length strings in TimesTen type mode</td>
<td>3-13</td>
</tr>
</tbody>
</table>
4 Errors and Exception Handling

Understanding exceptions ................................................................. 4-1
  About exceptions ........................................................................... 4-1
  Exception types ............................................................................ 4-2

Trapping exceptions ........................................................................ 4-2
  Trapping predefined TimesTen errors ......................................... 4-3
  Trapping user-defined exceptions ............................................... 4-4
    Using the RAISE statement ...................................................... 4-4
    Using the RAISE_APPLICATION_ERROR procedure .................. 4-5

Showing errors in ttIsql ................................................................... 4-6

Differences in TimesTen: exception handing and error behavior ........ 4-7
  TimesTen PL/SQL transaction and rollback behavior for unhandled exceptions 4-7
  TimesTen error messages and SQL codes ..................................... 4-9
  Warnings not visible in PL/SQL ..................................................... 4-9
  Unsupported predefined errors ..................................................... 4-9
  Possibility of runtime errors after clean compile (use of Oracle Database SQL parser) 4-9
  Use of TimesTen expressions at runtime ....................................... 4-10

5 Examples Using TimesTen SQL in PL/SQL

Examples using the SELECT...INTO statement in PL/SQL ................. 5-1
  Using SELECT... INTO to return sum of salaries ........................... 5-1
  Using SELECT... INTO to query another user’s table .................... 5-2

Example using the INSERT statement .............................................. 5-2

Examples using input and output parameters and bind variables ......... 5-3
  Using IN and OUT parameters .................................................... 5-3
  Using IN OUT parameters ........................................................ 5-4
  Using associative arrays ............................................................ 5-5

Examples using cursors ................................................................... 5-6
  Fetching values ........................................................................... 5-6
  Using the %ROWCOUNT and %NOTFOUND attributes ................... 5-8
  Using cursor FOR loops ............................................................... 5-8

Examples using FORALL and BULK COLLECT ............................... 5-9
  Using FORALL with SQL%BULK_ROWCOUNT ............................... 5-9
  Using BULK COLLECT INTO with queries ................................. 5-10
  Using BULK COLLECT INTO with cursors .................................. 5-11
  Using SAVE EXCEPTIONS with BULK COLLECT .......................... 5-12

Examples using EXECUTE IMMEDIATE ........................................... 5-13
  Using EXECUTE IMMEDIATE to create a table ............................. 5-14
  Using EXECUTE IMMEDIATE with a single row query .................... 5-14
  Using EXECUTE IMMEDIATE to alter a connection attribute .......... 5-15
  Using EXECUTE IMMEDIATE to call a TimesTen built-in procedure .. 5-16
  Using EXECUTE IMMEDIATE with TimesTen-specific syntax ........... 5-16

Examples using RETURNING INTO ............................................... 5-16
  Using the RETURNING INTO clause with a record ..................... 5-17
  Using BULK COLLECT INTO with the RETURNING INTO clause ..... 5-17

Examples using the AUTHID clause .............................................. 5-18
Oracle TimesTen In-Memory Database (TimesTen) is a relational database that is memory-optimized for fast response and throughput. The database resides entirely in memory at runtime and is persisted to disk storage for the ability to recover and restart. Replication features allow high availability. TimesTen supports standard application interfaces JDBC, ODBC, and ODP.NET, in addition to Oracle interfaces PL/SQL, OCI, and Pro*C/C++. TimesTen is available separately or as a cache for Oracle Database.

This document covers TimesTen support for PL/SQL.

The following topics are discussed in the preface:

- Audience
- Related documents
- Conventions
- Documentation Accessibility

**Audience**

This document is intended for anyone developing or supporting applications that use PL/SQL with TimesTen. Although it provides some overview, you should be familiar with PL/SQL or have access to more detailed documentation. This manual emphasizes TimesTen-specific functionality.

You should also be familiar with TimesTen, SQL (Structured Query Language), and database operations.

You would typically use PL/SQL through some programming interface such as those mentioned above, so should also consult the appropriate TimesTen developer documentation.

Also see “Audiences for this document” on page 1-4, which goes into more detail.

**Related documents**

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


Oracle Database documentation is also available on the Oracle Technology network. This may be especially useful for Oracle Database features that TimesTen supports but does not attempt to fully document.
In particular, these Oracle Database documents may be of interest:

- Oracle Database PL/SQL Language Reference
- Oracle Database PL/SQL Packages and Types Reference
- Oracle Database SQL Language Reference
- Oracle Database Reference

In addition, numerous third-party documents are available that describe PL/SQL in detail.

**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 and Linux platforms. Refer to the "Platforms" section in Oracle TimesTen In-Memory Database Release Notes for specific platform versions supported by TimesTen.

---

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

---

This document uses the following text conventions:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>italic</em></td>
<td>Italic type indicates terms defined in text, book titles, or emphasis.</td>
</tr>
<tr>
<td><em>monospace</em></td>
<td>Monospace type indicates commands, URLs, procedure and function names, package names, attribute names, directory names, file names, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>
| *italic monospace* | Italic monospace type indicates a placeholder or a variable in a code example for which you specify or use a particular value, such as in the following example:  
Driver=install_dir/lib/libtten.sl  
Replace *install_dir* with the path of your TimesTen installation directory. |
| []             | Square brackets indicate that an item in a command line is optional.    |
| {}             | Curly braces indicated that you must choose one of the items separated by a vertical bar ( | ) in a command line. |
| |             | A vertical bar (or pipe) separates alternative arguments.               |
| . . .           | An ellipsis ( . . . ) after an argument indicates that you may use more than one argument on a single command line. An ellipsis in a code example indicates that what is shown is only a partial example. |
| %             | The percent sign indicates the UNIX shell prompt.                       |

TimesTen documentation uses the following variables to identify path, file and user names.
<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>install_dir</td>
<td>The path that represents the directory where TimesTen is installed.</td>
</tr>
<tr>
<td>TTinstance</td>
<td>The instance name for your specific installation of TimesTen. Each</td>
</tr>
<tr>
<td></td>
<td>installation of TimesTen must be identified at installation time with a</td>
</tr>
<tr>
<td></td>
<td>unique instance name. This name appears in the installation path.</td>
</tr>
<tr>
<td>bits or bb</td>
<td>Two digits, either 32 or 64, that represent either a 32-bit or 64-bit</td>
</tr>
<tr>
<td></td>
<td>operating system.</td>
</tr>
<tr>
<td>release or rr</td>
<td>The first three parts in a release number, with or without dots. The</td>
</tr>
<tr>
<td></td>
<td>first three parts of a release number represent a major TimesTen release.</td>
</tr>
<tr>
<td></td>
<td>For example, 1122 or 11.2.2 represents TimesTen 11g Release 2 (11.2.2).</td>
</tr>
<tr>
<td>DSN</td>
<td>TimesTen data source name (for the TimesTen database).</td>
</tr>
</tbody>
</table>

**Documentation Accessibility**


**Access to Oracle Support**

What's New

This section summarizes the new features and functionality of Oracle TimesTen In-Memory Database 11g Release 2 (11.2.2) that are documented in this guide, providing links into the guide for more information.

New features in Release 11.2.2.0.0

- **LOB support**
  
  TimesTen supports LOBs (large objects). This includes CLOBs (character LOBs), NCLOBs (national character LOBs), and BLOBs (binary LOBs).
  
  For details of support in PL/SQL, refer to "Large objects (LOBs)" on page 2-17.

- **Associative array binds**
  
  Associative arrays, formerly known as index-by tables or PL/SQL tables, are supported as **IN**, **OUT**, or **IN OUT** bind parameters in TimesTen PL/SQL, such as from an OCI, Pro*C/C++, or JDBC application. This enables arrays of data to be passed efficiently between an application and the database.
  
  See "Using associative arrays from applications" on page 3-6.

- **Use of PL/SQL statements in dynamic SQL**
  
  TimesTen supports the use of PL/SQL anonymous blocks and procedure or function calls, in addition to SQL statements, in an **EXECUTE IMMEDIATE** statement or **DBMS_SQL** procedure or function call.
  
  Refer to "Dynamic SQL in PL/SQL (EXECUTE IMMEDIATE statement)" on page 2-12.
Introduction to PL/SQL in the TimesTen Database

TimesTen supports PL/SQL (Procedural Language Extension to SQL), a programming language that enables you to integrate procedural constructs with SQL in your database. TimesTen 11g Release 2 (11.2.2) implements the PL/SQL language from Oracle Database release 11.2.0.2. As such, most PL/SQL features present in that release of Oracle Database are also present in TimesTen, operating in essentially the same way. (Refer to Chapter 9, "TimesTen PL/SQL Support: Reference Summary" for differences.)

This chapter provides a brief introduction to TimesTen PL/SQL, covering the following topics:

- Features of PL/SQL in TimesTen
- TimesTen PL/SQL components and operations
- Audiences for this document
- About the TimesTen PL/SQL demos

Features of PL/SQL in TimesTen

PL/SQL support in TimesTen enables you to do the following:

- Take full advantage of the PL/SQL programming language.
- Execute PL/SQL from your client applications that use these APIs:
  - ODBC
  - JDBC
  - Oracle Call Interface (OCI)
  - Oracle Pro*C/C++
  - Oracle Data Provider for .NET (ODP.NET)
  - TTClasses (TimesTen C++ library)
- Execute TimesTen SQL from PL/SQL.
- Create, alter, or drop standalone procedures, functions, packages and package bodies.
- Use PL/SQL packages to extend your database functionality and to provide PL/SQL access to SQL features.
- Handle exceptions and errors in your PL/SQL applications.
Set connection attributes in your database to customize your PL/SQL environment.

- Alter session parameters so you can manage your PL/SQL environment.
- Display PL/SQL metadata in your database by using PL/SQL system views.

**TimesTen PL/SQL components and operations**

This section provides an overview of PL/SQL operations in TimesTen, including discussion of how an application interacts with PL/SQL and how PL/SQL components interact with other components of TimesTen. The following topics are covered:

- Application interaction with TimesTen and PL/SQL
- PL/SQL in TimesTen versus PL/SQL in Oracle Database

**Application interaction with TimesTen and PL/SQL**

Figure 1–1 shows the PL/SQL components and their interactions with each other and with other TimesTen components during PL/SQL operations.

*Figure 1–1  TimesTen PL/SQL components*

An application uses the API of its choice—ODBC, JDBC, OCI, Pro*C, ODP.NET, or TTClasses—to send requests to the database. ODBC is the TimesTen native API, so each of the other APIs ultimately calls the ODBC layer.

The ODBC driver calls the TimesTen SQL parser to examine each incoming request and determine whether it is SQL or PL/SQL. The request is then passed to the appropriate subsystem within TimesTen. PL/SQL source and SQL statements are compiled, optimized and executed by the PL/SQL subsystem and SQL subsystem, respectively.

The PL/SQL compiler is responsible for generating executable code from PL/SQL source, while the SQL compiler does the same for SQL statements. Each compiler generates intermediate code that can then be executed by the appropriate PL/SQL or SQL execution engine. This executable code, along with metadata about the PL/SQL blocks, is then stored in tables in the database.
When PL/SQL blocks are executed, the PL/SQL execution engine is invoked. As PL/SQL blocks in turn invoke SQL, the PL/SQL execution engine calls the TimesTen SQL compiler and the TimesTen SQL execution engine to handle SQL execution.

---

**Note:** The introduction of PL/SQL into TimesTen has little impact on applications that do not use it. If applications execute SQL directly, then requests are passed from the TimesTen ODBC driver to the TimesTen SQL compiler and execution engine in the same way as in previous releases.

---

**PL/SQL in TimesTen versus PL/SQL in Oracle Database**

PL/SQL processing in TimesTen is largely identical to its processing in Oracle Database. The PL/SQL compiler and execution engine that are included with TimesTen originated in Oracle Database, and the relationship between PL/SQL components and the SQL compiler and execution engine is comparable. The tables used to store PL/SQL units are the same in TimesTen and Oracle Database, as are the views that are available to query information about stored PL/SQL units.

Beyond these basic similarities, however, are some potentially significant differences. These are detailed in the following subsections:

- SQL statements in PL/SQL blocks
- Execution of PL/SQL from SQL

**SQL statements in PL/SQL blocks**

In TimesTen, as in Oracle Database, PL/SQL blocks may include SQL statements. Consider the anonymous block in the following example:

```sql
Command> create table tab2 (x number, last_name VARCHAR2 (25) INLINE NOT NULL);
Command> declare
>    x number;
> begin
>    select salary into x from employees where last_name = 'Whalen';
>    insert into tab2 values(x, 'Whalen');
> end;
> /
```

PL/SQL procedure successfully completed.

The PL/SQL compiler in TimesTen calls a copy of the Oracle Database SQL parser to analyze and validate the syntax of such SQL statements. This Oracle Database parser is included in TimesTen for this purpose. As part of this processing, PL/SQL may rewrite parts of the SQL statements (for example, by removing `INTO` clauses or replacing PL/SQL variables with binds). This processing is identical in TimesTen and in Oracle Database. The rewritten SQL statements are then included in the executable code for the PL/SQL block. When the PL/SQL block is executed, these SQL statements are compiled and executed by the TimesTen SQL subsystem.

In Oracle Database, the same SQL parser is used by the PL/SQL compiler and the SQL compiler. In TimesTen, however, different SQL parsers are used. TimesTen PL/SQL uses the Oracle Database SQL parser, while TimesTen SQL uses the native TimesTen SQL parser. This difference is typically, but not always, transparent to the end user. In particular, be aware of the following:

- SQL statements in TimesTen PL/SQL programs must obey Oracle Database SQL syntax. While TimesTen SQL is generally a subset of Oracle Database SQL, there...
are some expressions that are permissible in TimesTen SQL but not in Oracle Database SQL. Such TimesTen-specific SQL operations cannot be used within PL/SQL except by using dynamic SQL through `EXECUTE IMMEDIATE` statements or the `DBMS_SQL` package. See "Dynamic SQL in PL/SQL (EXECUTE IMMEDIATE statement)" on page 2-12.

- SQL statements that would be permissible in Oracle Database are accepted by the PL/SQL compiler as valid even if they cannot be executed by TimesTen. If SQL features are used that TimesTen does not support, compilation of a PL/SQL block may be successful, but a runtime error would occur when the PL/SQL block is executed.

**Execution of PL/SQL from SQL**

In Oracle Database, PL/SQL blocks can invoke SQL statements, and SQL statements can in turn invoke PL/SQL functions. For example, a stored procedure can invoke an `UPDATE` statement that employs a user-written PL/SQL function in its `WHERE` clause.

In TimesTen, a SQL statement cannot invoke a PL/SQL function.

In addition, TimesTen does not support triggers. (See "XLA and TimesTen Event Management" in Oracle TimesTen In-Memory Database C Developer’s Guide for information about XLA, a high-performance, asynchronous TimesTen alternative to triggers.)

**Audiences for this document**

There are two primary developer audiences for this document:

- Developers experienced with Oracle Database and Oracle Database PL/SQL who want to learn how to use PL/SQL in TimesTen: These readers want to learn the differences between PL/SQL in Oracle Database and PL/SQL in TimesTen.

- Developers experienced with TimesTen who are not familiar with PL/SQL: These readers need general information about PL/SQL.

The following subsections note areas of particular interest in this document for each audience.

**Developers experienced with Oracle Database and Oracle Database PL/SQL**

Developers experienced with Oracle Database PL/SQL can bypass much of this document, which covers many general concepts of PL/SQL. Likely areas of interest, particularly differences in PL/SQL functionality between Oracle Database and TimesTen, include the following. Note that TimesTen-specific considerations are discussed at the end of Chapter 2, Chapter 3, and Chapter 4 and throughout Chapter 9.

- "How to execute PL/SQL procedures and functions" on page 2-8: This includes a comparison between how you can execute them in TimesTen and in Oracle Database.

- "Differences in TimesTen: transaction behavior" on page 2-31: This discusses cursor behavior when a transaction ends in TimesTen.

- "Differences in TimesTen: data type considerations" on page 3-10: This includes TimesTen-specific conversions, and types that TimesTen does not support.

- "Differences in TimesTen: exception handing and error behavior" on page 4-7: This describes differences in error support, handling, and reporting.
About the TimesTen PL/SQL demos

- **Chapter 6, "PL/SQL Installation and Environment"**: This includes discussion of TimesTen connection attributes.
- **Chapter 8, "TimesTen Supplied PL/SQL Packages"**: This documents the subset of Oracle Database PL/SQL packages that TimesTen supports.
- **Chapter 9, "TimesTen PL/SQL Support: Reference Summary"**: This reference chapter provides a detailed treatment of differences between TimesTen PL/SQL and Oracle Database PL/SQL.

Developers experienced with TimesTen

Most of this document is geared toward readers without prior PL/SQL experience, especially prior TimesTen users who are not familiar with PL/SQL, and nearly the entire document should be useful. In particular, Chapter 2, "Programming Features in PL/SQL in TimesTen," will help these readers get started and Chapter 5, “Examples Using TimesTen SQL in PL/SQL,” includes some additional examples.

Chapter 9, "TimesTen PL/SQL Support: Reference Summary," is geared toward differences between TimesTen PL/SQL and Oracle Database PL/SQL and may be of less interest.

About the TimesTen PL/SQL demos

After you have configured your environment, you can confirm that everything is set up correctly by compiling and running the TimesTen Quick Start demo applications. Refer to the Quick Start welcome page at `install_dir/quickstart.html`, especially the links under SAMPLE PROGRAMS, for information about the following:

- **Demo schema and setup**: The `build_sampledb` script (.sh on UNIX or .bat on Windows) creates a sample database and demo schema. You must use this before you start using the demos.

- **Demo environment and setup**: The `ttquickstartenv` script (.sh or .csh on UNIX or .bat on Windows), a superset of the `ttenv` script generally used for TimesTen setup, sets up the demo environment. You must use this each time you enter a session where you want to compile or run any of the demos.

- **Demos and setup**: TimesTen provides demos for PL/SQL in a subdirectory under the `quickstart/sample_code` directory. For instructions on running the demos, see the README file in the subdirectory.

- **What the demos do**: A synopsis of each demo is provided when you click **PL/SQL** under SAMPLE PROGRAMS.
One of the advantages of PL/SQL in TimesTen is the ability to integrate PL/SQL procedural constructs with the flexible and powerful TimesTen SQL language.

This chapter surveys the main PL/SQL programming features described in "Overview of PL/SQL" in Oracle Database PL/SQL Language Reference. Working from simple examples, you will learn how to use PL/SQL in TimesTen. Unless otherwise noted, the examples have the same results in TimesTen as in Oracle Database.

See the end of the chapter for TimesTen-specific considerations. See “TimesTen PL/SQL components and operations” on page 1-2 for an overview of how applications interact with TimesTen in general and PL/SQL in particular.

The following are the main topics of this chapter:

- PL/SQL blocks
- PL/SQL variables and constants
- SQL function calls from PL/SQL
- PL/SQL control structures
- How to execute PL/SQL procedures and functions
- How to pass data between an application and PL/SQL
- Use of SQL in PL/SQL programs
- Use of cursors in PL/SQL programs
- PL/SQL procedures and functions
- PL/SQL packages
- Wrapping PL/SQL source code
- Differences in TimesTen: transaction behavior

---

**Note:** Except where stated otherwise, the examples in this guide use the TimesTen ttIsql utility. In order to display output in the examples, the setting `SET SERVEROUTPUT ON` is used. For more information on ttIsql, see "ttIsql" in Oracle TimesTen In-Memory Database Reference.

---

**PL/SQL blocks**

The basic unit of a PL/SQL source program is the block, or anonymous block, which groups related declarations and statements. TimesTen supports PL/SQL blocks.
A PL/SQL block is defined by the keywords DECLARE, BEGIN, EXCEPTION, and END. Example 2–1 shows the basic structure of a PL/SQL block.

---

**Note:** If you use TimesTen Application-Tier Database Cache (TimesTen Cache): A PL/SQL block cannot be passed through to Oracle Database. (Also see “TimesTen PL/SQL with TimesTen Cache” on page 2-16.)

---

**Example 2–1  PL/SQL block structure**

```
DECLARE -- (optional)
  -- Variables, cursors, user-defined exceptions
BEGIN -- (mandatory)
  -- PL/SQL statements
EXCEPTION -- (optional)
  -- Actions to perform when errors occur
END -- (mandatory)
```

You can define either anonymous or named blocks in your PL/SQL programs. This example creates an anonymous block that queries the employees table and returns the data in a PL/SQL variable:

```
Command> SET SERVEROUTPUT ON;
Command> DECLARE
  >    v_fname VARCHAR2 (20);
  > BEGIN
  >    SELECT first_name
  >    INTO v_fname
  >    FROM employees
  >    WHERE employee_id = 100;
  >  DBMS_OUTPUT.PUT_LINE (v_fname);
  > END;
  > /
Steven

PL/SQL procedure successfully completed.
```

---

**PL/SQL variables and constants**

You can define variables and constants in PL/SQL and then use them in procedural statements and in SQL anywhere an expression can be used.

For example:

```
Command> DECLARE
  >    v_hiredate DATE;
  >    v_deptno NUMBER (2) NOT NULL := 10;
  >    v_location VARCHAR2 (13) := 'San Francisco';
  >    c_comm CONSTANT NUMBER := 1400;
BEGIN
  > /
```

You can use the %TYPE attribute to declare a variable according to either a TimesTen column definition or another declared variable. For example, use %TYPE to create variables emp_lname and min_balance:

```
Command> DECLARE
  >    emp_lname employees.last_name%TYPE;
  >    balance NUMBER (7,2);
  >    min_balance balance%TYPE:= 1000;
BEGIN
```
You can assign a value to a variable in the following ways.

- With the assignment operator (:=), as shown in Example 2–2
- By selecting or fetching values into it, as shown in Example 2–3
- By passing the variable as an OUT or IN OUT parameter to a subprogram (procedure or function) and then assigning the value inside the subprogram, as shown in Example 2–4

---

**Note:** The `DBMS_OUTPUT` package used in these examples is supplied with TimesTen. For information on this and other supplied packages, refer to Chapter 8, "TimesTen Supplied PL/SQL Packages".

---

**Example 2–2 Assigning values to variables with the assignment operator**

```plsql
DECLARE -- Assign values in the declarative section
    wages NUMBER;
    hours_worked NUMBER := 40;
    hourly_salary NUMBER := 22.50;
    bonus NUMBER := 150;
    country VARCHAR2(128);
    counter NUMBER := 0;
    done BOOLEAN;
    valid_id BOOLEAN;
    emp_rec1 employees%ROWTYPE;
    emp_rec2 employees%ROWTYPE;
    TYPE commissions IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
    comm_tab commissions;
BEGIN -- Assign values in the executable section
    wages := (hours_worked * hourly_salary) + bonus;
    country := 'France';
    country := UPPER('Canada');
    done := (counter > 100);
    valid_id := TRUE;
    emp_rec1.first_name := 'Amy';
    emp_rec1.last_name := 'Peiner';
    emp_rec1 := emp_rec2;
    comm_tab(5) := 20000 * 0.15;
END;
```

PL/SQL procedure successfully completed.
Example 2–3 Using SELECT INTO to assign values to variables

Select 10% of an employee's salary into the bonus variable:

Command>
DECLARE
  bonus NUMBER(8,2);
  emp_id NUMBER(6) := 100;
BEGIN
  SELECT salary * 0.10 INTO bonus FROM employees
  WHERE employee_id = emp_id;
  DBMS_OUTPUT.PUT_LINE (bonus);
END;
/
2400
PL/SQL procedure successfully completed.

Example 2–4 Assigning values to variables as parameters of a subprogram

Declare the variable new_sal and then pass the variable as a parameter (sal) to procedure adjust_salary. Procedure adjust_salary computes the average salary for employees with job_id='ST_CLERK' and then updates sal. After the procedure is executed, the value of the variable is displayed to verify that the variable was correctly updated.

Command>
DECLARE
  new_sal NUMBER(8,2);
  emp_id NUMBER(6) := 126;
PROCEDURE adjust_salary (emp_id NUMBER, sal IN OUT NUMBER) IS
  emp_job VARCHAR2(10);
  avg_sal NUMBER(8,2);
BEGIN
  SELECT job_id INTO emp_job FROM employees
  WHERE employee_id = emp_id;
  SELECT AVG(salary) INTO avg_sal FROM employees
  WHERE job_id = emp_job;
  DBMS_OUTPUT.PUT_LINE ('The average salary for ' || emp_job
  || ' employees: ' || TO_CHAR(avg_sal));
  sal := (sal + avg_sal)/2;
  DBMS_OUTPUT.PUT_LINE ('New salary is ' || sal);
END;
BEGIN
  SELECT AVG(salary) INTO new_sal FROM employees;
  DBMS_OUTPUT.PUT_LINE ('The average salary for all employees: ' ||
  TO_CHAR(new_sal));
  adjust_salary(emp_id, new_sal);
  DBMS_OUTPUT.PUT_LINE ('Salary should be same as new salary ' ||
  new_sal);
END;
/
The average salary for all employees: 6461.68
The average salary for ST_CLERK employees: 2785
New salary is 4623.34

Note: This example uses records, which are composite data structures that have fields with different data types. You can use the %ROWTYPE attribute, as shown, to declare a record that represents a row in a table or a row from a query result set. Records are further discussed under "PL/SQL composite data types" on page 3-4.
Salary should be same as new salary 4623.34

PL/SQL procedure successfully completed.

---

**Note:** This example illustrates the ability to nest PL/SQL blocks within blocks. The outer anonymous block contains an enclosed procedure. This `PROCEDURE` statement is distinct from the `CREATE PROCEDURE` statement documented in "PL/SQL procedures and functions" on page 2-21, which creates a subprogram that remains stored in the user’s schema.

---

**SQL function calls from PL/SQL**

Most SQL functions are supported for calls directly from PL/SQL. In the first example that follows, the function `RTRIM` is used as a PL/SQL function in a PL/SQL assignment statement. In the second example, it is used as a SQL function in a static SQL statement.

**Example 2–5 Using the RTRIM function from PL/SQL**

Use the TimesTen PL/SQL `RTRIM` built-in function to remove the right-most "x" and "y" characters from the string. Note that `RTRIM` is used in a PL/SQL assignment statement.

```
Command> DECLARE p_var VARCHAR2(30);
  > BEGIN
  >   p_var := RTRIM ('RTRIM Examplexxxxyyxyxy', 'xy');
  >   DBMS_OUTPUT.PUT_LINE (p_var);
  > END;
  > /
RTRIM Example

PL/SQL procedure successfully completed.
```

**Example 2–6 Using the RTRIM function from SQL**

Use the TimesTen SQL function `RTRIM` to remove the right-most "x" and "y" characters from the string. Note that `RTRIM` is used in a static SQL statement.

```
Command> DECLARE tt_var VARCHAR2 (30);
  > BEGIN
  >   SELECT RTRIM ('RTRIM Examplexxxxyyxyxy', 'xy')
  >   INTO tt_var FROM DUAL;
  >   DBMS_OUTPUT.PUT_LINE (tt_var);
  > END;
  > /
RTRIM Example

PL/SQL procedure successfully completed.
```

You can refer to information about SQL functions in TimesTen under "Expressions" in Oracle TimesTen In-Memory Database SQL Reference. See "SQL Functions in PL/SQL Expressions" in Oracle Database PL/SQL Language Reference for information about support for SQL functions in PL/SQL.
PL/SQL control structures

Control structures are among the PL/SQL extensions to SQL. TimesTen supports the same control structures as Oracle Database.

The following control structures are discussed in this section.

- Conditional control
- Iterative control
- CONTINUE statement

Conditional control

The IF-THEN-ELSE and CASE constructs are examples of conditional control. In Example 2–7, the IF-THEN-ELSE construct is used to determine the salary raise of an employee based on the current salary. The CASE construct is also used to choose the course of action to take based on the job_id of the employee.

Example 2–7 Using the IF-THEN-ELSE and CASE constructs

Command>

```sql
DECLARE
  jobid employees.job_id%TYPE;
  empid employees.employee_id%TYPE := 115;
  sal employees.salary%TYPE;
  sal_raise NUMBER(3,2);
BEGIN
  SELECT job_id, salary INTO jobid, sal FROM employees
  WHERE employee_id = empid;
  CASE
    WHEN jobid = 'PU_CLERK' THEN
      IF sal < 3000 THEN sal_raise := .12;
      ELSE sal_raise := .09;
      END IF;
    WHEN jobid = 'SH_CLERK' THEN
      IF sal < 4000 THEN sal_raise := .11;
      ELSE sal_raise := .08;
      END IF;
    WHEN jobid = 'ST_CLERK' THEN
      IF sal < 3500 THEN sal_raise := .10;
      ELSE sal_raise := .07;
      END IF;
    ELSE
      BEGIN
        DBMS_OUTPUT.PUT_LINE('No raise for this job: ' || jobid);
        END;
    END CASE;
  DBMS_OUTPUT.PUT_LINE('Original salary ' || sal);
  -- Update
  UPDATE employees SET salary = salary * sal_raise
  WHERE employee_id = empid;
END;
/```

Original salary 3100

PL/SQL procedure successfully completed.
Iterative control

An iterative control construct executes a sequence of statements repeatedly, as long as a specified condition is true. Loop constructs are used to perform iterative operations.

There are three loop types:

- Basic loop
- FOR loop
- WHILE loop

The basic loop performs repetitive actions without overall conditions. The FOR loop performs iterative actions based on a count. The WHILE loops perform iterative actions based on a condition.

**Example 2–8 Using a WHILE loop**

```
Command> CREATE TABLE temp (tempid NUMBER(6),
> tempsal NUMBER(8,2),
> tempname VARCHAR2(25));
Command> DECLARE
> sal employees.salary%TYPE := 0;
> mgr_id employees.manager_id%TYPE;
> lname employees.last_name%TYPE;
> starting_empid employees.employee_id%TYPE := 120;
> BEGIN
> SELECT manager_id INTO mgr_id
> FROM employees
> WHERE employee_id = starting_empid;
> WHILE sal <= 15000 LOOP -- loop until sal > 15000
> SELECT salary, manager_id, last_name INTO sal, mgr_id, lname
> FROM employees WHERE employee_id = mgr_id;
> END LOOP;
> INSERT INTO temp VALUES (NULL, sal, lname);  -- insert NULL for tempid
> COMMIT;
> EXCEPTION
> WHEN NO_DATA_FOUND THEN
> INSERT INTO temp VALUES (NULL, NULL, 'Not found');  -- insert NULLs
> COMMIT;
> END;
> /
```

PL/SQL procedure successfully completed.

```
Command> SELECT * FROM temp;
< <NULL>, 24000, King >
1 row found.
```

**CONTINUE statement**

The CONTINUE statement enables you to transfer control within a loop back to a new iteration.

**Example 2–9 Using the CONTINUE statement**

In this example, the first \( v_{\text{total}} \) assignment is executed for each of the 10 iterations of the loop. The second \( v_{\text{total}} \) assignment is executed for the first five iterations of the loop. The CONTINUE statement transfers control within a loop back to a new iteration, so for the last five iterations of the loop, the second \( v_{\text{total}} \) assignment is not executed. The end \( v_{\text{total}} \) value is 70.
How to execute PL/SQL procedures and functions

TimesTen supports execution of PL/SQL from client applications using ODBC, OCI, Pro*C/C++, ODP.NET, JDBC, or TimesTen TTClasses (for C++).

As noted earlier, a block is the basic unit of a PL/SQL source program. Anonymous blocks were also discussed earlier. By contrast, procedures and functions are PL/SQL blocks that have been defined with a specified name. See “PL/SQL procedures and functions” on page 2-21 for how to define and create them.

In TimesTen, a PL/SQL procedure or function that is standalone (created with CREATE PROCEDURE or CREATE FUNCTION) or part of a package can be executed using an anonymous block or a CALL statement. (See "CALL" in Oracle TimesTen In-Memory Database SQL Reference for details about CALL syntax.)

Consider the following function:

```sql
CREATE OR REPLACE FUNCTION mytest RETURN NUMBER IS
BEGIN
  RETURN 1;
END;
```

In TimesTen, you can execute mytest in either of the following ways.

- In an anonymous block:

  ```sql
  DECLARE
  >  v_total  SIMPLE_INTEGER := 0;
  >  BEGIN
  >  FOR i IN 1..10 LOOP
  >    v_total := v_total + i;
  >    DBMS_OUTPUT.PUT_LINE ('Total is : ' || v_total);
  >    CONTINUE WHEN i > 5;
  >    v_total := v_total + i;
  >    DBMS_OUTPUT.PUT_LINE ('Out of loop Total is: ' || v_total);
  >  END LOOP;
  >  END;
  >  /
  Total is : 1
  Out of loop Total is: 2
  Total is : 4
  Out of loop Total is: 6
  Total is : 9
  Out of loop Total is: 12
  Total is : 16
  Out of loop Total is: 20
  Total is : 25
  Out of loop Total is: 30
  Total is : 36
  Total is : 43
  Total is : 51
  Total is : 60
  Total is : 70

  PL/SQL procedure successfully completed.
  ```
PL/SQL procedure successfully completed.

Command> print n;
N : 1

- **In a CALL statement:**
  
  Command> variable n number;
  Command> call mytest() into :n;
  Command> print n;
  N : 1

In Oracle Database, you could also execute `mytest` through a SQL statement, as follows. This execution mechanism is *not* supported in TimesTen.

- **In a SELECT statement:**

  SQL> select mytest from dual;

  MYTEST
  --------
  1

---

**Note:** A user's own procedure takes precedence over a TimesTen built-in procedure with the same name, but it is best to avoid such naming conflicts.

---

### How to pass data between an application and PL/SQL

This section covers the following topics for passing data between an application and PL/SQL:

- **Using bind variables from an application**
- **IN, OUT, and IN OUT parameter modes**

Refer to "Bind Variables" in *Oracle Database PL/SQL Language Reference* for additional information.

#### Using bind variables from an application

You can use `:var` notation for bind variables to be passed between your application (such as a C or Java application) and PL/SQL. The term *bind variable* (or sometimes *host variable*) is used equivalently to how the term *parameter* has historically been used in TimesTen, and bind variables from an application would correspond to the parameters declared in a PL/SQL procedure or function specification.

Here is a simple example using `ttIsql` to call a PL/SQL procedure that retrieves the name and salary of the employee corresponding to a specified employee ID. In this example, `ttIsql` essentially acts as the calling application, and the name and salary are output from PL/SQL:

Command> VARIABLE b_name VARCHAR2 (25)
Command> VARIABLE b_sal  NUMBER

Command> BEGIN
  > query_emp (171, :b_name, :b_sal);
  > END;
  > /

Programming Features in PL/SQL in TimesTen 2-9
PL/SQL procedure successfully completed.

Command> PRINT b_name
B_NAME               : Smith
Command> PRINT b_sal
B_SAL                : 7400

See "Examples using input and output parameters and bind variables" on page 5-3 for the complete example.

See "PL/SQL procedures and functions" on page 2-21 for how to create and define procedures and functions.

See "Binding parameters and executing statements" in Oracle TimesTen In-Memory Database C Developer’s Guide and "Preparing SQL statements and setting input parameters" in Oracle TimesTen In-Memory Database Java Developer’s Guide for additional information and examples for those languages.

---

**Notes:**

- The TimesTen binding mechanism (early binding) differs from that of Oracle Database (late binding). TimesTen requires the data types before preparing queries. As a result, there will be an error if the data type of each bind parameter is not specified or cannot be inferred from the SQL statement. This would apply, for example, to the following statement:

  ```sql
  SELECT 'x' FROM DUAL WHERE :a = :b;
  ```

  You could address the issue as follows, for example:

  ```sql
  SELECT 'x' from DUAL WHERE CAST(:a as VARCHAR2(10)) = CAST(:b as VARCHAR2(10));
  ```

- For duplicate parameters, the implementation in PL/SQL in TimesTen is no different than the implementation in PL/SQL in Oracle Database.

- The term “bind parameter” as used in TimesTen developer guides (in keeping with ODBC terminology) is equivalent to the term "bind variable" as used in TimesTen PL/SQL documents (in keeping with Oracle Database PL/SQL terminology).

---

**IN, OUT, and IN OUT parameter modes**

Parameter modes define whether parameters declared in a PL/SQL subprogram (procedure or function) specification are used for input, output, or both. The three parameter modes are **IN** (the default), **OUT**, and **IN OUT**.

An **IN** parameter lets you pass a value to the subprogram being invoked. Inside the subprogram, an **IN** parameter acts like a constant and cannot be assigned a value. You can pass a constant, literal, initialized variable, or expression as an **IN** parameter.

An **OUT** parameter returns a value to the caller of a subprogram. Inside the subprogram, an **OUT** parameter acts like a variable. You can change its value and reference the value after assigning it.
An **IN OUT** parameter passes an initial value to a subprogram and returns an updated value to the caller. It can be assigned a value and its value can be read. Typically, an **IN OUT** parameter is a string buffer or numeric accumulator that is read inside the subprogram and then updated. The actual parameter that corresponds to an **IN OUT** formal parameter must be a variable, not a constant or an expression.

---

**Note:** TimesTen supports the binding of associative arrays (but not varrays or nested tables) as **IN**, **OUT**, or **IN OUT** parameters. See "Using associative arrays from applications" on page 3-6.

---

See "Examples using input and output parameters and bind variables" on page 5-3.

### Use of SQL in PL/SQL programs

PL/SQL is tightly integrated with the TimesTen database through the SQL language. This section covers use of the following SQL features in PL/SQL:

- Static SQL in PL/SQL for queries and DML statements
- Dynamic SQL in PL/SQL (EXECUTE IMMEDIATE statement)
- **FORALL** and **BULK COLLECT** operations
- **RETURNING INTO** clause
- TimesTen PL/SQL with TimesTen Cache
- Large objects (LOBs)

#### Static SQL in PL/SQL for queries and DML statements

From within PL/SQL, you can execute the following as static SQL:

- DML statements: **INSERT**, **UPDATE**, **DELETE**, and **MERGE**
- Queries: **SELECT**
- Transaction control: **COMMIT** and **ROLLBACK**

---

**Notes:**

- You must use dynamic SQL to execute DDL statements in PL/SQL. See the next section, "Dynamic SQL in PL/SQL (EXECUTE IMMEDIATE statement)".
- See "Differences in TimesTen: transaction behavior" on page 2-31 for details about how TimesTen transaction behavior differs from Oracle Database behavior.

---

For information on these SQL statements, refer to "SQL Statements" in *Oracle TimesTen In-Memory Database SQL Reference.*

*Example 2–10* shows how to execute a query. For additional examples using TimesTen SQL in PL/SQL, see Chapter 5, "Examples Using TimesTen SQL in PL/SQL".

**Example 2–10  Retrieving data with SELECT...INTO**

Use the **SELECT... INTO** statement to retrieve exactly one row of data. TimesTen returns an error for any query that returns no rows or multiple rows.
This example retrieves hire_date and salary for the employee with employee_id=100 from the employees table of the HR schema.

Command> run selectinto.sql

```
DECLARE
  v_emp_hiredate employees.hire_date%TYPE;
  v_emp_salary employees.salary%TYPE;
BEGIN
  SELECT hire_date, salary
  INTO   v_emp_hiredate, v_emp_salary
  FROM   employees
  WHERE  employee_id = 100;
  DBMS_OUTPUT.PUT_LINE(v_emp_hiredate || ' ' || v_emp_salary);
END;
/

1987-06-17 24000

PL/SQL procedure successfully completed.
```

**Dynamic SQL in PL/SQL (EXECUTE IMMEDIATE statement)**

You can use native dynamic SQL, through the EXECUTE IMMEDIATE statement, to accomplish any of the following:

- Execute a DML statement such as INSERT, UPDATE, or DELETE.
- Execute a DDL statement such as CREATE or ALTER. For example, you can use ALTER SESSION to change a PL/SQL first connection attribute.
- Execute a PL/SQL anonymous block.
- Call a PL/SQL stored procedure or function.
- Call a TimesTen built-in procedure. (See "Built-In Procedures" in Oracle TimesTen In-Memory Database Reference.)

One use case is if you do not know the full text of your SQL statement until execution time. For example, during compilation you may not know the name of the column to use in the WHERE clause of your SELECT statement. In such a situation, you can use the EXECUTE IMMEDIATE statement.

Another use case is for DDL, which cannot be executed in static SQL from within PL/SQL.

To call a TimesTen built-in procedure that returns a result set, create a record type and use EXECUTE IMMEDIATE with BULK COLLECT to fetch the results into an array.

Example 2–11 provides a set of brief examples of EXECUTE IMMEDIATE. For additional examples, see "Examples using EXECUTE IMMEDIATE" on page 5-13.

For more information, see "EXECUTE IMMEDIATE Statement" in Oracle Database PL/SQL Language Reference.
Example 2–11  Using EXECUTE IMMEDIATE to execute PL/SQL

This provides a set of examples using PL/SQL in EXECUTE IMMEDIATE statements. (The examples are independent of each other.)

Create a table and execute a DML statement on it within a PL/SQL block, specifying the input parameter through a USING clause. Then select the table to see the result.

```
Command> create table t(i int);
Command> declare
    >    i number := 1;
    > begin
    >    execute immediate 'begin insert into t values(:j);end;' using i;
    > end;
    > /

PL/SQL procedure successfully completed.

Command> select * from t;
< 1 >
1 row found.
```

Create a PL/SQL procedure `foo` then execute it in a PL/SQL block, specifying the input parameter through a USING clause:

```
Command> create or replace procedure foo(message varchar2) is
    > begin
    >    dbms_output.put_line(message);
    > end;
    > /

Procedure created.

Command> begin
    >    execute immediate 'begin foo(:b);end;' using 'hello';
    > end;
    > /

hello

PL/SQL procedure successfully completed.
```

Create a PL/SQL procedure `myprint` then execute it through a CALL statement, specifying the input parameter through a USING clause:

```
Command> declare
    >    a number := 1;
    > begin

Notes:
- See "Differences in TimesTen: transaction behavior" on page 2-31 for important information.
- As a DDL statement is being parsed to drop a procedure or a package, a timeout occurs if the procedure, or a procedure in the package, is still in use. After a call to a procedure, that procedure is considered to be in use until execution has returned to the user side. Any such deadlock times out after a short time.
- You can also use the DBMS_SQL package for dynamic SQL. See "DBMS_SQL" on page 8-8.
Use of SQL in PL/SQL programs

```sql
> execute immediate 'call myprint(:b)' using a;
> end;
> /
myprint procedure got number 1

PL/SQL procedure successfully completed.

Usage notes for dynamic SQL
Code that is executed through EXECUTE IMMEDIATE generally shares the same environment as the outer PL/SQL block, as in Oracle Database. In particular, be aware of the following. (These points apply to using DBMS_SQL as well as EXECUTE IMMEDIATE.)

- SQL and PL/SQL executed through EXECUTE IMMEDIATE run in the same transaction as the outer block.
- Any exception raised during execution of an EXECUTE IMMEDIATE statement is propagated to the outer block. Therefore, any errors on the error stack when the EXECUTE IMMEDIATE statement is executed are visible inside the outer block. This is useful for procedures such as DBMSUTILITY FORMAT_ERROR_STACK.
- Errors on the error stack prior to execution of a PL/SQL block in an EXECUTE IMMEDIATE statement are visible inside the block, for example by using DBMS_UTILITY FORMAT_ERROR_STACK.
- The execution environment in which an EXECUTE IMMEDIATE statement executes is the same as for the outer block. PL/SQL and TimesTen parameters, REF CURSOR state, and package state from the EXECUTE IMMEDIATE statement are visible inside the outer block.

FORALL and BULK COLLECT operations

Bulk binding is a powerful feature used in the execution of SQL statements from PL/SQL to move large amounts of data between SQL and PL/SQL. (This is different from binding parameters from an application program to PL/SQL.) With bulk binding, you bind arrays of values in a single operation rather than using a loop to perform FETCH, INSERT, UPDATE, and DELETE operations multiple times. TimesTen supports bulk binding, which can result in significant performance improvement.

Use the FORALL statement to bulk-bind input collections before sending them to the SQL engine. Use BULK COLLECT to bring back batches of results from SQL. You can bulk-collect into any type of PL/SQL collection, such as a varray, nested table, or associative array (index-by table). For additional information on collections, refer to "Using collections in PL/SQL" on page 3-5.

You can use the %BULK_EXCEPTIONS cursor attribute and the SAVE EXCEPTIONS clause with FORALL statements. SAVE EXCEPTIONS allows an UPDATE, INSERT, or DELETE statement to continue executing after it issues an exception (for example, a constraint error). Exceptions are collected into an array that you can examine using %BULK_ EXCEPTIONS after the statement has executed. When you use SAVE EXCEPTIONS, if exceptions are encountered during the execution of the FORALL statement, then all rows in the collection are processed. When the statement finishes, an error is issued to indicate that at least one exception occurred. If you do not use SAVE EXCEPTIONS, then when an exception is issued during a FORALL statement, the statement returns the exception immediately and no other rows are processed.

Refer to "Using FORALL Statement and BULK COLLECT Clause Together" in Oracle Database PL/SQL Language Reference for more information on these features.
Example 2–12 shows basic use of bulk binding and the FORALL statement. For more information and examples on bulk binding, see "Examples using FORALL and BULK COLLECT" on page 5-9.

**Example 2–12 Using the FORALL statement**

In the following example, the PL/SQL program increases the salary for employees with IDs 100, 102, 104, or 110. The FORALL statement bulk-binds the collection.

```
Command> CREATE OR REPLACE PROCEDURE raise_salary (p_percent NUMBER) IS
    >   TYPE numlist_type IS TABLE OF NUMBER
    >   INDEX BY BINARY_INTEGER;
    >   v_id  numlist_type;  -- collection
    > BEGIN
    >   v_id(1) := 100; v_id(2) := 102; v_id (3) := 104; v_id (4) := 110;
    >   -- bulk-bind the associative array
    >   FORALL i IN v_id.FIRST .. v_id.LAST
    >    UPDATE employees
    >     SET salary = (1 + p_percent/100) * salary
    >     WHERE employee_id = v_id (i);
    > END;
    > /
Procedure created.
```

Find out salaries before executing the raise_salary procedure:

```
Command> SELECT salary FROM employees WHERE employee_id = 100 OR employee_id =
102 OR employee_id = 104 OR employee_id = 100;
< 24000 >
< 17000 >
< 6000 >
3 rows found.
```

Execute the procedure and verify results as follows.

```
Command> EXECUTE raise_salary (10);

PL/SQL procedure successfully completed.
```

```
Command> SELECT salary FROM employees WHERE employee_id = 100 or employee_id =
102 OR employee_id = 104 OR employee_id = 100;
< 26400 >
< 18700 >
< 6600 >
3 rows found.
```

**RETURNING INTO clause**

You can use a RETURNING INTO clause, sometimes referred to as DML returning, with an INSERT, UPDATE, or DELETE statement to return specified columns or expressions, optionally including rowids, from rows that were affected by the action. This eliminates the need for a subsequent SELECT statement and separate round trip, in case, for example, you want to confirm what was affected or want the rowid after an insert or update.

A RETURNING INTO clause can be used with dynamic SQL (with EXECUTE IMMEDIATE) or static SQL.

Through the PL/SQL BULK COLLECT feature, the clause can return items from a single row into either a set of parameters or a record, or can return columns from multiple
Use of SQL in PL/SQL programs

rows into a PL/SQL collection such as a varray, nested table, or associative array (index-by table). Parameters in the INTO part of the clause must be output only, not input/output. For information on collections, refer to "Using collections in PL/SQL" on page 3-5. For BULK COLLECT, see "FORALL and BULK COLLECT operations" on page 2-14 and "Examples using FORALL and BULK COLLECT" on page 5-9.

SQL syntax and restrictions for the RETURNING INTO clause in TimesTen are documented as part of the "INSERT", "UPDATE", and "DELETE" documentation in Oracle TimesTen In-Memory Database SQL Reference.

Also see "Examples using RETURNING INTO" on page 5-16.

Refer to "RETURNING INTO Clause" in Oracle Database PL/SQL Language Reference for additional information about DML returning.

TimesTen PL/SQL with TimesTen Cache

When PL/SQL programs execute SQL statements, the SQL statements are processed by TimesTen in the same manner as when SQL is executed from applications written in other programming languages. All standard behaviors of TimesTen SQL apply. In an TimesTen Cache environment, this includes the ability to use all cache features from PL/SQL. When PL/SQL accesses tables in cache groups, the normal rules for those tables apply. For example, issuing a SELECT statement against a cache instance in a dynamic cache group may cause the instance to be automatically loaded into TimesTen from Oracle Database.

In particular, be aware of the following points about this functionality.

■ When you use static SQL in PL/SQL, any tables accessed must exist in TimesTen or the PL/SQL will not compile successfully. In the following example, ABC must exist in TimesTen.

        begin
            insert into abc values(1, 'Y');
        end;

■ In an TimesTen Cache environment, there is the capability to use the TimesTen passthrough facility to automatically route SQL statements from TimesTen to Oracle Database. (See "Setting a passthrough level" in Oracle TimesTen Application-Tier Database Cache User’s Guide for details of the passthrough facility.)

With passthrough=1, a statement can be passed through to Oracle Database if any accessed table does not exist in TimesTen. In PL/SQL, however, the statement would have to be executed using dynamic SQL.

Updating the preceding example, the following TimesTen PL/SQL block could be used to access ABC in Oracle Database with passthrough=1:

        begin
            execute immediate 'insert into abc values(1, 'Y')';
        end;

In this case, TimesTen PL/SQL can compile the block because the SQL statement is not examined at compile time.

■ While PL/SQL can be executed in TimesTen, in the current release the TimesTen passthrough facility cannot be used to route PL/SQL blocks from TimesTen to Oracle Database. For example, when using TimesTen Cache with passthrough=3, statements executed on a TimesTen connection are routed to Oracle Database in most circumstances. In this scenario, you may not execute PL/SQL blocks from your application program, because TimesTen would attempt to forward them to
Use of SQL in PL/SQL programs

Oracle Database, which is not supported. (In the passthrough=1 example, it is just the SQL statement being routed to Oracle Database, not the block as a whole.)

---

**Important:** PL/SQL procedures and functions can use any of the following cache operations with either definer's rights or invoker's rights:

- Loading or refreshing a cache group with commit every \( n \) rows
- DML on AWT cache groups
- DML on non-propagated cache groups (user managed cache groups without PROPAGATE enabled)
- SELECT on cache group tables that do not invoke passthrough or dynamic load
- UNLOAD CACHE GROUP

PL/SQL procedures or functions that use any of the following cache operations must use invoker’s rights (AUTHID CURRENT_USER): passthrough, dynamic loading of a cache group, loading or refreshing a cache group using WITH ID, DDL on cache groups, DML on SWT cache groups, or FLUSH CACHE GROUP.

See "Definer’s rights and invoker’s rights" on page 7-7.

---

**Large objects (LOBs)**

TimesTen supports LOBs (large objects). This includes CLOBs (character LOBs), NCLOBs (national character LOBs), and BLOBs (binary LOBs).

PL/SQL language features support LOBs in TimesTen as they do in Oracle Database, unless noted otherwise.

This section provides a brief overview of LOBs and discusses their use in PL/SQL, covering the following topics.

- About LOBs
- LOB locators
- Temporary LOBs
- Differences between TimesTen LOBs and Oracle Database LOBs
- Using LOBs
- PL/SQL package support for LOBs

**Note:** TimesTen does not support CLOBs if the database character set is TIMESTEN8.

You can also refer to the following:

- "LOB data types" in Oracle TimesTen In-Memory Database SQL Reference for additional information about LOBs in TimesTen
- Oracle Database SecureFiles and Large Objects Developer’s Guide for general information about programming with LOBs (but not specific to TimesTen functionality)
About LOBs

A LOB is a large binary object (BLOB) or character object (CLOB or NCLOB). In TimesTen, a BLOB can be up to 16 MB in size and a CLOB or NCLOB up to 4 MB. LOBs in TimesTen have essentially the same functionality as in Oracle Database, except as noted otherwise. (See "Differences between TimesTen LOBs and Oracle Database LOBs" on page 2-19.)

LOBs may be either persistent or temporary. A persistent LOB exists in a LOB column in the database. A temporary LOB exists only within an application.

LOB locators

In PL/SQL, a LOB consists of a LOB locator and a LOB value. The locator is an opaque structure that acts as a handle to the value. When an application uses a LOB in an operation such as passing a LOB as a parameter, it is passing the locator, not the actual value.

---

**Important:** LOB manipulations through APIs that use LOB locators result in usage of TimesTen temporary space. Any significant number of such manipulations may necessitate a size increase for the TimesTen temporary data region. See "TempSize" in Oracle TimesTen In-Memory Database Reference.

---

To update a LOB, your transaction must have an exclusive lock on the row containing the LOB. You can accomplish this by selecting the LOB with a `SELECT ... FOR UPDATE` statement. This results in a writable locator. With a simple `SELECT` statement, the locator is read-only. Read-only and writable locators behave as follows.

- A read-only locator is read consistent, meaning that throughout its lifetime, it sees only the contents of the LOB as of the time it was selected. Note that this would include any uncommitted updates made to the LOB within the same transaction prior to when the LOB was selected.

- A writable locator is updated with the latest data from the database each time a write is made through the locator. So each write is made to the most current data of the LOB, including updates that have been made through other locators.

The following example details behavior for two writable locators for the same LOB:

1. The LOB column contains “XY”.
2. Select locator `L1` for update.
3. Select locator `L2` for update.
4. Write "Z" through `L1` at offset 1.
5. Read through locator `L1`. This would return "ZY".
6. Read through locator `L2`. This would return "XY", because `L2` remains read-consistent until it is used for a write.
7. Write "W" through `L2` at offset 2.
8. Read through locator `L2`. This would return "ZW". Prior to the write in the preceding step, the locator was updated with the latest data ("ZY").
Temporary LOBs
A PL/SQL block can create a temporary LOB explicitly, for its own use. In TimesTen, the lifetime of such a LOB does not extend past the end of the transaction in which it is created (as is the case with the lifetime of any LOB locator in TimesTen).

A temporary LOB may also be created implicitly by TimesTen. For example, if a SELECT statement selects a LOB concatenated with an additional string of characters, TimesTen implicitly creates a temporary LOB to contain the concatenated data. Note that a temporary LOB is a server-side object. TimesTen has no concept of client-side LOBs.

Temporary LOBs are stored in the TimesTen temporary data region.
See “CREATETEMPORARY procedures” in Oracle TimesTen In-Memory Database PL/SQL Packages Reference for how to create temporary LOBs.

Differences between TimesTen LOBs and Oracle Database LOBs
Be aware of the following:

- A key difference between the TimesTen LOB implementation and the Oracle Database implementation is that in TimesTen, LOB locators do not remain valid past the end of the transaction. All LOB locators are invalidated after a commit or rollback, whether explicit or implicit. This includes after any DDL statement if TimesTen DDLCommitBehavior is set to 0 (the default), for Oracle Database behavior, which is always the case for PL/SQL users.

- TimesTen does not support BFILEs, SecureFiles, array reads and writes for LOBs, or callback functions for LOBs.

- In TimesTen, the DBMS_LOB FRAGMENT procedures are not supported, so you can write data into the middle of a LOB only by overwriting previous data. There is no functionality to insert data into the middle of a LOB and move previous data, beginning at that point, higher in the LOB correspondingly. Similarly, in TimesTen you can delete data from the middle of a LOB only by overwriting previous data with zeros or null data. There is no functionality to remove data from the middle of a LOB and move previous data, beginning at that point, lower in the LOB correspondingly. In either case in TimesTen, the size of the LOB does not change, except in the circumstance where from the specified offset there is less space available in the LOB than there is data to write. (In Oracle Database there is functionality for either mode, either overwriting and not changing the size of the LOB, or inserting or deleting and changing the size of the LOB.)

- TimesTen does not support binding arrays of LOBs.

- TimesTen does not support batch processing of LOBs.

- Relevant to BLOBs, there are differences in the usage of hexadecimal literals in TimesTen. See the description of HexadecimalLiteral in "Constants" in Oracle TimesTen In-Memory Database SQL Reference.

Using LOBs
The following shows basic use of a CLOB. Assume a table defined and populated as follows, with a BLOB column (not used here) and a CLOB column:

```
Command> create table t1 (a int, b blob, c clob);
Command> insert into t1(a,b,c) values(1, 0x123451234554321, 'abcde');
1 row inserted.
Command> commit;
```
Select a CLOB from the table and display it:

Command> declare
    myclob clob;
begin
    select c into myclob from t1 where a=1;
    dbms_output.put_line('CLOB selected from table t1 is: ' || myclob);
end;
/
CLOB selected from table t1 is: abcde

PL/SQL procedure successfully completed.

The following tries to display the temporary CLOB again after a commit statement has ended the transaction, showing that the LOB locator becomes invalid from that point:

Command> declare
    myclob clob;
begin
    select c into myclob from t1 where a=1;
    dbms_output.put_line('CLOB selected from table t1 is: ' || myclob);
    commit;
    dbms_output.put_line('CLOB after end of transaction is: ' || myclob);
end;
/
1806: invalid LOB locator specified
8507: ORA-06512: at line 8
CLOB selected from table t1 is: abcde
The command failed.

**PL/SQL package support for LOBs**

TimesTen supports subprograms of the DBMS_LOB package for manipulation of LOB data.

See "DBMS_LOB" on page 8-2 in this document for a list and descriptions of these subprograms. See "DBMS_LOB" in Oracle TimesTen In-Memory Database PL/SQL Packages Reference for further information.

**Passthrough LOBs**

Passthrough LOBs, which are LOBs in Oracle Database accessed through TimesTen, are exposed as TimesTen LOBs and are supported by TimesTen in much the same way that any TimesTen LOB is supported, but note the following:

- TimesTen LOB size limitations do not apply to storage of passthrough LOBs, but do apply to binding. Also, if a passthrough LOB is copied to a TimesTen LOB, such as through DBMS_LOB.COPY, the size limit applies to the copy.

  An attempt to copy a passthrough LOB to a TimesTen LOB when the passthrough LOB is larger than the TimesTen LOB size limit results in an error.

- As with TimesTen local LOBs, a locator for a passthrough LOB does not remain valid past the end of the transaction.

See "DBMS_LOB" on page 8-2 in this document for information about DBMS_LOB support for passthrough LOBs.
Use of cursors in PL/SQL programs

A cursor, either explicit or implicit, is used to handle the result set of a SELECT statement. As a programmer, you can declare an explicit cursor to manage queries that return multiple rows of data. PL/SQL declares and opens an implicit cursor for any SELECT statement that is not associated with an explicit cursor.

---

**Important:** Be aware that in TimesTen, any operation that ends your transaction closes all cursors associated with the connection. This includes any COMMIT or ROLLBACK statement. This also includes any DDL statement executed when PL/SQL is enabled, because the DDLCommitBehavior connection must be set to 0 (Oracle Database behavior) if PL/SQL is enabled. This results in autocommits of DDL statements. See "Differences in TimesTen: transaction behavior” on page 2-31 for additional information.

---

Example 2–13 shows basic use of a cursor. See "Examples using cursors" on page 5-6 for additional information and examples. Also see "PL/SQL REF CURSORS" on page 3-7.

**Example 2–13 Using a cursor to retrieve information about an employee**

Declare a cursor c1 to retrieve the last name, salary, hire date, and job class for the employee whose employee ID is 120.

```sql
Command> DECLARE
    > CURSOR c1 IS
    > SELECT last_name, salary, hire_date, job_id FROM employees
    > WHERE employee_id = 120;
    > --declare record variable that represents a row
    > --fetched from the employees table
    > \employee_rec c1%ROWTYPE;
    > BEGIN
    > -- open the explicit cursor
    > -- and use it to fetch data into employee_rec
    > OPEN c1;
    > FETCH c1 INTO employee_rec;
    > DBMS_OUTPUT.PUT_LINE('Employee name: ' || employee_rec.last_name);
    > CLOSE c1;
    > END;

Employee name: Weiss

PL/SQL procedure successfully completed.
```

---

**PL/SQL procedures and functions**

Procedures and functions are PL/SQL blocks that have been defined with a specified name.

**Creating and using procedures and functions**

Standalone subprograms (stored procedures or functions) are created at the database level with the CREATE PROCEDURE or CREATE FUNCTION statement.

Optionally use CREATE OR REPLACE PROCEDURE or CREATE OR REPLACE FUNCTION if you want the subprogram to be replaced if it already exists.
Use `ALTER PROCEDURE` or `ALTER FUNCTION` to explicitly compile a procedure or function or modify the compilation options. (To recompile a procedure or function that is part of a package, recompile the package using the `ALTER PACKAGE` statement.)

In TimesTen, syntax for `CREATE PROCEDURE` and `CREATE FUNCTION` is a subset of what is supported in Oracle Database. For information on these statements and the `ALTER PROCEDURE` and `ALTER FUNCTION` statements in TimesTen, see "SQL Statements" in Oracle TimesTen In-Memory Database SQL Reference.

Also see "How to execute PL/SQL procedures and functions" on page 2-8.

---

**Notes:**

- If you use replication: PL/SQL DDL statements, such as `CREATE` statements for PL/SQL functions, procedures, and packages, are not replicated. See "Creating a new PL/SQL object in an existing active standby pair" and "Adding a PL/SQL object to an existing replication scheme" in Oracle TimesTen In-Memory Database Replication Guide for steps to address this.

- If you use TimesTen Cache: A PL/SQL procedure or function resident in Oracle Database cannot be called in TimesTen by passthrough. Procedures and functions must be defined in TimesTen to be executable in TimesTen. (Also see "TimesTen PL/SQL with TimesTen Cache" on page 2-16.)

- PL/SQL and database object names: TimesTen does not support non-ASCII or quoted non-uppercase names of PL/SQL objects (procedures, functions, and packages). Also, trailing spaces in the quoted names of PL/SQL objects are not supported. In addition, trailing spaces in the quoted names of objects such as tables and views that are passed to PL/SQL are silently removed.

- Definer's rights or invoker's rights determines access to SQL objects used by a PL/SQL procedure or function. For information, refer to "Definer's rights and invoker's rights" on page 7-7.

- See "Showing errors in ttIsql" on page 4-6 for how to get information when you encounter errors in compiling a procedure or function.

---

**Example 2–14  Create and execute a procedure with OUT parameters**

This example creates a procedure that uses `OUT` parameters, executes the procedure in an anonymous block, then displays the `OUT` values. The procedure takes an employee ID as input then outputs the salary and job ID for the employee.

```
Command>  CREATE OR REPLACE PROCEDURE get_employee
  > (p_empid in employees.employee_id%TYPE,
  >   p_sal OUT employees.salary%TYPE,
  >   p_job OUT employees.job_id%TYPE) IS
  > BEGIN
  >   SELECT salary, job_id
  >   INTO p_sal, p_job
  >   FROM employees
  >   WHERE employee_id = p_empid;
  > END;
  > /
```
Procedure created.

Command> VARIABLE v_salary NUMBER;
Command> VARIABLE v_job VARCHAR2(15);
Command> BEGIN
  >   GET_EMPLOYEE (120, :v_salary, :v_job);
  >   END;
  > /

PL/SQL procedure successfully completed.

Command> PRINT
V_SALARY             : 8000
V_JOB                : ST_MAN

Command> SELECT salary, job_id FROM employees WHERE employee_id = 120;
< 8000, ST_MAN >
1 row found.

Note: Instead of using the anonymous block shown in the preceding example, you could use a CALL statement:

Command> CALL GET_EMPLOYEE(120, :v_salary, :v_job);

Example 2–15 Create and call a function

This example creates a function that returns the salary of the employee whose employee ID is specified as input, then calls the function and displays the result that was returned.

Command> CREATE OR REPLACE FUNCTION get_sal
  >   (p_id employees.employee_id%TYPE) RETURN NUMBER IS
  >     v_sal employees.salary%TYPE := 0;
  >     BEGIN
  >       SELECT salary INTO v_sal FROM employees
  >         WHERE employee_id = p_id;
  >       RETURN v_sal;
  >     END get_sal;
  > /

Function created.

Command> variable n number;
Command> call get_sal(100) into :n;
Command> print n;
N                    : 24000

Note: Instead of using the CALL statement shown in the preceding example, you could use an anonymous block:

Command> begin
  >   :n := get_sal(100);
  >   end;
  > /


Using synonyms for procedures and functions

TimesTen supports private and public synonyms (aliases) for database objects, including PL/SQL procedures, functions, and packages. Synonyms are often used to mask object names and object owners or to simplify SQL statements.

Create a private synonym for procedure foo in your schema as follows:

```sql
CREATE SYNONYM synfoo FOR foo;
```

Create a public synonym as follows:

```sql
CREATE PUBLIC SYNONYM pubfoo FOR foo;
```

A private synonym exists in the schema of a specific user and shares the same namespace as database objects such as tables, views, and sequences. A private synonym cannot have the same name as a table or other object in the same schema.

A public synonym does not belong to any particular schema, is accessible to all users, and can have the same name as any private object.

To use a synonym you must have appropriate privileges to access the underlying object. For required privileges to create or drop a synonym, see "Required privileges for PL/SQL statements and operations" on page 7-1.

For general information about synonyms, see "Understanding synonyms" in Oracle TimesTen In-Memory Database Operations Guide. For information about the CREATE SYNONYM and DROP SYNONYM statements, see "SQL Statements" in Oracle TimesTen In-Memory Database SQL Reference.

**Example 2–16  Use a synonym for a procedure**

In the following example, USER1 creates a procedure in his schema and creates a public synonym for it. Then USER2 executes the procedure through the public synonym.

Assume the following:

- **USER1** has been granted CREATE SESSION, CREATE PROCEDURE, and CREATE PUBLIC SYNONYM privileges.
- **USER2** has been granted CREATE SESSION and EXECUTE ANY PROCEDURE privileges.
- Both users have connected to the database.
- **USER2** employs the SET SERVEROUTPUT ON setting.

**USER1:**

```sql
Command> create or replace procedure test is
    > begin
    >   dbms_output.put_line('Running the test');
    > end;
    > /

Procedure created.

Command> create public synonym pubtest for test;

Synonym created.
```

**USER2:**

```sql
Command> begin
    > pubtest;
    > end;
```
PL/SQL packages

This section discusses how to create and use PL/SQL packages.

For information about PL/SQL packages provided with TimesTen, refer to Chapter 8, "TimesTen Supplied PL/SQL Packages."

Package concepts

A package is a database object that groups logically related PL/SQL types, variables, and subprograms. You specify the package and then define its body in separate steps.

The package specification is the interface to the package, declaring the public types, variables, constants, exceptions, cursors, and subprograms that are visible outside the immediate scope of the package. The body defines the objects declared in the specification, as well as queries for the cursors, code for the subprograms, and private objects that are not visible to applications outside the package.

TimesTen stores the package specification separately from the package body in the database. Other schema objects that call or reference public program objects depend only on the package specification, not on the package body.

---

**Note:** The syntax for creating packages and package bodies is the same as in Oracle Database; however, while Oracle Database documentation mentions that you must run a script named `DBMSSTDPL.SQL`, this does not apply to TimesTen.

---

Creating and using packages

To create packages and store them permanently in the database, use the `CREATE PACKAGE` and `CREATE PACKAGE BODY` statements.

To create a new package, do the following:

1. Create the package specification with the `CREATE PACKAGE` statement.
   
   You can declare program objects in the package specification. Such objects are referred to as public objects and can be referenced outside the package, and by other objects in the package.
   
   Optionally use `CREATE OR REPLACE PACKAGE` if you want the package specification to be replaced if it already exists.
   
2. Create the package body with the `CREATE PACKAGE BODY` (or `CREATE OR REPLACE PACKAGE BODY`) statement.

   You can declare and define program objects in the package body.

   - You must define public objects declared in the package specification.
   - You can declare and define additional package objects, referred to as private objects. Private objects are declared in the package body rather than in the package specification, so they can be referenced only by other objects in the package. They cannot be referenced outside the package.
Use **ALTER PACKAGE** to explicitly compile the member procedures and functions of a package or modify the compilation options.

For more information on the **CREATE PACKAGE**, **CREATE PACKAGE BODY**, and **ALTER PACKAGE** statements, see "SQL Statements" in *Oracle TimesTen In-Memory Database SQL Reference*.

---

**Notes:**

- If you use replication: PL/SQL DDL statements, such as **CREATE** statements for PL/SQL functions, procedures, and packages, are not replicated. See "Creating a new PL/SQL object in an existing active standby pair" and "Adding a PL/SQL object to an existing replication scheme" in *Oracle TimesTen In-Memory Database Replication Guide* for steps to address this.

- See "Showing errors in ttisql" on page 4-6 for how to get information when you encounter errors in compiling a package.

---

**Example 2–17  Create and use a package**

Consider the case where you want to add a row to the employees tables when you hire a new employee and delete a row from the employees table when an employee leaves your company. The following example creates two procedures to accomplish these tasks and bundles the procedures in a package. The package also contains a function to return the count of employees with a salary greater than that of a specific employee. The example then executes the function and procedures and verifies the results.

```
Command> CREATE OR REPLACE PACKAGE emp_actions AS
    > PROCEDURE hire_employee (employee_id NUMBER,
    >   last_name VARCHAR2,
    >   first_name VARCHAR2,
    >   email VARCHAR2,
    >   phone_number VARCHAR2,
    >   hire_date DATE,
    >   job_id VARCHAR2,
    >   salary NUMBER,
    >   commission_pct NUMBER,
    >   manager_id NUMBER,
    >   department_id NUMBER);
    > PROCEDURE remove_employee (emp_id NUMBER);
    > FUNCTION num_above_salary (emp_id NUMBER) RETURN NUMBER;
    > END emp_actions;
    > /

Package created.
```

```
Command> -- Package body:
    > CREATE OR REPLACE PACKAGE BODY emp_actions AS
    > -- Code for procedure hire_employee:
    >   PROCEDURE hire_employee (employee_id NUMBER,
    >     last_name VARCHAR2,
    >     first_name VARCHAR2,
    >     email VARCHAR2,
    >     phone_number VARCHAR2,
    >     hire_date DATE,
    >     job_id VARCHAR2,
    >     salary NUMBER,
    >     commission_pct NUMBER,
```
PL/SQL packages

Programming Features in PL/SQL in TimesTen

```plsql
CREATE OR REPLACE PACKAGE emp_actions IS

  -- Code for procedure hire_employee:
  PROCEDURE hire_employee (
    employee_id NUMBER,
    last_name VARCHAR2,
    first_name VARCHAR2,
    email VARCHAR2,
    phone_number VARCHAR2,
    hire_date DATE,
    job_id NUMBER,
    salary NUMBER,
    commission_pct NUMBER,
    manager_id NUMBER,
    department_id NUMBER) IS
  BEGIN
    INSERT INTO employees VALUES (employee_id,
      last_name,
      first_name,
      email,
      phone_number,
      hire_date,
      job_id,
      salary,
      commission_pct,
      manager_id,
      department_id);
  END hire_employee;
  -- Code for procedure remove_employee:
  PROCEDURE remove_employee (emp_id NUMBER) IS
  BEGIN
    DELETE FROM employees WHERE employee_id = emp_id;
  END remove_employee;
  -- Code for function num_above_salary:
  FUNCTION num_above_salary (emp_id NUMBER) RETURN NUMBER IS
   emp_sal NUMBER(8,2);
   num_count NUMBER;
  BEGIN
    SELECT salary INTO emp_sal FROM employees
    WHERE employee_id = emp_id;
    SELECT COUNT(*) INTO num_count FROM employees
    WHERE salary > emp_sal;
    RETURN num_count;
  END num_above_salary;
END emp_actions;
/

Package body created.

Command> BEGIN
  /* call function to return count of employees with salary
     greater than salary of employee with employee_id = 120
  */
  DBMS_OUTPUT.PUT_LINE ('Number of employees with higher salary: ' ||
    TO_CHAR(emp_actions.num_above_salary(120)));
END;

Number of employees with higher salary: 33

PL/SQL procedure successfully completed.

Verify the count of 33.

Command> SELECT salary FROM employees WHERE employee_id = 120;
< 8000 >
1 row found.

Command> SELECT COUNT(*) FROM employees WHERE salary > 8000;
< 33 >
1 row found.
```
Now add an employee and verify results. Then, remove the employee and verify that the employee was deleted from the employees table.

Command> BEGIN  
  > emp_actions.hire_employee(300,  
  > 'Belden',  
  > 'Enrique',  
  > 'EBELDEN',  
  > '555.111.2222',  
  > '31-AUG-04',  
  > 'AC_MGR',  
  > 9000,  
  > .1,  
  > 101,  
  > 110);  
  > END;  
  > /

PL/SQL procedure successfully completed.

Command> SELECT * FROM employees WHERE employee_id = 300;
< 300, Belden, Enrique, EBELDEN, 555.111.2222, 2004-08-31 00:00:00, AC_MGR, 9000, .1, 101, 110 >
1 row found.

Command> BEGIN  
  > emp_actions.remove_employee (300);  
  > END;  
  > /

PL/SQL procedure successfully completed.

Command> SELECT * FROM employees WHERE employee_id = 300;
0 rows found.

Using synonyms for packages

TimesTen supports private and public synonyms (aliases) for database objects, including PL/SQL procedures, functions, and packages. Synonyms are often used to mask object names and object owners or to simplify SQL statements.

To create a private synonym for package foopkg in your schema:

CREATE SYNONYM synfoopkg FOR foopkg;

To create a public synonym for foopkg:

CREATE PUBLIC SYNONYM pubfoopkg FOR foopkg;

Also see "Using synonyms for procedures and functions" on page 2-24 and "Required privileges for PL/SQL statements and operations" on page 7-1.
Wrapping PL/SQL source code

Wrapping is the process of hiding PL/SQL source code. You can wrap PL/SQL source code with the wrap utility, which processes an input SQL file and wraps only the PL/SQL units in the file, such as a package specifications, package bodies, functions, and procedures.

Consider the following example, which uses a file wrap_test.sql to define a procedure named wraptest. It then uses the wrap utility to process wrap_test.sql. The procedure is created with the source code hidden, and executes successfully. As a final step, the ALL_OBJECTS view is queried to see the wrapped source code.

Here are the contents of wrap_test.sql:

```sql
CREATE OR REPLACE PROCEDURE wraptest IS
    TYPE emp_tab IS TABLE OF employees%ROWTYPE INDEX BY PLS_INTEGER;
    all_emps emp_tab;
BEGIN
    SELECT * BULK COLLECT INTO all_emps FROM employees;
    FOR i IN 1..10
        LOOP
            DBMS_OUTPUT.PUT_LINE('Emp Id: ' || all_emps(i).employee_id);
        END LOOP;
END;
/
```

In the example that follows, "$" is the UNIX prompt, "Command>" is the ttIsql prompt, and user input is shown in bold.

```
$ wrap iname=wrap_test.sql

PL/SQL Wrapper: Release 11.2.0.2.0- Production on Wed Sep 14 12:59:27 2011
Copyright (c) 1993, 2009, Oracle. All rights reserved.

Processing wrap_test.sql to wrap_test.plb

$ cat wrap_test.plb
CREATE OR REPLACE PROCEDURE wraptest wrapped
a000000
1
abcd
abcd
abcd
abcd
abcd
abcd
abcd
abcd
```

Note: You cannot create synonyms for individual member subprograms of a package.

This is valid:

```sql
create or replace public synonym pubtestpkg for testpkg;
```

This is not valid:

```sql
create or replace public synonym pubtestproc for testpkg.testproc;
```
Wrapping PL/SQL source code

```sql
CREATE OR REPLACE PROCEDURE wraptest
  wrapped
  a000000
  1
  abcd
  abcd
  abcd
  abcd
  abcd
  abcd
  7
  124 12c
Y26L0v2ntFagtW8h5JD5IHYYccwg+nvWfZqfHQCv/9kMfYzmvLh8FepNXpNS1fzVBDkTke
L91hPzCfmmJ5GGrCwrrqngEhfRpg7ck5Dzsf7sD1nQeE3QGb/yu9Dec1+J02kOMLx3dq
BuC7f2f5jsDteBx1GQC0kJ5QBVregtoBck2NO9Mo1NS4w0jF6T1CPY0Aoi/KhooC8S818n
amP5xQDCTyDajs77or1GeqtX747k0YOa+r1e9adGUusVgZK10NcTM/+Wit+LYK17b03eJxdB
+aKn/Lh
/
$ ttisql sampledb_1122
Copyright (c) 1996-2011, Oracle. All rights reserved.
Type ? or "help" for help, type 'exit' to quit ttisql.

connect "DSN=sampledb_1122";
Connection successful:
DSN=sampledb_1122;UID=myuserid;DataStore=.../install/info/DemoDataStore/
sampledb_1122;DatabaseCharacterSet=US7ASCII;ConnectionCharacterSet=US7ASCII;DRIVER
=.../install/lib/libtten.so;PermSize=40;TempSize=32;TypeMode=0;
(Default setting AutoCommit=1)

Command> @wrap_test.plb

CREATE OR REPLACE PROCEDURE wraptest
  wrapped
  a000000
  1
  abcd
  abcd
  abcd
  abcd
  abcd
  abcd
  abcd
  abcd
  abcd
  abcd
  abcd
  abcd
  abcd
  7
  124 12c
Y26L0v2ntFagtW8h5JD5IHYYccwg+nvWfZqfHQCv/9kMfYzmvLh8FepNXpNS1fzVBDkTke
L91hPzCfmmJ5GGrCwrrqngEhfRpg7ck5Dzsf7sD1nQeE3QGb/yu9Dec1+J02kOMLx3dq
BuC7f2f5jsDteBx1GQC0kJ5QBVregtoBck2NO9Mo1NS4w0jF6T1CPY0Aoi/KhooC8S818n
amP5xQDCTyDajs77or1GeqtX747k0YOa+r1e9adGUusVgZK10NcTM/+Wit+LYK17b03eJxdB
+aKn/Lh

Procedure created.

Command> SET SERVEROUTPUT ON
Differences in TimesTen: transaction behavior

In TimesTen, any operation that ends your transaction closes all cursors associated with the connection. This includes the following:

- Any COMMIT or ROLLBACK statement
- For PL/SQL users, any DDL statement

This is because when PL/SQL is enabled (the PLSQL first connection attribute is set to 1), the TimesTen DDLCommitBehavior general connection attribute must be set to 0 for Oracle Database behavior (autocommit DDL).

PL/SQL procedure successfully completed.

BEGIN
  wraptest();
  
END;
/

Emp Id: 100
Emp Id: 101
Emp Id: 102
Emp Id: 103
Emp Id: 104
Emp Id: 105
Emp Id: 106
Emp Id: 107
Emp Id: 108
Emp Id: 109

Command> SELECT text FROM all_source WHERE name = 'WRAPTEST';

< PROCEDURE wraptest wrapped
a000000
  1
  abcd
  abcd
  abcd
  abcd
  abcd
  abcd
  abcd
  abcd
  abcd
  abcd
  abcd
  abcd
  abcd
  abcd
  abcd
  7
  124 12c
Y26L0v2ntFaqtw8hSJD5IHYccwq+mwNf2zfHQCv/9kJMyzmdLh8FepNXpWS1fzVBDkTke
Lw1hDbfzCMfmmJ5GGrCwrgmgfEfgPq7ck5Dzsf7sDlnQeB3Qgm3/yu9Dec1+J02kOMlx3dq
Bu7f2f5sjDctBeDiGCC0kJ5QBVregtoBckZN09MoWiW4w0jF6T1CPY0AoI/KUwxC8S8I8n
amF5xQDQYTDaajs77orIGEqtX747k0YA0+r1e9adGUsVg2K10NcTM/+tLYKi7b03eJzdB
+aKn/Lh

1 row found.
For example, consider the following scenario, where you want to recompile a set of procedures. This would not work, because the first time ALTER PROCEDURE is executed, the cursor (pnamecurs) would be closed:

```sql
declare
cursor pnamecurs is select * from all_objects where object_name like 'MYPROC%';
begin
  for rec in pnamecurs loop
    execute immediate 'alter procedure ' || rec.object_name || ' compile';
  end loop;
end;
```

Instead, you can do something like the following, which fetches all the procedure names into an internal table then executes ALTER PROCEDURE on them with no active cursor.

```sql
declare
cursor pnamecurs is select * from all_objects where object_name like 'MYPROC%';
type tbl is table of c%rowtype index by binary_integer;
myprocs tbl;
begin
  open pnamecurs;
  fetch pnamecurs bulk collect into myprocs;
  close pnamecurs;
  for i in 1..myprocs.count loop
    execute immediate 'alter procedure ' || myprocs(i).object_name || ' compile';
  end loop;
end;
```
This chapter focuses on the range of data types available to you for manipulating data in PL/SQL, TimesTen SQL, and your application programs.

TimesTen supports PL/SQL data types and the interactions between PL/SQL data types, TimesTen data types, and client application program data types. Data type conversions and data type mappings are supported.

See the end of the chapter for TimesTen-specific considerations.

Topics in this chapter include the following:

- Understanding the data type environments
- Understanding and using PL/SQL data types
- Data type conversion
- Differences in TimesTen: data type considerations

Understanding the data type environments

There are three distinct environments to consider when discussing data types:

- PL/SQL programs that contain variables and constants that use PL/SQL data types
- TimesTen SQL statements that make use of database rows, columns, and constants. These elements are expressed using TimesTen SQL data types.
- Application programs that interact with the database and the PL/SQL programming language. Application programs are written in programming languages such as C and Java and contain variables and constants that use data types from these programming languages.

Table 3–1 summarizes the environments and gives examples of data types for each environment.

<table>
<thead>
<tr>
<th>Environment</th>
<th>Data type examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/SQL programs</td>
<td>NUMBER, PLS_INTEGER, VARCHAR2, STRING, DATE, TIMESTAMP</td>
</tr>
<tr>
<td>TimesTen SQL statements</td>
<td>TT_BIGINT, TT_INTEGER, BINARY_FLOAT, VARCHAR2, DATE, TIMESTAMP</td>
</tr>
<tr>
<td>Application programs</td>
<td>int, double, String</td>
</tr>
</tbody>
</table>
Understanding and using PL/SQL data types

This section describes the PL/SQL data types that are supported in PL/SQL programs. It does not describe the data types supported in TimesTen SQL statements. For information on data types supported in TimesTen SQL statements, see "Data Types" in Oracle TimesTen In-Memory Database SQL Reference.

The following topics are covered in this section:
- PL/SQL data type categories
- Predefined PL/SQL scalar data types
- PL/SQL composite data types
- PL/SQL REF CURSORs

For additional information see "PL/SQL Data Types" in Oracle Database PL/SQL Language Reference.

### PL/SQL data type categories

In a PL/SQL block, every constant, variable, and parameter has a data type. PL/SQL provides predefined data types and subtypes and lets you define your own PL/SQL subtypes.

Table 3–2 lists the categories of the predefined PL/SQL data types.

<table>
<thead>
<tr>
<th>Data type category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar</td>
<td>Single values with no internal components</td>
</tr>
<tr>
<td>Composite</td>
<td>Internal components that are either scalar or composite</td>
</tr>
<tr>
<td>Reference</td>
<td>Pointers to other data items such as REF CURSORs</td>
</tr>
</tbody>
</table>

**Note:** See “Non-supported data types” on page 3-13.

### Predefined PL/SQL scalar data types

Scalar data types store single values with no internal components. Table 3–3 lists predefined PL/SQL scalar data types of interest, grouped by data type families.

<table>
<thead>
<tr>
<th>Data type family</th>
<th>Data type name</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMERIC</td>
<td>NUMBER</td>
</tr>
<tr>
<td></td>
<td>PLS_INTEGER</td>
</tr>
<tr>
<td></td>
<td>BINARY_FLOAT</td>
</tr>
<tr>
<td></td>
<td>BINARY_DOUBLE</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>CHAR[ACTER]</td>
</tr>
<tr>
<td></td>
<td>VARCHAR2</td>
</tr>
<tr>
<td></td>
<td>NVARCHAR (national character CHAR)</td>
</tr>
<tr>
<td></td>
<td>NVARCHAR2 (national character VARCHAR2)</td>
</tr>
<tr>
<td>BINARY</td>
<td>RAW</td>
</tr>
</tbody>
</table>
Example 3–1 Declaring PL/SQL variables

Command> DECLARE
   > v_emp_job      VARCHAR2 (9);
   > v_count_loop   BINARY_INTEGER := 0;
   > v_dept_total_sal NUMBER (9,2) := 0;
   > v_orderdate    DATE := SYSDATE + 7;
   > v_valid        BOOLEAN NOT NULL := TRUE;
   > ...

PLS_INTEGER and BINARY_INTEGER data types

The PLS_INTEGER and BINARY_INTEGER data types are identical and are used interchangeably in this document.

The PLS_INTEGER data type stores signed integers in the range -2,147,483,648 through 2,147,483,647 represented in 32 bits. It has the following advantages over the NUMBER data type and subtypes:

- PLS_INTEGER values require less storage.
- PLS_INTEGER operations use hardware arithmetic, so they are faster than NUMBER operations, which use library arithmetic.

For efficiency, use PLS_INTEGER values for all calculations that fall within its range. For calculations outside the PLS_INTEGER range, use INTEGER, a predefined subtype of the NUMBER data type.

See "PLS_INTEGER and BINARY_INTEGER Data Types" in Oracle Database PL/SQL Language Reference for additional information.

Note: When a calculation with two PLS_INTEGER data types overflows the PLS_INTEGER range, an overflow exception is raised even if the result is assigned to a NUMBER data type.
**SIMPLE_INTEGER data type**

SIMPLE_INTEGER is a predefined subtype of the PL_S_INTEGER data type that has the same range as PL_S_INTEGER (-2,147,483,648 through 2,147,483,647) and has a NOT NULL constraint. It differs from PL_S_INTEGER in that it does not overflow.

You can use SIMPLE_INTEGER when the value is never null and overflow checking is unnecessary. Without the overhead of checking for null values and overflow, SIMPLE_INTEGER provides better performance than PL_S_INTEGER.

See "SIMPLE_INTEGER Subtype of PL_S_INTEGER" in Oracle Database PL/SQL Language Reference for additional information.

**ROWID data type**

Each row in a table has a unique identifier known as its rowid.

An application can specify literal rowid values in SQL statements, such as in WHERE clauses, as CHAR constants enclosed in single quotes.

Also refer to "ROWID data type" and "ROWID" in Oracle TimesTen In-Memory Database SQL Reference for additional information about rowids and the ROWID data type, including usage and life.

**LOB data types**

The LOB (large object) type family includes CLOB (character LOBs), NCLOB (national character LOBs), and BLOB (binary LOBs).

A LOB consists of a LOB locator and a LOB value. The locator acts as a handle to the value. When an application selects a LOB or passes a LOB as a parameter, for example, it is using the locator, not the actual value.

LOBs may be either persistent or temporary. A persistent LOB exists in the database, in a particular row of a LOB column. A temporary LOB is used internally within a program, but could then be inserted into a LOB column in the database to become a persistent LOB.

See "LOB data types" in Oracle TimesTen In-Memory Database SQL Reference for additional information about LOBs in TimesTen.

Also see "Large objects (LOBs)" on page 2-17 for usage information.

**PL/SQL composite data types**

Composite types have internal components that can be manipulated individually, such as the elements of an array, record, or table.

Oracle TimesTen In-Memory Database supports the following composite data types:

- Associative array (index-by table)
- Nested table
- Varray
- Record

Associative arrays, nested tables, and varrays are also referred to as collections.

The following sections discuss the use of composite data types:

- Using collections in PL/SQL
- Using records in PL/SQL
Understanding and using PL/SQL data types

Using associative arrays from applications

See "PL/SQL Collections and Records" in Oracle Database PL/SQL Language Reference for additional information.

Using collections in PL/SQL

You can declare collection data types similar to arrays, sets, and hash tables found in other languages. A collection is an ordered group of elements, all of the same type. Each element has a unique subscript that determines its position in the collection.

In PL/SQL, array types are known as varrays (variable size arrays), set types are known as nested tables, and hash table types are known as associative arrays or index-by tables. These are all collection types.

**Example 3–2 Using a PL/SQL collection type**

This example declares collection type `staff_list` as a table of `employee_id`, then uses the collection type in a loop and in the `WHERE` clause of the `SELECT` statement.

```plsql
Command> DECLARE
    > TYPE staff_list IS TABLE OF employees.employee_id%TYPE;
    > staff staff_list;
    > lname employees.last_name%TYPE;
    > fname employees.first_name%TYPE;
    > BEGIN
    >     staff := staff_list(100, 114, 115, 120, 122);
    >     FOR i IN staff.FIRST..staff.LAST LOOP
    >         SELECT last_name, first_name INTO lname, fname FROM employees
    >             WHERE employees.employee_id = staff(i);
    >         DBMS_OUTPUT.PUT_LINE (TO_CHAR(staff(i)) ||
    >             ': ' || lname || ', ' || fname);
    >     END LOOP;
    > END;
    > /
100: King, Steven
114: Raphaely, Den
115: Khoo, Alexander
120: Weiss, Matthew
122: Kaufling, Payam

PL/SQL procedure successfully completed.
```

Any collections can be passed between PL/SQL subprograms as parameters, but in TimesTen only associative arrays can be passed between PL/SQL and applications written in other languages. (See "Using associative arrays from applications" below.)

You can use collections to move data in and out of TimesTen tables using bulk SQL.

Using records in PL/SQL

Records are composite data structures that have fields with different data types. You can pass records to subprograms with a single parameter. You can also use the %ROWTYPE attribute to declare a record that represents a row in a table or a row from a query result set, without specifying the names and types for the fields, as shown in Example 2–2 on page 2-3.

**Example 3–3 Declaring a record type**

Declare various record types.
Command> DECLARE
>   TYPE timerec IS RECORD (hours SMALLINT, minutes SMALLINT);
>   TYPE meetin_typ IS RECORD {
>     date_held DATE,
>     duration timerec, -- nested record
>     location VARCHAR2(20),
>     purpose VARCHAR2(50)));
> BEGIN
>   ...
> END;
> /

Using associative arrays from applications

Associative arrays, formerly known as index-by tables or PL/SQL tables, are supported as IN, OUT, or IN OUT bind parameters in TimesTen PL/SQL, such as from an OCI, Pro*C/C++, or JDBC application. This enables arrays of data to be passed efficiently between an application and the database.

An associative array is a set of key-value pairs. In TimesTen, for associative array binding (but not for use of associative arrays only within PL/SQL), the keys, or indexes, must be integers (BINARY_INTEGER or PLS_INTEGER). The values must be simple scalar values of the same data type. For example, there could be an array of department managers indexed by department numbers. Indexes are stored in sort order, not creation order.

You can declare an associative array type and then an associative array in PL/SQL as in the following example (note the INDEX BY):

```
define
   TYPE VARCHARARRTYP IS TABLE OF VARCHAR2(30) INDEX BY BINARY_INTEGER;
   x VARCHARARRTYP;
...
```

See Example 3–4 below and "Using associative arrays" on page 5-5 for examples.

Also see "Associative array bindings in TimesTen OCI" and "Associative array bindings in TimesTen Pro*C/C++" in Oracle TimesTen In-Memory Database C Developer’s Guide, and "Binding associative arrays" in Oracle TimesTen In-Memory Database Java Developer’s Guide.

For general information about associative arrays, see "Associative Arrays" in Oracle Database PL/SQL Language Reference.

Notes: Note the following restrictions in TimesTen:

- The following types are not supported in binding associative arrays: LOBs, REF CURSORs, TIMESTAMP, ROWID.
- Associative array binding is not allowed in passthrough statements.
- General bulk binding of arrays is not supported in TimesTen programmatic APIs. Varrays and nested tables are not supported as bind parameters.

The following example does some simple manipulation of an associative array, effectively binding it from ttIsql and printing the array.
Example 3–4 Using an associative array from ttisql

Command> var lngvc[1000] varchar2(30);
Command> declare
> TYPE VARCHARARRTYP IS TABLE OF VARCHAR2(30) INDEX BY BINARY_INTEGER;
> x VARCHARARRTYP;
> begin
>   x := :lngvc;
>   x ( 1 ) := 'One';
>   x ( 10 ) := 'Ten';
>   :lngvc := x;
> end;
> /

PL/SQL procedure successfully completed.

Command> print lngvc;
LNGVC : ARRAY [ 1000 ] (Current Size 10)
LNGVC[1] : One
LNGVC[7] : <NULL>
LNGVC[8] : <NULL>
LNGVC[9] : <NULL>
LNGVC[10] : Ten

PL/SQL REF CURSORS

A REF CURSOR is a handle to a cursor over a SQL result set that can be passed as a parameter between PL/SQL and an application. TimesTen supports OUT REF CURSORS, from PL/SQL to the application. The application would open the REF CURSOR within PL/SQL, pass it from there through the applicable API, and fetch the result set.

TimesTen supports REF CURSORS in ODBC, JDBC, ODP.NET, OCI, Pro*C/C++, and TTClasses for either direct connections or client/server connections. REF CURSORS are also discussed in the following TimesTen documents:

- "Working with REF CURSORS" in Oracle TimesTen In-Memory Database C Developer’s Guide
- "Working with REF CURSORS" in Oracle TimesTen In-Memory Database Java Developer’s Guide
- "Working with REF CURSORS" in Oracle TimesTen In-Memory Database TTClasses Guide

Note: TimesTen supports one OUT REF CURSOR per statement.

You can define a REF CURSOR in PL/SQL in TimesTen as you would in Oracle Database. (See "Cursor Variables" in Oracle Database PL/SQL Language Reference.) It is typical to use REF CURSOR as a metatype, where you define a "strong" (specific) REF CURSOR type tailored to your data, then declare a cursor variable of that type. For example:

Command> DECLARE
>   TYPE DeptCurTyp IS REF CURSOR RETURN departments%ROWTYPE;
Example 3–5 Use a REF CURSOR as an output parameter

This example creates a procedure GET_EMP in a package FOO_PACK to retrieve information about employees from the employees table. The procedure declares a REF CURSOR type cursor_out, then uses that type for the output parameter.

First specify the package definition, REF CURSOR type, and procedure definition.

```sql
create or replace package foo_pack is
type cursor_out is ref cursor;
procedure get_emp (results out cursor_out);
end foo_pack;
```

Then specify the package body and procedure implementation.

```sql
create or replace package body foo_pack as
  procedure get_emp (results out cursor_out) is
    begin
      open results for select employee_id, last_name from employees
        where employee_id < 110 order by last_name;
    end get_emp;
  end foo_pack;
```

Declare a REF CURSOR variable for the output, execute the procedure, and display the results. Note that outside of PL/SQL, you can declare only “weak” (generic) REF CURSORS.

```sql
Command> var proc_result refcursor;
Command> exec foo_pack.get_emp(:proc_result);
PL/SQL procedure successfully completed.

Command> print proc_result;
PROC_RESULT          :
< 105, Austin >
< 102, De Haan >
< 104, Ernst >
< 109, Faviet >
< 108, Greenberg >
< 103, Hunold >
< 100, King >
< 101, Kochhar >
< 107, Lorentz >
< 106, Pataballa >
10 rows found.
```

Alternatively, you could declare a weakly typed REF CURSOR variable in FOO_PACK:

```sql
create or replace package foo_pack is
  procedure get_emp (results out sys_refcursor);
end foo_pack;

create or replace package body foo_pack as
  procedure get_emp (results out sys_refcursor) is
    begin
      open results for select employee_id, last_name from employees
        where employee_id < 110 order by last_name;
    end get_emp;
  end foo_pack;
```
Data type conversion

This section covers the following data type conversions:

- Conversion between PL/SQL data types
- Conversion between application data types and PL/SQL or SQL data types

Also see type conversion information under "Differences in TimesTen: data type considerations" on page 3-10.

Conversion between PL/SQL data types

TimesTen supports implicit and explicit conversions between PL/SQL data types.

Consider this example: The variable v_sal_hike is of type VARCHAR2. When calculating the total salary, PL/SQL first converts v_sal_hike to NUMBER then performs the operation. The result is of type NUMBER. PL/SQL uses implicit conversion to obtain the correct result.

Command> DECLARE
  >   v_salary NUMBER (6) := 6000;
  >   v_sal_hike VARCHAR2(5) := '1000';
  >   v_total_salary v_salary%TYPE;
  > BEGIN
  > v_total_salary := v_salary + v_sal_hike;
  > DBMS_OUTPUT.PUT_LINE (v_total_salary);
  > end;
  > /

7000

PL/SQL procedure successfully completed.

Conversion between application data types and PL/SQL or SQL data types

TimesTen supports data type conversions between application program data types and PL/SQL data types, and between application program data types and TimesTen SQL data types. For SQL, the conversions are the same whether SQL is invoked by your PL/SQL program or is invoked directly by your application.

As an example, Table 3–4 shows a few representative data type mappings from an application using the ODBC API to PL/SQL program data types. For more information about ODBC-to-PL/SQL type mappings, refer to "Determination of parameter type assignments and type conversions" in Oracle TimesTen In-Memory Database C Developer’s Guide.

Table 3–4  Sampling of ODBC SQL to PL/SQL type mapping

<table>
<thead>
<tr>
<th>ODBC type</th>
<th>PL/SQL type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL_BINARY</td>
<td>RAW (Bound precision is used.)</td>
</tr>
<tr>
<td>SQL_CHAR</td>
<td>CHAR (Bound precision is used.)</td>
</tr>
<tr>
<td>SQL_DATE</td>
<td>DATE</td>
</tr>
<tr>
<td>SQL_DECIMAL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>SQL_DOUBLE</td>
<td>NUMBER</td>
</tr>
</tbody>
</table>
Differences in TimesTen: data type considerations

Example 3–6 ODBC to PL/SQL data type conversions
Consider a scenario where your C program uses the ODBC API and your goal is to bind your C variable of type VARCHAR2 to a PL/SQL variable of type NUMBER. TimesTen performs the implicit conversion for you.

```
Command> VARIABLE c_var VARCHAR2 (30) := '961';
Command> DECLARE v_var NUMBER;
  > BEGIN
  >   v_var := :c_var;
  >   DBMS_OUTPUT.PUT_LINE (v_var);
  > END;
  > /
961
PL/SQL procedure successfully completed.
```

Example 3–7 ODBC to TimesTen SQL data type conversions
This example creates a table with a column of type TT_BIGINT and uses PL/SQL to invoke the TimesTen SQL INSERT statement. A bind variable of type SQL_VARCHAR is used in the INSERT statement. The conversions are the same as the conversions that would occur if your application invoked the INSERT statement directly.

```
Command> CREATE TABLE conversion_test2 (Col1 TT_BIGINT);
Command>  VARIABLE v_var VARCHAR2 (100) := '1000';
Command>  BEGIN
    >   INSERT INTO conversion_test2 VALUES (:v_var);
  > END;
  > /
PL/SQL procedure successfully completed.
```

```
Command> SELECT * FROM conversion_test2;
< 1000 >
1 row found.
```

Differences in TimesTen: data type considerations
This section covers the following TimesTen-specific considerations regarding data type support and type conversions:

- Conversion between PL/SQL and TimesTen SQL data types
- Date and timestamp formats: NLS_DATE_FORMAT and NLS_TIMESTAMP_FORMAT
- Non-supported data types
- Zero-length strings in TimesTen type mode

<table>
<thead>
<tr>
<th>ODBC type</th>
<th>PL/SQL type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL_FLOAT</td>
<td>BINARY_DOUBLE</td>
</tr>
<tr>
<td>SQL_INTEGER</td>
<td>PLS_INTEGER</td>
</tr>
<tr>
<td>SQL_REFCURSOR</td>
<td>REF CURSOR</td>
</tr>
<tr>
<td>SQL_TIMESTAMP</td>
<td>TIMESTAMP (Bound scale is used.)</td>
</tr>
<tr>
<td>SQL_VARCHAR</td>
<td>VARCHAR2 (Bound precision is used.)</td>
</tr>
</tbody>
</table>

Table 3–4 (Cont.) Sampling of ODBC SQL to PL/SQL type mapping
Conversion between PL/SQL and TimesTen SQL data types

TimesTen supports conversions between PL/SQL data types and TimesTen SQL data types.

Table 3–5 shows supported data type conversions, with PL/SQL types along the top and SQL types down the left side. The data types are grouped by data type families, with columns referring to PL/SQL type families and rows referring to TimesTen type families. "Y" indicates that a conversion is possible between the two families. Supported conversions are bidirectional.

<table>
<thead>
<tr>
<th>Type Family</th>
<th>NUMERIC</th>
<th>CHARACTER</th>
<th>BINARY</th>
<th>DATETIME</th>
<th>INTERVAL</th>
<th>ROWID</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMERIC</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHARACTER</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>DATETIME</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>ROWID</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>BINARY</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
</tr>
</tbody>
</table>

Table 3–6 that follows summarizes the TimesTen data types and suggestions for PL/SQL type mappings.

<table>
<thead>
<tr>
<th>TimesTen data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TT_TINYINT</td>
<td>This is for unsigned integers ranging from 0 to 255. Numeric overflows can occur if you insert a value with type PL/SQL NUMBER or PL/SQL PLS_INTEGER (or BINARY_INTEGER) into a TT_TINYINT column.</td>
</tr>
<tr>
<td>TT_SMALLINT</td>
<td>This is for signed 16-bit integers in the range -32,768 to 32,767. Numeric overflows can occur if you insert a value with type PL/SQL NUMBER or PL/SQL PLS_INTEGER (or BINARY_INTEGER) into a TT_SMALLINT column.</td>
</tr>
<tr>
<td>TT_INTEGER</td>
<td>This is for signed integers in the range -2,147,483,648 to 2,147,483,647. This is equivalent to PLS_INTEGER.</td>
</tr>
<tr>
<td>TT_BIGINT</td>
<td>This is for signed eight-byte integers in the range -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. Use PL/SQL NUMBER. A PL/SQL PLS_INTEGER (or BINARY_INTEGER) variable could overflow.</td>
</tr>
<tr>
<td>NUMBER, BINARY_FLOAT, BINARY_DOUBLE</td>
<td>Use when floating point precision is required.</td>
</tr>
</tbody>
</table>

Character types

All PL/SQL character types can hold up to 32,767 bytes of data.

- TimesTen CHAR can hold up to 8300 bytes.
- TimesTen NCHAR can hold up to 4150 characters (8300 bytes).
- TimesTen VARCHAR2 can hold up to 4,194,304 bytes.
- TimesTen NVARCHAR2 can hold up to 2,097,152 characters (4,194,304 bytes).
Differences in TimesTen: data type considerations

Example 3–8 Conversions between TimesTen SQL data types and PL/SQL data types

Consider the case where you have a table with two columns. Col1 has a data type of TT_INTEGER and Col2 has a data type of NUMBER. In your PL/SQL program, you declare two variables: v_var1 of type PLS_INTEGER and v_var2 of type VARCHAR2. The goal is to SELECT the row of data from your table into the two PL/SQL variables.

Data type conversions occur when you execute the SELECT statement. Col1 is converted from a TimesTen SQL TT_INTEGER type into a PLS_INTEGER type. Col2 is converted from a TimesTen SQL NUMBER type into a PL/SQL VARCHAR2 type. The query executes successfully.

Command> CREATE TABLE test_conversion (Col1 TT_INTEGER, Col2 NUMBER);
Command> INSERT INTO test_conversion VALUES (100, 20);
1 row inserted.

Command> DECLARE
  >   v_var1 PLS_INTEGER;
  >   v_var2 VARCHAR2 (100);
  >   BEGIN
  >   SELECT Col1, Col2 INTO v_var1, v_var2 FROM test_conversion;
  >   DBMS_OUTPUT.PUT_LINE (v_var1);
  >   DBMS_OUTPUT.PUT_LINE (v_var2);
  >   END;
  > /
100
20

PL/SQL procedure successfully completed.

Date and timestamp formats: NLS_DATE_FORMAT and NLS_TIMESTAMP_FORMAT

TimesTen does not support user-specified NLS_DATE_FORMAT and NLS_TIMESTAMP_FORMAT settings.

- NLS_DATE_FORMAT is always 'yyyy-mm-dd'.
- NLS_TIMESTAMP_FORMAT is always 'yyyy-mm-dd hh:mi:ss.ff6' (fractional seconds to six decimal places).

You can use the SQL and PL/SQL TO_DATE and TO_CHAR functions to specify other desired formats. See "Expressions" in Oracle TimesTen In-Memory Database SQL Reference for details of these functions.
Non-supported data types

Note the following non-support of data types:

- **PL/SQL data type categories:** PL/SQL in TimesTen does not support Internet data types (XMLType, URIType, HttpURIType) or "Any" data types (AnyType, AnyData, AnyDataSet).

- **PL/SQL scalar data types:** TimesTen does not support the PL/SQL data types TIMESTAMP WITH [LOCAL] TIME ZONE and UROWID.

- TimesTen PL/SQL does not support the TimesTen type TT_DECIMAL.

Zero-length strings in TimesTen type mode

In TimesTen type mode, still supported for backward compatibility, the string value "" is considered by TimesTen to be an empty, zero-length string, but not a null value. In PL/SQL, however, a zero-length string is always considered to be null. If a parameter with a value of empty string is passed to a SQL statement that is executed in PL/SQL, the parameter is converted to NULL by PL/SQL before the SQL statement is passed to TimesTen.

See "TimesTen type mode (backward compatibility)" in Oracle TimesTen In-Memory Database SQL Reference for information about this mode.
Errors and Exception Handling

This chapter describes the flexible error trapping and error handling you can use in your PL/SQL programs.

For more information on error-handling and exceptions in PL/SQL, see "PL/SQL Error Handling" in Oracle Database PL/SQL Language Reference.

See the end of this chapter for TimesTen-specific considerations.

The following topics are covered:

- Understanding exceptions
- Trapping exceptions
- Showing errors in ttIsql
- Differences in TimesTen: exception handing and error behavior

Understanding exceptions

This section provides an overview of exceptions in PL/SQL programming, covering the following topics:

- About exceptions
- Exception types

About exceptions

An exception is a PL/SQL error that is raised during program execution, either implicitly by TimesTen or explicitly by your program. Handle an exception by trapping it with a handler or propagating it to the calling environment.

For example, if your SELECT statement returns multiple rows, TimesTen returns an error (exception) at runtime. As the following example shows, you would see TimesTen error 8507, then the associated ORA error message. (ORA messages, originally defined for Oracle Database, are similarly implemented by TimesTen.)

```
Command> DECLARE
  >   v_lname VARCHAR2 (15);
  > BEGIN
  >   SELECT last_name INTO v_lname
  >   FROM employees
  >   WHERE first_name = 'John';
  >   DBMS_OUTPUT.PUT_LINE ('Last name is:' || v_lname);
  > END;
  > /
```
You can handle such exceptions in your PL/SQL block so that your program completes successfully. For example:

```
Command> DECLARE
    > v_lname VARCHAR2 (15);
    > BEGIN
    >   SELECT last_name INTO v_lname
    >     FROM employees
    >     WHERE first_name = 'John';
    >   DBMS_OUTPUT.PUT_LINE ('Last name is :' || v_lname);
    > EXCEPTION
    >   WHEN TOO_MANY_ROWS THEN
    >   DBMS_OUTPUT.PUT_LINE ('Your SELECT statement retrieved multiple rows. Consider using a cursor.');
    > END;
    > /
```

Your SELECT statement retrieved multiple rows. Consider using a cursor.

PL/SQL procedure successfully completed.

### Exception types

There are three types of exceptions:

- **Predefined exceptions** are error conditions that are defined by PL/SQL.
- **Non-predefined exceptions** include any standard TimesTen errors.
- **User-defined exceptions** are exceptions specific to your application.

In TimesTen, these three types of exceptions are used in the same way as in Oracle Database.

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
<th>How to handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predefined TimesTen error</td>
<td>One of approximately 20 errors that occur most often in PL/SQL code</td>
<td>You are not required to declare these exceptions. They are predefined by TimesTen. TimesTen implicitly raises the error.</td>
</tr>
<tr>
<td>Non-predefined TimesTen error</td>
<td>Any other standard TimesTen error</td>
<td>These must be declared in the declarative section of your application. TimesTen implicitly raises the error and you can use an exception handler to catch the error.</td>
</tr>
<tr>
<td>User-defined error</td>
<td>Error defined and raised by the application</td>
<td>These must be declared in the declarative section. The developer raises the exception explicitly.</td>
</tr>
</tbody>
</table>

### Trapping exceptions

This section describes how to trap predefined TimesTen errors or user-defined errors.
Trapping exceptions

Trapping predefined TimesTen errors

Trap a predefined TimesTen error by referencing its predefined name in your exception-handling routine. PL/SQL declares predefined exceptions in the STANDARD package.

Table 4–1 lists predefined exceptions supported by TimesTen, the associated ORA error numbers and SQLCODE values, and descriptions of the exceptions.

Also see "Unsupported predefined errors" on page 4-9.

Table 4–1  Predefined exceptions

<table>
<thead>
<tr>
<th>Exception name</th>
<th>Oracle Database error number</th>
<th>SQLCODE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS INTO NULL</td>
<td>ORA-06530</td>
<td>-6530</td>
<td>Program attempted to assign values to the attributes of an uninitialized object.</td>
</tr>
<tr>
<td>CASE NOT FOUND</td>
<td>ORA-06592</td>
<td>-6592</td>
<td>None of the choices in the WHEN clauses of a CASE statement were selected and there is no ELSE clause.</td>
</tr>
<tr>
<td>COLLECTION IS NULL</td>
<td>ORA-06531</td>
<td>-6531</td>
<td>Program attempted to apply collection methods other than EXISTS to an uninitialized nested table or varray, or program attempted to assign values to the elements of an uninitialized nested table or varray.</td>
</tr>
<tr>
<td>CURSOR ALREADY OPENED</td>
<td>ORA-06511</td>
<td>-6511</td>
<td>Program attempted to open an already opened cursor.</td>
</tr>
<tr>
<td>DUP VAL ON INDEX</td>
<td>ORA-00001</td>
<td>-1</td>
<td>Program attempted to insert duplicate values in a column that is constrained by a unique index.</td>
</tr>
<tr>
<td>INVALID_CURSOR</td>
<td>ORA-01001</td>
<td>-1001</td>
<td>There is an illegal cursor operation.</td>
</tr>
<tr>
<td>INVALID NUMBER</td>
<td>ORA-01722</td>
<td>-1722</td>
<td>Conversion of character string to number failed.</td>
</tr>
<tr>
<td>NO DATA FOUND</td>
<td>ORA-01403</td>
<td>+100</td>
<td>Single row SELECT returned no rows or your program referenced a deleted element in a nested table or an uninitialized element in an associative array (index-by table).</td>
</tr>
<tr>
<td>PROGRAM ERROR</td>
<td>ORA-06501</td>
<td>-6501</td>
<td>PL/SQL has an internal problem.</td>
</tr>
<tr>
<td>ROWTYPE MISMATCH</td>
<td>ORA-06504</td>
<td>-6504</td>
<td>Host cursor variable and PL/SQL cursor variable involved in an assignment statement have incompatible return types.</td>
</tr>
<tr>
<td>STORAGE ERROR</td>
<td>ORA-06500</td>
<td>-6500</td>
<td>PL/SQL ran out of memory or memory was corrupted.</td>
</tr>
<tr>
<td>SUBSCRIPT BEYOND COUNT</td>
<td>ORA-06533</td>
<td>-6533</td>
<td>A program referenced a nested table or varray using an index number larger than the number of elements in the collection.</td>
</tr>
</tbody>
</table>
Trapping exceptions

**Example 4–1 Using the ZERO_DIVIDE predefined exception**

In this example, a PL/SQL program attempts to divide by 0. The ZERO_DIVIDE predefined exception is used to trap the error in an exception-handling routine.

```
Command> DECLARE v_invalid PLS_INTEGER;
> BEGIN
>   v_invalid := 100/0;
>   EXCEPTION
>   WHEN ZERO_DIVIDE THEN
>     DBMS_OUTPUT.PUT_LINE ('Attempt to divide by 0');
> END;
> /
Attempt to divide by 0
```

PL/SQL procedure successfully completed.

**Trapping user-defined exceptions**

You can define your own exceptions in PL/SQL in TimesTen, and you can raise user-defined exceptions explicitly with either the PL/SQL RAISE statement or the RAISE_APPLICATION_ERROR procedure.

**Using the RAISE statement**

The RAISE statement stops normal execution of a PL/SQL block or subprogram and transfers control to an exception handler. RAISE statements can raise predefined exceptions, or user-defined exceptions whose names you decide.

**Example 4–2 Using RAISE statement to trap user-defined exception**

In this example, the department number 500 does not exist, so no rows are updated in the departments table. The RAISE statement is used to explicitly raise an exception and display an error message, returned by the SQLERRM built-in function, and an error code, returned by the SQLCODE built-in function. Use the RAISE statement by itself within an exception handler to raise the same exception again and propagate it back to the calling environment.

---

**Table 4–1 (Cont.) Predefined exceptions**

<table>
<thead>
<tr>
<th>Exception name</th>
<th>Oracle Database error number</th>
<th>SQLCODE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBSCRIPT_OUTSIDE_LIMIT</td>
<td>ORA-06532</td>
<td>-6532</td>
<td>A program referenced a nested table or varray element using an index number that is outside the legal range (for example, -1).</td>
</tr>
<tr>
<td>SYS_INVALID_ROWID</td>
<td>ORA-01410</td>
<td>-1410</td>
<td>The conversion of a character string into a universal rowid failed because the character string does not represent a ROWID value.</td>
</tr>
<tr>
<td>TOO_MANY_ROWS</td>
<td>ORA-01422</td>
<td>-1422</td>
<td>Single row SELECT returned multiple rows.</td>
</tr>
<tr>
<td>VALUE_ERROR</td>
<td>ORA-06502</td>
<td>-6502</td>
<td>An arithmetic, conversion, truncation, or size constraint error occurred.</td>
</tr>
<tr>
<td>ZERO_DIVIDE</td>
<td>ORA-01476</td>
<td>-1476</td>
<td>A program attempted to divide a number by zero.</td>
</tr>
</tbody>
</table>

**Example 4–1 Using the ZERO_DIVIDE predefined exception**

In this example, a PL/SQL program attempts to divide by 0. The ZERO_DIVIDE predefined exception is used to trap the error in an exception-handling routine.

```
Command> DECLARE v_invalid PLS_INTEGER;
> BEGIN
>   v_invalid := 100/0;
>   EXCEPTION
>   WHEN ZERO_DIVIDE THEN
>     DBMS_OUTPUT.PUT_LINE ('Attempt to divide by 0');
> END;
> /
Attempt to divide by 0
```

PL/SQL procedure successfully completed.

**Trapping user-defined exceptions**

You can define your own exceptions in PL/SQL in TimesTen, and you can raise user-defined exceptions explicitly with either the PL/SQL RAISE statement or the RAISE_APPLICATION_ERROR procedure.

**Using the RAISE statement**

The RAISE statement stops normal execution of a PL/SQL block or subprogram and transfers control to an exception handler. RAISE statements can raise predefined exceptions, or user-defined exceptions whose names you decide.

**Example 4–2 Using RAISE statement to trap user-defined exception**

In this example, the department number 500 does not exist, so no rows are updated in the departments table. The RAISE statement is used to explicitly raise an exception and display an error message, returned by the SQLERRM built-in function, and an error code, returned by the SQLCODE built-in function. Use the RAISE statement by itself within an exception handler to raise the same exception again and propagate it back to the calling environment.
Command> DECLARE
    > v_deptno NUMBER := 500;
    > v_name VARCHAR2 (20) := 'Testing';
    > e_invalid_dept EXCEPTION;
    > BEGIN
    > UPDATE departments
    > SET department_name = v_name
    > WHERE department_id = v_deptno;
    > IF SQL%NOTFOUND THEN
    > RAISE e_invalid_dept;
    > END IF;
    > ROLLBACK;
    > EXCEPTION
    > WHEN e_invalid_dept THEN
    > DBMS_OUTPUT.PUT_LINE ('No such department');
    > DBMS_OUTPUT.PUT_LINE (SQLERRM);
    > DBMS_OUTPUT.PUT_LINE (SQLCODE);
    > END;
    > /
No such department
User-Defined Exception
1
PL/SQL procedure successfully completed.
The command succeeded.

Note: Given the same error condition in TimesTen and Oracle Database, SQLCODE returns the same error code, but SQLERRM does not necessarily return the same error message. This is also noted in "TimesTen error messages and SQL codes" on page 4-9.

Using the RAISE_APPLICATION_ERROR procedure
Use the RAISE_APPLICATION_ERROR procedure in the executable section or exception section (or both) of your PL/SQL program. TimesTen reports errors to your application so you can avoid returning unhandled exceptions.

Use an error number between -20,000 and -20,999. Specify a character string up to 2,048 bytes for your message.

Example 4–3 Using the RAISE_APPLICATION_ERROR procedure
This example attempts to delete from the employees table where last_name=Patterson. The RAISE_APPLICATION_ERROR procedure raises the error, using error number -20201.

Command> DECLARE
    > v_last_name employees.last_name%TYPE := 'Patterson';
    > BEGIN
    > DELETE FROM employees WHERE last_name = v_last_name;
    > IF SQL%NOTFOUND THEN
    > RAISE_APPLICATION_ERROR (-20201, v_last_name || ' does not exist');
    > END IF;
    > END;
    > /
8507: ORA-20201: Patterson does not exist
8507: ORA-06512: at line 6
The command failed.
Showing errors in ttIsql

You can use the `show errors` command in ttIsql to see details about errors you encounter in executing anonymous blocks or compiling packages, procedures, or functions. This is shown in Example 4–4.

**Example 4–4  ttIsql show errors command**

Again consider Example 2–17 on page 2-26. Assume the same package specification shown there, which declares the procedures and functions `hire_employee`, `remove_employee`, and `num_above_salary`. But instead of the body definition shown there, consider the following, which defines `hire_employee` and `num_above_salary` but not `remove_employee`:

```sql
CREATE OR REPLACE PACKAGE BODY emp_actions AS
    -- Code for procedure hire_employee:
    procedure hire_employee (employee_id NUMBER,
        last_name VARCHAR2,
        first_name VARCHAR2,
        email VARCHAR2,
        phone_number VARCHAR2,
        hire_date DATE,
        job_id VARCHAR2,
        salary NUMBER,
        commission_pct NUMBER,
        manager_id NUMBER,
        department_id NUMBER) IS
        BEGIN
            INSERT INTO employees VALUES (employee_id,
                last_name,
                first_name,
                email,
                phone_number,
                hire_date,
                job_id,
                salary,
                commission_pct,
                manager_id,
                department_id);
        END hire_employee;

    -- Code for function num_above_salary:
    function num_above_salary (emp_id NUMBER) RETURN NUMBER IS
        emp_sal NUMBER(8,2);
        num_count NUMBER;
        BEGIN
            SELECT salary INTO emp_sal FROM employees
                WHERE employee_id = emp_id;
            SELECT COUNT(*) INTO num_count FROM employees
                WHERE salary > emp_sal;
            RETURN num_count;
        END num_above_salary;
    END emp_actions;
/
```

Attempting this body definition after the original package specification results in the following:

`Warning: Package body created with compilation errors.`
To get more information, run `ttIsql` and use the command `show errors`. In this example, `show errors` provides the following:

```
Command> show errors;
Errors for PACKAGE BODY EMP_ACTIONS:

LINE/COL ERROR
-----------------------------------------------
13/13   PLS-00323: subprogram or cursor 'REMOVE_EMPLOYEE' is declared in a package specification and must be defined in the package body
```

Differences in TimesTen: exception handing and error behavior

You should be aware of some error-related behaviors that differ between TimesTen PL/SQL and Oracle Database PL/SQL:

- TimesTen PL/SQL transaction and rollback behavior for unhandled exceptions
- TimesTen error messages and SQL codes
- Warnings not visible in PL/SQL
- Unsupported predefined errors
- Possibility of runtime errors after clean compile (use of Oracle Database SQL parser)
- Use of TimesTen expressions at runtime

TimesTen PL/SQL transaction and rollback behavior for unhandled exceptions

TimesTen PL/SQL differs from Oracle Database PL/SQL in a scenario where an application executes PL/SQL in the middle of a transaction, and an unhandled exception occurs during execution of the PL/SQL. Oracle Database rolls back to the beginning of the anonymous block. TimesTen does not roll back.

An application should always handle any exception that results from execution of a PL/SQL block, as in the following example, run with autocommit disabled:

```
create table mytable (num int not null primary key);
set serveroutput on
insert into mytable values(1);
begin
  insert into mytable values(2);
  insert into mytable values(1);
  exception
    when dup_val_on_index then
      dbms_output.put_line('oops: ' || sqlerrm);
      rollback;
  end;
select * from mytable;
commit;
```

The second INSERT fails because values must be unique, so an exception occurs and the program performs a rollback. Running this in TimesTen results in the following:

```
oops:TT0907: Unique constraint (MYTABLE) violated at Rowid <BMUFVUAAABQAAAADjg>
select * from mytable;
```
0 rows found.

The result is equivalent in Oracle Database, with the `SELECT` results showing no rows.

Now consider a TimesTen example where the exception is not handled, again run with autocommit disabled:

```sql
create table mytable (num int not null primary key);
set serveroutput on

insert into mytable values(1);
bEGIN
  insert into mytable values(2);
  insert into mytable values(1);
end;

select * from mytable;

commit;
```

In TimesTen, the `SELECT` query indicates execution of the first two inserts:

```
907: Unique constraint (MYTABLE) violated at Rowid <BMUFVUAAABQAAAADjq>
8507: ORA-06512: at line 3
The command failed.
```

```sql
select * from mytable;
< 1 >
< 2 >
2 rows found.
```

If you execute this in Oracle Database, there is a rollback to the beginning of the PL/SQL block, so the results of the `SELECT` indicate execution of only the first insert:

```
ORA-00001: unique constraint (SYSTEM.SYS_C004423) violated
ORA-06512: at line 3
```

```
NUM
-------
 1
```

**Notes:**

- If there is an unhandled exception in a PL/SQL block, TimesTen leaves the transaction open only to allow the application to assess its state and determine appropriate action.

- An application in TimesTen should not execute a PL/SQL block while there are uncommitted changes in the current transaction, unless those changes together with the PL/SQL operations really do constitute a single logical unit of work and the application is able to determine appropriate action. Such action, for example, might consist of a rollback to the beginning of the transaction.

- If autocommit is enabled and an unhandled exception occurs in TimesTen, the entire transaction is rolled back.
TimesTen error messages and SQL codes

Given the same error condition, TimesTen does not guarantee that the error message returned by TimesTen is the same as the message returned by Oracle Database, although the SQL code is the same. Therefore, the information returned by the \texttt{SQLERRM} function may be different, but that returned by the \texttt{SQLCODE} function is the same.

For further information:
- Example 4–2 on page 4-4 uses \texttt{SQLERRM} and \texttt{SQLCODE}.
- Refer to "Warnings and Errors" in Oracle TimesTen In-Memory Database Error Messages and SNMP Traps for information about specific TimesTen error messages.
- Refer to "SQLERRM Function" and "SQLCODE Function" in Oracle Database PL/SQL Language Reference for general information.

Warnings not visible in PL/SQL

Oracle Database does not have the concept of runtime warnings, so Oracle Database PL/SQL does not support warnings.

TimesTen does have the concept of warnings, but because the TimesTen PL/SQL implementation is based on the Oracle Database PL/SQL implementation, TimesTen PL/SQL does not support warnings.

As a result, in TimesTen you could execute a SQL statement and see a resulting warning, but if you execute the same statement through PL/SQL you would not see the warning.

Unsupported predefined errors

"Trapping predefined TimesTen errors" on page 4-3 lists predefined exceptions supported by TimesTen, the associated \texttt{ORA} error numbers and \texttt{SQLCODE} values, and descriptions of the exceptions.

Table 4–2 notes predefined exceptions that are not supported by TimesTen.

<table>
<thead>
<tr>
<th>Exception name</th>
<th>Oracle Database error number</th>
<th>SQLCODE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOGIN_DENIED</td>
<td>ORA-01017</td>
<td>-1017</td>
<td>User name or password is invalid.</td>
</tr>
<tr>
<td>NOT_LOGGED_ON</td>
<td>ORA-01012</td>
<td>-1012</td>
<td>Program issued a database call without being connected to the database.</td>
</tr>
<tr>
<td>SELF_IS_NULL</td>
<td>ORA-30625</td>
<td>-30625</td>
<td>Program attempted to invoke a \texttt{MEMBER} method, but the object was not initialized.</td>
</tr>
<tr>
<td>TIMEOUT_ON_RESOURCE</td>
<td>ORA-00051</td>
<td>-51</td>
<td>Timeout occurred while the database was waiting for a resource.</td>
</tr>
</tbody>
</table>

Possibility of runtime errors after clean compile (use of Oracle Database SQL parser)

The TimesTen PL/SQL implementation uses the Oracle Database SQL parser in compiling PL/SQL programs. (This is discussed in "PL/SQL in TimesTen versus PL/SQL in Oracle Database" on page 1-3.) As a result, if your program uses Oracle...
Database syntax or built-in procedures that are not supported by TimesTen, the issue is not discovered during compilation. A runtime error occurs during program execution, however.

**Use of TimesTen expressions at runtime**

TimesTen SQL includes several constructs that are not present in Oracle Database SQL. The PL/SQL language does not include these constructs. To use TimesTen-specific SQL from PL/SQL, execute the SQL statements using the `EXECUTE IMMEDIATE` statement. This avoids compilation errors.

For lists of TimesTen-specific SQL and expressions, see "Compatibility Between TimesTen and Oracle Databases" in *Oracle TimesTen Application-Tier Database Cache User’s Guide*.

For more information about `EXECUTE IMMEDIATE`, refer to "Dynamic SQL in PL/SQL (EXECUTE IMMEDIATE statement)" on page 2-12.
Examples Using the SELECT...INTO statement in PL/SQL

Use the SELECT...INTO statement to retrieve exactly one row of data. TimesTen returns an error for any query that returns no rows or multiple rows.

The section provides the following examples:

- Using SELECT...INTO to return sum of salaries
- Using SELECT...INTO to query another user's table

Using SELECT...INTO to return sum of salaries

This example uses a SELECT...INTO statement to calculate the sum of salaries for all employees in the department where department_id is 60.

Example 5–1 Using SELECT...INTO to return sum of salaries

```
Command> DECLARE
  >   v_sum_sal  NUMBER (10,2);
  >   v_dept_no  NUMBER NOT NULL := 60;
  > BEGIN
  >   SELECT SUM(salary) -- aggregate function
  >   INTO v_sum_sal FROM employees
  >   WHERE department_id = v_dept_no;
  >   DBMS_OUTPUT.PUT_LINE ('Sum is ' || v_sum_sal);
  > END;
  > /
```
Sum is 28800

PL/SQL procedure successfully completed.

---

**Using SELECT...INTO to query another user's table**

This example provides two users, USER1 and USER2, to show one user employing SELECT...INTO to query another user's table.

**Example 5–2 Using SELECT...INTO to query another user's table**

The following privileges are assumed:

grant create session to user1;
grant create session to user2;
grant create table to user1;
grant select on user1.test to user2;

**USER1:**

Command> create table test(name varchar2(20), id number);
Command> insert into test values('posey', 363);
1 row inserted.

**USER2:**

Command> declare
    targetid number;
begin
    select id into targetid from user1.test where name='posey';
    dbms_output.put_line('Target ID is ' || targetid);
    end;
/
Target ID is 363

PL/SQL procedure successfully completed.

---

**Example using the INSERT statement**

TimesTen supports the TimesTen DML statements INSERT, UPDATE, DELETE, and MERGE. This section has an example of the INSERT statement.

**Example 5–3 Using the INSERT statement in PL/SQL**

This example uses the AS SELECT query clause to create table emp_copy, sets AUTOCOMMIT off, creates a sequence to increment employee_id, and uses the INSERT statement in PL/SQL to insert a row of data in table emp_copy.

Command> CREATE TABLE emp_copy AS SELECT * FROM employees;
107 rows inserted.
Command> SET AUTOCOMMIT OFF;

Command> CREATE SEQUENCE emp_copy_seq
    START WITH 207
    INCREMENT BY 1;
Command> BEGIN
    INSERT INTO emp_copy
        (employee_id, first_name, last_name, email, hire_date, job_id,
        salary)
    VALUES (emp_copy_seq.NEXTVAL, 'Parker', 'Cores', 'PCORES', SYSDATE,
Examples using input and output parameters and bind variables

The following examples in this section use IN, OUT, and IN OUT parameters, including bind variables (host variables) from outside PL/SQL:

- Using IN and OUT parameters
- Using IN OUT parameters
- Using associative arrays

Using IN and OUT parameters

This example creates a procedure query_emp to retrieve information about an employee, passes the employee_id value 171 to the procedure, and retrieves the name and salary into two OUT parameters.

Example 5–4 Using IN and OUT parameters

Command> CREATE OR REPLACE PROCEDURE query_emp
>   (p_id IN employees.employee_id%TYPE,
>    p_name  OUT employees.last_name%TYPE,
>    p_salary OUT employees.salary%TYPE)
>   IS
>      BEGIN
>         SELECT last_name, salary INTO p_name, p_salary
>         FROM employees
Examples using input and output parameters and bind variables

> WHERE employee_id = p_id;
> END query_emp;
> /

Procedure created.

Command> -- Execute the procedure
> DECLARE
> v_emp_name employees.last_name%TYPE;
> v_emp_sal employees.salary%TYPE;
> BEGIN
> query_emp (171, v_emp_name, v_emp_sal);
> DBMS_OUTPUT.PUT_LINE ('v_emp_name || ' earns ' ||
> TO_CHAR (v_emp_sal, '999,999.00'));
> END;
> /
Smith earns $7,400.00

PL/SQL procedure successfully completed.

Using IN OUT parameters

Consider a situation where you want to format a phone number. This example takes a 10-character string containing digits for a phone number and passes this unformatted string to a procedure as an IN OUT parameter. After the procedure is executed, the IN OUT parameter contains the formatted phone number value.

Example 5–5 Using IN OUT parameters

Command> CREATE OR REPLACE PROCEDURE format_phone
> (p_phone_no IN OUT VARCHAR2 ) IS
> BEGIN
> p_phone_no := '(' || SUBSTR (p_phone_no,1,3) ||
> ')' ' ' || SUBSTR (p_phone_no,4,3) ||
> '-' || SUBSTR (p_phone_no,7);
> END format_phone;
> /

Procedure created.

Create the bind variable, execute the procedure, and verify the results.

Command> VARIABLE b_phone_no VARCHAR2 (15);
Command> EXECUTE :b_phone_no := '8006330575';

PL/SQL procedure successfully completed.

Command> PRINT b_phone_no;
B_PHONE_NO           : 8006330575
Command> BEGIN
> format_phone (:b_phone_no);
> END;
> /

PL/SQL procedure successfully completed.

Command> PRINT b_phone_no
B_PHONE_NO           : (800) 633-0575
Using associative arrays

This example uses `ttIsql` to bind a `NUMBER` array and a `VARCHAR2` array to corresponding `OUT` associative arrays in a PL/SQL procedure.

See "Using associative arrays from applications" on page 3-6 for related information.

**Example 5–6 Binding to an associative array from ttIsql**

Assume the following SQL setup.

```
DROP TABLE FOO;

CREATE TABLE FOO (CNUM INTEGER,
                   CVC2 VARCHAR2(20));

INSERT INTO FOO VALUES ( null,
                         'VARCHAR 1');
INSERT INTO FOO VALUES (-102,
                        null);
INSERT INTO FOO VALUES ( 103,
                         'VARCHAR 3');
INSERT INTO FOO VALUES (-104,
                         'VARCHAR 4');
INSERT INTO FOO VALUES ( 105,
                         'VARCHAR 5');
INSERT INTO FOO VALUES ( 106,
                         'VARCHAR 6');
INSERT INTO FOO VALUES ( 107,
                         'VARCHAR 7');
INSERT INTO FOO VALUES ( 108,
                         'VARCHAR 8');

COMMIT;
```

Assume the following PL/SQL package definition. This includes the `INTEGER` associative array type `NUMARRTYP` and the `VARCHAR2` associative array type `VCHARRTYP`, used for output associative arrays `c1` and `c2`, respectively, in the definition of procedure `P1`.

```
CREATE OR REPLACE PACKAGE PKG1 AS
  TYPE NUMARRTYP IS TABLE OF INTEGER INDEX BY BINARY_INTEGER;
  TYPE VCHARRTYP IS TABLE OF VARCHAR2(20) INDEX BY BINARY_INTEGER;

  PROCEDURE P1(c1 OUT NUMARRTYP,c2 OUT VCHARRTYP);
END PKG1;
/

CREATE OR REPLACE PACKAGE BODY PKG1 AS

  CURSOR CUR1 IS SELECT CNUM, CVC2 FROM FOO;

  PROCEDURE P1(c1 OUT NUMARRTYP,c2 OUT VCHARRTYP) IS
    BEGIN
      IF NOT CUR1%ISOPEN THEN
        OPEN CUR1;
      END IF;
      FOR i IN 1..8 LOOP
        FETCH CUR1 INTO c1(i), c2(i);
        IF CUR1%NOTFOUND THEN
```

```
Examples using cursors

TimesTen supports cursors, as discussed in "Use of cursors in PL/SQL programs" on page 2-21. Use a cursor to handle the result set of a SELECT statement.

Examples in this section cover the following:

- Fetching values
- Using the %ROWCOUNT and %NOTFOUND attributes
- Using cursor FOR loops

See "Explicit Cursor Attributes" in Oracle Database PL/SQL Language Reference for information about the cursor attributes used in these examples.

Fetching values

This section provides examples of how to fetch values from a cursor, including how to fetch the values into a record.
Example 5–7  Fetching values from a cursor

The following example uses a cursor to select `employee_id` and `last_name` from the `employees` table where `department_id` is 30. Two variables are declared to hold the fetched values from the cursor, and the `FETCH` statement retrieves rows one at a time in a loop to retrieve all rows. Execution stops when there are no remaining rows in the cursor, illustrating use of the `%NOTFOUND` cursor attribute.

%NOTFOUND yields TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows.

Command>
```plsql
DECLARE
  CURSOR c_emp_cursor IS
    SELECT employee_id, last_name FROM employees
    WHERE department_id = 30;
  v_empno employees.employee_id%TYPE;
  v_lname employees.last_name%TYPE;
BEGIN
  OPEN c_emp_cursor;
  LOOP
    FETCH c_emp_cursor INTO v_empno, v_lname;
    EXIT WHEN c_emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE (v_empno || ' ' || v_lname);
  END LOOP;
  CLOSE c_emp_cursor;
END;
/
```

114 Raphaely
115 Khoo
116 Baida
117 Tobias
118 Himuro
119 Colmenares

Example 5–8  Fetching values into a record

This is similar to Example 5–7 above, with the same results, but fetches the values into a PL/SQL record instead of PL/SQL variables.

Command>
```plsql
DECLARE
  CURSOR c_emp_cursor IS
    SELECT employee_id, last_name FROM employees
    WHERE department_id = 30;
  v_emp_record c_emp_cursor%ROWTYPE;
BEGIN
  OPEN c_emp_cursor;
  LOOP
    FETCH c_emp_cursor INTO v_emp_record;
    EXIT WHEN c_emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE (v_emp_record.employee_id || ' ' || v_emp_record.last_name);
  END LOOP;
  CLOSE c_emp_cursor;
END;
/
```

114 Raphaely
115 Khoo
116 Baida
117 Tobias
118 Himuro
Using the %ROWCOUNT and %NOTFOUND attributes

Example 5–9 shows how to use the %ROWCOUNT cursor attribute as well as the %NOTFOUND cursor attribute previously shown in Example 5–7 on page 5-7 and Example 5–8 above.

Example 5–9  Using %ROWCOUNT and %NOTFOUND attributes

This example has the same results as Example 5–8, but illustrating the %ROWCOUNT cursor attribute as well as the %NOTFOUND attribute for exit conditions in the loop.

%ROWCOUNT yields the number of rows affected by an INSERT, UPDATE, or DELETE statement or returned by a SELECT...INTO or FETCH...INTO statement.

Command>
\[
\begin{align*}
&\text{DECLARE} \\
&\qquad \text{CURSOR c_emp_cursor IS} \\
&\qquad \qquad \text{SELECT employee_id, last_name FROM employees} \\
&\qquad \qquad \text{WHERE department_id = 30;} \\
&\qquad \qquad v_emp_record c_emp_cursor%ROWTYPE; \\
&\qquad \text{BEGIN} \\
&\qquad \qquad \text{OPEN c_emp_cursor;} \\
&\qquad \qquad \text{LOOP} \\
&\qquad \qquad \qquad \text{FETCH c_emp_cursor INTO v_emp_record;} \\
&\qquad \qquad \qquad \text{EXIT WHEN c_emp_cursor%ROWCOUNT > 10 OR c_emp_cursor%NOTFOUND;} \\
&\qquad \qquad \qquad \text{DBMS_OUTPUT.PUT_LINE (v_emp_record.employee_id || ' ' || v_emp_record.last_name);} \\
&\qquad \qquad \qquad \text{END LOOP;} \\
&\qquad \qquad \text{CLOSE c_emp_cursor;} \\
&\qquad \text{END;}
\end{align*}
\]

PL/SQL procedure successfully completed.
Examples using FORALL and BULK COLLECT

Examples using FORALL and BULK COLLECT

TimesTen supports bulk binding and the FORALL statement and BULK COLLECT feature, as noted in "FORALL and BULK COLLECT operations" on page 2-14.

Examples in this section cover the following:

- Using FORALL with SQL%BULK_ROWCOUNT
- Using BULK COLLECT INTO with queries
- Using BULK COLLECT INTO with cursors
- Using SAVE EXCEPTIONS with BULK COLLECT

Using FORALL with SQL%BULK_ROWCOUNT

The %BULK_ROWCOUNT cursor attribute is a composite structure designed for use with the FORALL statement.

The attribute acts like an associative array (index-by table). Its \(i\)th element stores the number of rows processed by the \(i\)th execution of the INSERT statement. If the \(i\)th execution affects no rows, then \(\text{%BULK_ROWCOUNT}(i)\) returns zero.

This is demonstrated in Example 5-12.

Example 5–11  Using a cursor FOR loop with subqueries

This example illustrates a FOR loop using subqueries. The results are the same as in Example 5–9 on page 5-8 and Example 5–10 above.

Command>

```sql
BEGIN
  FOR emp_record IN (SELECT employee_id, last_name FROM employees WHERE department_id = 30)
  LOOP
    DBMS_OUTPUT.PUT_LINE (emp_record.employee_id || ' ' || emp_record.last_name);
  END LOOP;
END;
/ 
114 Raphaely
115 Khoo
116 Baida
117 Tobias
118 Himuro
119 Colmenares 

PL/SQL procedure successfully completed.
```

Examples using FORALL and BULK COLLECT
**Example 5–12  Using the FORALL statement with SQL%BULK_ROWCOUNT**

Command> DECLARE
   > TYPE num_list_type IS TABLE OF NUMBER
   >      INDEX BY BINARY_INTEGER;
   > v_nums num_list_type;
   > BEGIN
   >   v_nums (1) := 1;
   >   v_nums (2) := 3;
   >   v_nums (3) := 5;
   >   v_nums (4) := 7;
   >   v_nums (5) := 11;
   >   FORALL i IN v_nums.FIRST .. v_nums.LAST
   >      INSERT INTO num_table (n) VALUES (v_nums (i));
   >   FOR i IN v_nums.FIRST .. v_nums.LAST
   >   LOOP
   >      DBMS_OUTPUT.PUT_LINE ('Inserted ' ||
   >         SQL%BULK_ROWCOUNT (i) || ' row (s)' ||
   >         ' on iteration ' || i);
   >   END LOOP;
   > END;
   > /

Inserted 1 row (s) on iteration 1
Inserted 1 row (s) on iteration 2
Inserted 1 row (s) on iteration 3
Inserted 1 row (s) on iteration 4
Inserted 1 row (s) on iteration 5

PL/SQL procedure successfully completed.

**Using BULK COLLECT INTO with queries**

Use BULK COLLECT with the SELECT statement in PL/SQL to retrieve rows without using a cursor.

**Example 5–13  Using BULK COLLECT INTO with queries**

This example selects all rows from the departments table for a specified location into a nested table, then uses a FOR LOOP to output data.

Command> CREATE OR REPLACE PROCEDURE get_departments (p_loc NUMBER) IS
   >   TYPE dept_tab_type IS TABLE OF departments%ROWTYPE;
   >   v_depts dept_tab_type;
   > BEGIN
   >   SELECT * BULK COLLECT INTO v_depts
   >   FROM departments
   >   WHERE location_id = p_loc;
   >   FOR i IN 1 .. v_depts.COUNT
   >   LOOP
   >      DBMS_OUTPUT.PUT_LINE (v_depts(i).department_id
   >         || ' ' || v_depts (i).department_name);
   >   END LOOP;
   > END;
   > /

Procedure created.

The following executes the procedure and verifies the results:

Command> EXECUTE GET_DEPARTMENTS (1700);
Using BULK COLLECT INTO with cursors

Example 5–14 uses a cursor to bulk-collect rows from a table.

Example 5–14 Using BULK COLLECT INTO with cursors

This example uses a cursor to bulk-collect rows from the departments table with a specified location_id value. Results are the same as in Example 5–13 above.

Command> CREATE OR REPLACE PROCEDURE get_departments2 (p_loc NUMBER) IS
   >   CURSOR cur_dept IS

```sql
10 Administration
30 Purchasing
90 Executive
100 Finance
110 Accounting
120 Treasury
130 Corporate Tax
140 Control And Credit
150 Shareholder Services
160 Benefits
170 Manufacturing
180 Construction
190 Contracting
200 Operations
210 IT Support
220 NOC
230 IT Helpdesk
240 Government Sales
250 Retail Sales
260 Recruiting
270 Payroll
```

PL/SQL procedure successfully completed.

Command> SELECT department_id, department_name FROM departments WHERE location_id = 1700;
< 10, Administration >
< 30, Purchasing >
< 90, Executive >
< 100, Finance >
< 110, Accounting >
< 120, Treasury >
< 130, Corporate Tax >
< 140, Control And Credit >
< 150, Shareholder Services >
< 160, Benefits >
< 170, Manufacturing >
< 180, Construction >
< 190, Contracting >
< 200, Operations >
< 210, IT Support >
< 220, NOC >
< 230, IT Helpdesk >
< 240, Government Sales >
< 250, Retail Sales >
< 260, Recruiting >
< 270, Payroll >
21 rows found.
Examples using FORALL and BULK COLLECT

```sql
> SELECT * FROM departments
> WHERE location_id = p_loc;
> TYPE dept_tab_type IS TABLE OF cur_dept%ROWTYPE;
> v_depts dept_tab_type;
> BEGIN
>   OPEN cur_dept;
>   FETCH cur_dept BULK COLLECT INTO v_depts;
>   CLOSE cur_dept;
> FOR i IN 1 .. v_depts.COUNT
>   LOOP
>     DBMS_OUTPUT.PUT_LINE (v_depts (i).department_id
>     || ' ' || v_depts (i).department_name );
>   END LOOP;
> END;
> /
```

Procedure created.

Command> EXECUTE GET_DEPARTMENTS2 (1700);
10 Administration
30 Purchasing
90 Executive
100 Finance
110 Accounting
120 Treasury
130 Corporate Tax
140 Control And Credit
150 Shareholder Services
160 Benefits
170 Manufacturing
180 Construction
190 Contracting
200 Operations
210 IT Support
220 NOC
230 IT Helpdesk
240 Government Sales
250 Retail Sales
260 Recruiting
270 Payroll

PL/SQL procedure successfully completed.

Using SAVE EXCEPTIONS with BULK COLLECT

SAVE EXCEPTIONS allows an UPDATE, INSERT, or DELETE statement to continue executing after it issues an exception. When the statement finishes, an error is issued to signal that at least one exception occurred. Exceptions are collected into an array that you can examine using %BULK_EXCEPTIONS after the statement has executed.

Example 5–15 Using SAVE EXCEPTIONS with BULK COLLECT

In this example, PL/SQL raises predefined exceptions because some new values are too large for the job_id column. After the FORALL statement, SQL%BULK_EXCEPTIONS.COUNT returns 2, and the contents of SQL%BULK_EXCEPTIONS are (7, 01401) and (13, 01401), indicating the error number and the line numbers where the error was detected. To get the error message, the negative of SQL%BULK_EXCEPTIONS(i).ERROR_CODE is passed to the error-reporting function SQLERRM (which expects a negative number).
The following script is executed using ttIsql:

```sql
-- create a temporary table for this example
CREATE TABLE emp_temp AS SELECT * FROM employees;

DECLARE
    TYPE empid_tab IS TABLE OF employees.employee_id%TYPE;
    emp_sr empid_tab;
-- create an exception handler for ORA-24381
    errors NUMBER;
    dml_errors EXCEPTION;
    PRAGMA EXCEPTION_INIT(dml_errors, -24381);
BEGIN
    SELECT employee_id
    BULK COLLECT INTO emp_sr FROM emp_temp
    WHERE hire_date < '1994-12-30';
-- add '_SR' to the job_id of the most senior employees
    FORALL i IN emp_sr.FIRST..emp_sr.LAST SAVE EXCEPTIONS
        UPDATE emp_temp SET job_id = job_id || '_SR'
        WHERE emp_sr(i) = emp_temp.employee_id;
-- If any errors occurred during the FORALL SAVE EXCEPTIONS,
-- a single exception is raised when the statement completes.
EXCEPTION
-- Figure out what failed and why
    WHEN dml_errors THEN
        errors := SQL%BULK_EXCEPTIONS.COUNT;
        DBMS_OUTPUT.PUT_LINE
            ('Number of statements that failed: ' || errors);
        FOR i IN 1..errors LOOP
            DBMS_OUTPUT.PUT_LINE('Error #' || i || ' occurred during '||
                'iteration #' || SQL%BULK_EXCEPTIONS(i).ERROR_INDEX);
            DBMS_OUTPUT.PUT_LINE('Error message is ' ||
                SQLERRM(-SQL%BULK_EXCEPTIONS(i).ERROR_CODE));
        END LOOP;
    END;
/
DROP TABLE emp_temp;
```

Results are as follows:

Number of statements that failed: 2
Error #1 occurred during iteration #7
Error message is ORA-01401: inserted value too large for column
Error #2 occurred during iteration #13
Error message is ORA-01401: inserted value too large for column

PL/SQL procedure successfully completed.

Examples using EXECUTE IMMEDIATE

Examples using EXECUTE IMMEDIATE

TimesTen supports the EXECUTE IMMEDIATE statement, as noted in "Dynamic SQL in PL/SQL (EXECUTE IMMEDIATE statement)" on page 2-12. This section provides the following additional examples to consider as you develop your PL/SQL applications in TimesTen:

- Using EXECUTE IMMEDIATE to create a table
- Using EXECUTE IMMEDIATE with a single row query
- Using EXECUTE IMMEDIATE to alter a connection attribute
- Using EXECUTE IMMEDIATE to call a TimesTen built-in procedure
- Using EXECUTE IMMEDIATE with TimesTen-specific syntax

Using EXECUTE IMMEDIATE to create a table

Consider a situation where you do not know your table definition at compilation. By using an EXECUTE IMMEDIATE statement, you can create your table at execution time. This example shows a procedure that creates a table using the EXECUTE IMMEDIATE statement. The procedure is executed with the table name and column definitions passed as parameters, then creation of the table is verified.

**Example 5–16 Using EXECUTE IMMEDIATE to create a table**

```sql
Command> CREATE OR REPLACE PROCEDURE create_table
  > (p_table_name VARCHAR2, p_col_specs VARCHAR2) IS
  > BEGIN
  >   EXECUTE IMMEDIATE 'CREATE TABLE ' || p_table_name
  >   || ' (' || p_col_specs|| ' )';
  > END;
  > /

Procedure created.

Execute the procedure and verify the table is created.

Command> BEGIN
  > create_table ('EMPLOYEES_NAMES', 'id NUMBER (4)
  >  PRIMARY KEY, name VARCHAR2 (40)');
  > END;
  > /

PL/SQL procedure successfully completed.

Command> DESCRIBE employees_names;

Table USER.EMPLOYEES_NAMES:
  Columns:
  *ID                              NUMBER (4) NOT NULL
  NAME                            VARCHAR2 (40) INLINE

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

Using EXECUTE IMMEDIATE with a single row query

In this example, the function get_emp retrieves an employee record. The function is executed and returns the results in v_emprec.

**Example 5–17 Using EXECUTE IMMEDIATE with a single row query**

```sql
Command> CREATE OR REPLACE FUNCTION get_emp (p_emp_id NUMBER)
  > RETURN employees%ROWTYPE IS
  > v_stmt VARCHAR2 (200);
  > v_emprec employees%ROWTYPE;
  > BEGIN
  >   v_stmt:= 'SELECT * FROM EMPLOYEES '||
  >     'WHERE employee_id = :p_emp_id';
```
Examples using EXECUTE IMMEDIATE

> EXECUTE IMMEDIATE v_stmt INTO v_emprec USING p_emp_id;
> RETURN v_emprec;
> END;
> /

Function created.

Command> DECLARE
>   v_emprec employees%ROWTYPE := GET_EMP (100);
> BEGIN
>   DBMS_OUTPUT.PUT_LINE ('Employee: ' || v_emprec.last_name);
> END;
> /

Employee: King

PL/SQL procedure successfully completed.

Using EXECUTE IMMEDIATE to alter a connection attribute

This example uses an EXECUTE IMMEDIATE statement with ALTER SESSION to alter the PLSQL_OPTIMIZE_LEVEL setting, calling the ttConfiguration built-in procedure before and after to verify the results. (The next example calls ttConfiguration from inside an EXECUTE IMMEDIATE statement.) Refer to "ttConfiguration" in Oracle TimesTen In-Memory Database Reference for information about this procedure.

**Example 5–18 Using EXECUTE IMMEDIATE to alter PLSCOPE_SETTINGS**

Command> call ttconfiguration;
...
< PLSCOPE_SETTINGS, IDENTIFIERS:NONE >
< PLSQL, 1 >
< PLSQL_CCFLAGS, <NULL> >
< PLSQL_CODE_TYPE, INTERPRETED >
< PLSQL_CONN_MEM_LIMIT, 100 >
< PLSQL_MEMORY_ADDRESS, 0x10000000 >
< PLSQL_MEMORY_SIZE, 32 >
< PLSQL_OPTIMIZE_LEVEL, 2 >
< PLSQL_TIMEOUT, 30 >
...
54 rows found.

Command> begin
>   execute immediate 'alter session set PLSQL_OPTIMIZE_LEVEL=3';
> end;
> /

PL/SQL procedure successfully completed.

Command> call ttconfiguration;
...
< PLSCOPE_SETTINGS, IDENTIFIERS:NONE >
< PLSQL, 1 >
< PLSQL_CCFLAGS, <NULL> >
< PLSQL_CODE_TYPE, INTERPRETED >
< PLSQL_CONN_MEM_LIMIT, 100 >
< PLSQL_MEMORY_ADDRESS, 0x10000000 >
< PLSQL_MEMORY_SIZE, 32 >
< PLSQL_OPTIMIZE_LEVEL, 3 >
< PLSQL_TIMEOUT, 30 >
...
54 rows found.
Using EXECUTE IMMEDIATE to call a TimesTen built-in procedure

In PL/SQL, you can use an EXECUTE IMMEDIATE statement with CALL syntax to call a TimesTen built-in procedure.

For example, to call the built-in procedure `ttConfiguration` and return its output result set, create a PL/SQL record type then use EXECUTE IMMEDIATE with BULK COLLECT to fetch the result set into an array.

For more information on TimesTen built-in procedures, see "Built-In Procedures" in Oracle TimesTen In-Memory Database Reference.

**Example 5–19 Using EXECUTE IMMEDIATE to call `ttConfiguration`**

```plsql
Command> DECLARE
  >   TYPE ttConfig_record IS RECORD
  >      (name varchar2(255), value varchar2 (255));
  >   TYPE ttConfig_table IS TABLE OF ttConfig_record;
  >   v_ttConfigs ttConfig_table;
  > BEGIN
  >  EXECUTE IMMEDIATE 'CALL ttConfiguration'
  >   BULK COLLECT into v_ttConfigs;
  >  DBMS_OUTPUT.PUT_LINE ('Name: ' || v_ttConfigs(1).name
  >    || ' Value: ' || v_ttConfigs(1).value);
  > end;
  > /
Name: CacheGridEnable Value: 0
```

PL/SQL procedure successfully completed.

Using EXECUTE IMMEDIATE with TimesTen-specific syntax

This example uses an EXECUTE IMMEDIATE statement to execute a TimesTen SELECT FIRST n statement. This syntax is specific to TimesTen.

**Example 5–20 Using EXECUTE IMMEDIATE with TimesTen-specific syntax**

```plsql
Command> DECLARE v_empid NUMBER;
  > BEGIN
  >   EXECUTE IMMEDIATE 'SELECT FIRST 1 employee_id FROM employees'
  >   INTO v_empid;
  >  DBMS_OUTPUT.PUT_LINE ('Employee id: ' || v_empid);
  > end;
  > /
Employee id: 100
```

PL/SQL procedure successfully completed.

Examples using RETURNING INTO

This section includes the following two examples using the RETURNING INTO clause:

- Using the RETURNING INTO clause with a record
- Using BULK COLLECT INTO with the RETURNING INTO clause

See "RETURNING INTO clause" on page 2-15 for an overview.
Using the RETURNING INTO clause with a record

The following example uses ttIsql to run a SQL script that uses a RETURNING INTO clause to return data into a record. The example gives a raise to a specified employee, returns his name and new salary into a record, then outputs the data from the record. For reference, the original salary is shown before running the script.

**Example 5–21 Using the RETURNING INTO clause with a record**

```sql
Command> SELECT SALARY,LAST_NAME FROM EMPLOYEES WHERE EMPLOYEE_ID = 100;
< 24000, King >
1 row found.

Command> run ReturnIntoWithRecord.sql;

CREATE TABLE emp_temp AS SELECT * FROM employees;
107 rows inserted.

DECLARE
  TYPE EmpRec IS RECORD (last_name employees.last_name%TYPE,
                           salary employees.salary%TYPE);
  emp_info EmpRec;
  emp_id NUMBER := 100;
BEGIN
  UPDATE emp_temp SET salary = salary * 1.1
  WHERE employee_id = emp_id
  RETURNING last_name, salary INTO emp_info;
  DBMS_OUTPUT.PUT_LINE
    ('Just gave a raise to ' || emp_info.last_name ||
     ', who now makes ' || emp_info.salary);
  ROLLBACK;
END;
/

Just gave a raise to King, who now makes 26400

PL/SQL procedure successfully completed.
```

Using BULK COLLECT INTO with the RETURNING INTO clause

The following example uses ttIsql to run a SQL script that uses a RETURNING INTO clause with BULK COLLECT to return data into nested tables, a type of PL/SQL collection. The example deletes all the employees from a specified department, then, using one nested table for employee IDs and one for last names, outputs the employee ID and last name of each deleted employee. For reference, the IDs and last names of employees in the department are also displayed before execution of the script.

**Example 5–22 Using BULK COLLECT INTO with the RETURNING INTO clause**

```sql
Command> select employee_id, last_name from employees where department_id=30;
< 114, Raphaely >
< 115, Khoo >
< 116, Baida >
< 117, Tobias >
< 118, Himuro >
< 119, Colmenares >
6 rows found.
Command> run ReturnIntoWithBulkCollect.sql;

CREATE TABLE emp_temp AS SELECT * FROM employees;
```

Examples Using TimesTen SQL in PL/SQL 5-17
DECLARE
    TYPE NumList IS TABLE OF employees.employee_id%TYPE;
    enums NumList;
    TYPE NameList IS TABLE OF employees.last_name%TYPE;
    names NameList;
BEGIN
    DELETE FROM emp_temp WHERE department_id = 30
    RETURNING employee_id, last_name
    BULK COLLECT INTO enums, names;
    DBMS_OUTPUT.PUT_LINE ('Deleted ' || SQL%ROWCOUNT || ' rows:');
    FOR i IN enums.FIRST .. enums.LAST
    LOOP
        DBMS_OUTPUT.PUT_LINE ('Employee #' || enums(i) || ': ' || names(i));
    END LOOP;
END;
/
Deleted 6 rows:
Employee #114: Raphaely
Employee #115: Khoo
Employee #116: Baida
Employee #117: Tobias
Employee #118: Himuro
Employee #119: Colmenares

PL/SQL procedure successfully completed.

Examples using the AUTHID clause

This section 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. See “Definer’s rights and invoker’s rights” on page 7-7 for
related information.

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.
Examples using the AUTHID clause

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 execution 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 example 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.

Example 5–23 Using AUTHID CURRENT_USER

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 example 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).

Example 5–24  Using AUTHID 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 allow 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.

Use the following to terminate all the connections:
Command> disconnect all;
Example querying a system view

This section provides an example that queries a system view.

**Example 5–25  Querying system view USER_SOURCE**

This example queries the \texttt{USER\_SOURCE} system view to examine the source code of procedure \texttt{query\_emp} from Example 5–4 on page 5-3. (You must create that procedure before completing this example.)

```
Command> SELECT SUBSTR (text, 1, LENGTH(text)-1) 
  > FROM user_source 
  > WHERE name = 'QUERY\_EMP' AND type = 'PROCEDURE';
```

This produces the following output:

```
< PROCEDURE query\_emp >
< (p\_id IN employees.employee\_id\%TYPE, >
< p\_name OUT employees.last\_name\%TYPE, >
< p\_salary OUT employees.salary\%TYPE) IS >
< BEGIN >
< SELECT last\_name, salary INTO p\_name, p\_salary >
< FROM employees >
< WHERE employee\_id = p\_id; >
< END query\_emp; >
9 rows found.
```

**Note:** As with other \texttt{USER\_*} system views, all users have \texttt{SELECT} privilege for the \texttt{USER\_SOURCE} system view.
PL/SQL Installation and Environment

The chapter shows you how to manage PL/SQL in your TimesTen database, set connection attributes, and display system-provided packages. It also describes the ttSrcScan utility, which you can use to check for PL/SQL features unsupported in TimesTen. The chapter concludes with examples to assist you in your setup procedures.

Topics in this chapter include:

- Confirming that PL/SQL is installed and enabled in TimesTen
- PL/SQL connection attributes
- The ttSrcScan utility

Confirming that PL/SQL is installed and enabled in TimesTen

This section covers the following topics:

- PL/SQL installation and the ttmodinstall utility
- Checking that PL/SQL is enabled in a TimesTen database

PL/SQL installation and the ttmodinstall utility

TimesTen installs PL/SQL by default. If you choose not to install PL/SQL (which is discouraged), you can use the TimesTen ttmodinstall utility to install it later. For more information, see "ttmodinstall" in Oracle TimesTen In-Memory Database Reference.

---

**Note:** Only the instance administrator can run this utility.

Checking that PL/SQL is enabled in a TimesTen database

PL/SQL is enabled by default (first connection attribute setting PLSQL=1). You can confirm the status of PL/SQL in your database in the following ways. In these examples, $ is the UNIX prompt and Command> is the ttIsql prompt.

- Use the ttStatus utility to determine if PL/SQL is enabled in your database, as indicated in the following example. See "ttStatus" in Oracle TimesTen In-Memory Database Reference for information about this utility.

  $ ttstatus
  TimesTen status report as of Sat Sep 17 13:58:27 2011
  Daemon pid 20921 port 28959 instance myserver
  ...

...
Using the ttIsql utility, call the ttConfiguration built-in procedure to determine the PLSQL connection attribute setting for your database. The value 1, as shown in the following example, indicates PL/SQL is enabled. Refer to "ttConfiguration" in Oracle TimesTen In-Memory Database Reference for information about this built-in procedure.

Command> call ttconfiguration;
...
< DataStore, /mypath/install/info/DemoDataStore/sampledb_1122 >
...
< PLSQL, 1 >
...
< UID, MYUSER >
61 rows found.

### PL/SQL connection attributes

There are several TimesTen connection attributes specific to PL/SQL, as summarized in Table 6–1 that follows. For additional information on these connection attributes, see "PL/SQL first connection attributes" and "PL/SQL general connection attributes" in Oracle TimesTen In-Memory Database Reference.

The table also notes any required access control privileges and whether each connection attribute is a first connection attribute or general connection attribute. First connection attributes are set when the database is first loaded, and persist for all connections. Only the instance administrator can load a database with changes to first connection attribute settings. A general connection attribute setting applies to one connection only, and requires no special privilege.

#### Table 6–1 PL/SQL Connection Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLSQL</td>
<td>First connection attribute&lt;br&gt;Required privilege: Instance administrator&lt;br&gt;Enables PL/SQL in the database.&lt;br&gt;If PLSQL=1, PL/SQL is enabled.&lt;br&gt;Default: 1 (for platforms where PL/SQL is supported)</td>
</tr>
</tbody>
</table>
### PL/SQL Connection Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Summary</th>
</tr>
</thead>
</table>
| **PLSQL_MEMORY_ADDRESS** | First connection attribute  
Required privilege: Instance administrator  
Specifies the virtual address, as a hexadecimal value, at which the PL/SQL shared memory segment is loaded into each process that uses the TimesTen direct drivers. This memory address must be identical in all connections to a given database and in all processes that connect to that database.  
If a single application simultaneously makes direct connections to multiple databases, then you must set different values for each of the databases.  
Default: Platform-specific value  
Refer to "PLSQL_MEMORY_ADDRESS" in Oracle TimesTen In-Memory Database Reference for platform-specific information. |
| **PLSQL_MEMORY_SIZE**     | First connection attribute  
Required privilege: Instance administrator  
Determines the size, in megabytes, of memory allocated for the PL/SQL shared memory segment, which is shared by all connections. This memory is used to hold recently executed PL/SQL code and metadata about PL/SQL objects.  
Default: Platform-specific value  
Refer to "PLSQL_MEMORY_SIZE" in Oracle TimesTen In-Memory Database Reference for information about calculating the PL/SQL memory size and for platform-specific values and tuning information. |
| **PLSCOPE_SETTINGS**     | General connection attribute  
Required privilege: None  
Controls whether the PL/SQL compiler generates cross-reference information. Possible values are IDENTIFIERS:NONE or IDENTIFIERS:ALL.  
You can use the ALTER SESSION statement to change this value within your session.  
Default: IDENTIFIERS:NONE |
**Table 6–1 (Cont.) PL/SQL Connection Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLSQL_CCFLAGS</td>
<td>General connection attribute</td>
</tr>
<tr>
<td></td>
<td>Required privilege: None</td>
</tr>
<tr>
<td></td>
<td>Use this to set inquiry directives to control conditional</td>
</tr>
<tr>
<td></td>
<td>compilation of PL/SQL units, which enables you to</td>
</tr>
<tr>
<td></td>
<td>customize the functionality of a PL/SQL program</td>
</tr>
<tr>
<td></td>
<td>depending on conditions that are checked. This is</td>
</tr>
<tr>
<td></td>
<td>especially useful when applications may be deployed to</td>
</tr>
<tr>
<td></td>
<td>multiple database environments. For example, to</td>
</tr>
<tr>
<td></td>
<td>activate debugging features:</td>
</tr>
<tr>
<td></td>
<td>PLSQL_CCFLAGS= 'DEBUG:TRUE'</td>
</tr>
<tr>
<td></td>
<td>PL/SQL conditional compilation flags are similar in</td>
</tr>
<tr>
<td></td>
<td>concept to flags on a C compiler command line, such as</td>
</tr>
<tr>
<td></td>
<td>the following:</td>
</tr>
<tr>
<td></td>
<td>% cc -DEBUG=TRUE ...</td>
</tr>
<tr>
<td></td>
<td>You can use the ALTER SESSION statement to change</td>
</tr>
<tr>
<td></td>
<td>PLSQL_CCFLAGS within your session.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Conditional Compilation&quot; in *Oracle Database</td>
</tr>
<tr>
<td></td>
<td>PL/SQL Language Reference* for information about this</td>
</tr>
<tr>
<td></td>
<td>feature.</td>
</tr>
<tr>
<td></td>
<td>Default: NULL</td>
</tr>
</tbody>
</table>

Default: NULL
PL/SQL connection attributes

Table 6–1 (Cont.) PL/SQL Connection Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLSQL_CONN_MEM_LIMIT</td>
<td>General connection attribute</td>
</tr>
<tr>
<td></td>
<td>Required privilege: None</td>
</tr>
</tbody>
</table>
|                                   | Specifies the maximum amount of PL/SQL shared memory (process heap memory) that PL/SQL can allocate for the current connection. (Note that this memory is not actually allocated until needed.) This is memory used for runtime data, such as large PL/SQL collections, as opposed to cached executable code. This limit setting protects other parts of your application, such as C or Java components, when PL/SQL might otherwise take all available runtime memory. The amount of space consumed by PL/SQL variables is roughly what you might expect comparable variables to consume in other programming languages. As an example, consider a large array of strings:

```sql
  type chararr is table of varchar2(32767)  
    index by binary_integer;  
  big_array chararr;  
  If 100,000 strings of 100 bytes each are placed into such an array, approximately 12 megabytes of memory is consumed.  
Memory consumed by variables in PL/SQL blocks is used while the block executes, then is released. Memory consumed by variables in PL/SQL package specifications or bodies (not within a procedure or function) is used for the lifetime of the package. Memory consumed by variables in a PL/SQL procedure or function, including one defined within a package, is used for the lifetime of the procedure or function. However, in all cases, memory freed by PL/SQL is not returned to the operating system. Instead, it is kept by PL/SQL and reused by future PL/SQL invocations. The memory is freed when the application disconnects from TimesTen.  
The PLSQL_CONN_MEM_LIMIT value is a number specified in megabytes. A setting of 0 means no limit. You can use the ALTER SESSION statement to change this value within your session. Default: 100 megabytes  
Note: In ttPLSQLMemoryStats output, the related value CurrentConnectionMemory indicates how much process heap memory PL/SQL has actually acquired through malloc(). (Also see Example 6–3 on page 6-8.)
```
| PLSQL_OPTIMIZE_LEVEL               | General connection attribute  |
|                                   | Required privilege: None  |
|                                   | Specifies the optimization level used to compile PL/SQL library units. The higher the setting, the more effort the compiler makes to optimize PL/SQL library units. Possible values are 0, 1, 2, or 3. You can use the ALTER SESSION statement to change this value within your session. Default: 2  
```sql
  type varchar2(1000)  
  big_array varchar2(1000);  
  If 100,000 strings of 100 bytes each are placed into such an array, approximately 12 megabytes of memory is consumed.  
Memory consumed by variables in PL/SQL blocks is used while the block executes, then is released. Memory consumed by variables in PL/SQL package specifications or bodies (not within a procedure or function) is used for the lifetime of the package. Memory consumed by variables in a PL/SQL procedure or function, including one defined within a package, is used for the lifetime of the procedure or function. However, in all cases, memory freed by PL/SQL is not returned to the operating system. Instead, it is kept by PL/SQL and reused by future PL/SQL invocations. The memory is freed when the application disconnects from TimesTen.  
The PLSQL_CONN_MEM_LIMIT value is a number specified in megabytes. A setting of 0 means no limit. You can use the ALTER SESSION statement to change this value within your session. Default: 100 megabytes  
Note: In ttPLSQLMemoryStats output, the related value CurrentConnectionMemory indicates how much process heap memory PL/SQL has actually acquired through malloc(). (Also see Example 6–3 on page 6-8.)
```
The rest of this section provides some examples for setting and altering PL/SQL connection attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLSQL_TIMEOUT</td>
<td>General connection attribute</td>
</tr>
<tr>
<td></td>
<td>Required privilege: None</td>
</tr>
<tr>
<td></td>
<td>Controls how long PL/SQL program units are allowed to run, in seconds,</td>
</tr>
<tr>
<td></td>
<td>before being terminated. A new value impacts PL/SQL programs currently</td>
</tr>
<tr>
<td></td>
<td>running. Possible values are 0 (meaning no time limit) or any positive</td>
</tr>
<tr>
<td></td>
<td>integer. You can use the ALTER SESSION statement to change this value</td>
</tr>
<tr>
<td></td>
<td>within your session. (Also be aware of TimesTen SQL query timeout</td>
</tr>
<tr>
<td></td>
<td>settings, as discussed in &quot;Setting a timeout duration for SQL</td>
</tr>
<tr>
<td></td>
<td>statements&quot; in Oracle TimesTen In-Memory Database C Developer’s Guide,</td>
</tr>
<tr>
<td></td>
<td>and the TimesTen TTC_Timeout setting, discussed in &quot;TTC_Timeout&quot; in</td>
</tr>
<tr>
<td></td>
<td>Oracle TimesTen In-Memory Database Reference.)</td>
</tr>
<tr>
<td></td>
<td>Default: 30 seconds</td>
</tr>
</tbody>
</table>

**Notes:**

- The frequency with which PL/SQL programs check execution time against this timeout value is variable. It is possible for programs to run significantly longer than the timeout value before being terminated.
- If you are using TimesTen Client/Server, PLSQL_TIMEOUT should be significantly less than TTC_Timeout, and cannot be 0 (for no timeout) if TTC_Timeout is greater than 0. For details, see the TTC_Timeout documentation referenced above.

---

**Notes:** There are additional TimesTen connection attributes you should consider for PL/SQL. For more information about them, refer to the indicated sections in Oracle TimesTen In-Memory Database Reference.

- If PL/SQL is enabled in your database, the value for the DDLCommitBehavior general connection attribute must be 0. See "DDLCommitBehavior".
- If the LockLevel general connection attribute is set to 1 (database-level locking), certain PL/SQL internal functions cannot be performed. Therefore, set LockLevel to 0 for your connection. You can then use the ttLockLevel built-in procedure to selectively switch to database-level locking for those transactions that require it. See "LockLevel" and "ttLockLevel".
- The PL/SQL shared memory segment is not subject to the MemoryLock first connection attribute. See "MemoryLock".
Example 6–1  Create a database with PL/SQL default connection attributes

This example defines a database pldef without specifying PL/SQL connection attributes. (Be aware that only an instance administrator can create a database.)

Sample odbc.ini entry:

```
[pldef]
Driver=/mypath/install/lib/libtten.so
DataStore=/mypath/install/info/DemoDataStore/pldef
DatabaseCharacterSet=US7ASCII
```

Connect to database pldef:

```
$ ttisql pldef
```

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

```
connect 'DSN=pldef';
Connection successful:
DSN=pldef;UID=myuser;DataStore=/mypath/install/info/DemoDataStore/pldef;
DatabaseCharacterSet=US7ASCII;ConnectionCharacterSet=US7ASCII;
DRIVER=/mypath/install/lib/libtten.so;TypeMode=0;
(Default setting AutoCommit=1)
```

Call the ttConfiguration built-in procedure to display settings, which shows you the default PL/SQL settings:

```
Command> call ttconfiguration;
...
< DataBaseCharacterSet, US7ASCII >
< DataStore, /mypath/install/info/DemoDataStore/pldef >
...
< PLSCOPE_SETTINGS, IDENTIFIERS:NONE >
< PLSQL, 1 >
< PLSQL_CCFLAGS, <NULL> >
...
< PLSQL_CONN_MEM_LIMIT, 100 >
< PLSQL_MEMORY_ADDRESS, 0x10000000 >
< PLSQL_MEMORY_SIZE, 32 >
< PLSQL_OPTIMIZE_LEVEL, 2 >
< PLSQL_TIMEOUT, 30 >
...
< UID, MYUSER >
61 rows found.
```

Example 6–2  Use ALTER SESSION to change attribute settings

This example uses ALTER SESSION statements to alter PL/SQL connection attributes, changing the settings of PLSCOPE_SETTINGS, PLSQL_CONN_MEM_LIMIT, and PLSQL_OPTIMIZE_LEVEL. It then calls the ttConfiguration built-in procedure to display the new values.

```
Command> ALTER SESSION SET PLSCOPE_SETTINGS = "IDENTIFIERS:ALL";
Session altered.
```

```
Command> ALTER SESSION SET PLSQL_CONN_MEM_LIMIT=200;
Session altered.
```
**The ttSrcScan utility**

If you have an existing PL/SQL program and want to see whether it uses PL/SQL features that TimesTen does not support, you can use the `ttSrcScan` command line utility to scan your program for unsupported functions, packages, types, type codes, attributes, modes, and constants. This is a standalone utility that can be run without TimesTen or Oracle Database being installed and runs on any platform supported by TimesTen. It reads source code files as input and creates HTML and text files as output. If the utility finds unsupported items, they are logged and alternatives are suggested. You can find the `ttSrcScan` executable in the `quickstart/sample_util` directory in your TimesTen installation.

Specify an input file or directory for the program to be scanned and an output directory for the `ttSrcScan` reports. Other options are available as well. See the README file in the `sample_util` directory for information.
Access Control for PL/SQL Programs

TimesTen has features to control database access with object-level resolution for database objects such as tables, views, materialized views, indexes, sequences, functions, procedures, and packages, for example. You can refer to "Managing Access Control" in Oracle TimesTen In-Memory Database Operations Guide for introductory information about TimesTen access control features.

This chapter introduces access control as it relates to PL/SQL users.

Note: Access control is enabled when you install TimesTen. You cannot disable it.

Topics in this chapter include the following:

- Access control for PL/SQL operations
- Access control for SQL operations
- Definer's rights and invoker's rights
- Additional access control considerations

Access control for PL/SQL operations

This section covers the following topics:

- Required privileges for PL/SQL statements and operations
- Granting and revoking privileges
- Invalidated objects

Required privileges for PL/SQL statements and operations

For PL/SQL users, access control affects the ability 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 schema other than your own. 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.
To execute a procedure or function, you must be the owner, have the `EXECUTE` privilege for the procedure or function (or for the package to which it belongs, if applicable), or have the `EXECUTE ANY PROCEDURE` privilege. This is all summarized in Table 7–1.

<table>
<thead>
<tr>
<th>Action</th>
<th>SQL statement or operation</th>
<th>Required Privilege</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a procedure, function, package</td>
<td>CREATE [OR REPLACE]</td>
<td>CREATE PROCEDURE in user's schema</td>
</tr>
<tr>
<td>definition, or package body.</td>
<td>PROCEDURE</td>
<td>Or:</td>
</tr>
<tr>
<td>Create a procedure, function, package</td>
<td>CREATE [OR REPLACE]</td>
<td>CREATE ANY PROCEDURE in any other schema</td>
</tr>
<tr>
<td>definition, or package body.</td>
<td>FUNCTION</td>
<td></td>
</tr>
<tr>
<td>Create a procedure, function, package</td>
<td>CREATE [OR REPLACE]</td>
<td></td>
</tr>
<tr>
<td>definition, or package body.</td>
<td>PACKAGE BODY</td>
<td></td>
</tr>
<tr>
<td>Alter a procedure, function, or package.</td>
<td>ALTER PROCEDURE</td>
<td>Ownership of the procedure, function, or package</td>
</tr>
<tr>
<td></td>
<td>ALTER FUNCTION</td>
<td>Or:</td>
</tr>
<tr>
<td></td>
<td>ALTER PACKAGE</td>
<td></td>
</tr>
<tr>
<td>Drop a procedure, function, package</td>
<td>DROP PROCEDURE</td>
<td>Ownership of the procedure, function, or package</td>
</tr>
<tr>
<td>definition, or package body.</td>
<td>DROP FUNCTION</td>
<td>Or:</td>
</tr>
<tr>
<td></td>
<td>DROP PACKAGE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DROP PACKAGE BODY</td>
<td></td>
</tr>
<tr>
<td>Execute a procedure or function.</td>
<td>Invoke the procedure or</td>
<td>Ownership of the procedure or function, or the package</td>
</tr>
<tr>
<td></td>
<td>function.</td>
<td>to which it belongs (if applicable)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Or:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXECUTE for the procedure or function, or for the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>package to which it belongs (if applicable)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Or:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXECUTE ANY PROCEDURE</td>
</tr>
<tr>
<td>Create a private synonym for a procedure,</td>
<td>CREATE [OR REPLACE]</td>
<td>CREATE SYNONYM in user's schema</td>
</tr>
<tr>
<td>function, or package.</td>
<td>SYNONYM</td>
<td>Or:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CREATE ANY SYNONYM in any other schema</td>
</tr>
<tr>
<td>Create a public synonym for a procedure,</td>
<td>CREATE [OR REPLACE]</td>
<td>CREATE PUBLIC SYNONYM</td>
</tr>
<tr>
<td>function, or package.</td>
<td>PUBLIC SYNONYM</td>
<td></td>
</tr>
<tr>
<td>Use a synonym to execute a procedure or</td>
<td>Invoke the procedure or</td>
<td>Privilege to execute the underlying procedure or</td>
</tr>
<tr>
<td>function.</td>
<td>function through its synonym.</td>
<td>function</td>
</tr>
<tr>
<td>Drop a private synonym for a procedure,</td>
<td>DROP SYNONYM</td>
<td>Ownership of the synonym</td>
</tr>
<tr>
<td>function, or package.</td>
<td></td>
<td>Or:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DROP ANY SYNONYM</td>
</tr>
<tr>
<td>Drop a public synonym for a procedure,</td>
<td>DROP PUBLIC SYNONYM</td>
<td>DROP PUBLIC SYNONYM</td>
</tr>
<tr>
<td>function, or package.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
See “SQL Statements” in Oracle TimesTen In-Memory Database SQL Reference for the syntax and required privileges of SQL statements discussed in this section.

Notes:

- A user who has been granted privilege to execute a procedure (or function) can execute the procedure even if he or she has no privilege 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, he 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.

- 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.

Granting and revoking privileges

Use the SQL statement GRANT to grant a privilege. Use REVOKE to revoke one.

The following example grants 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;

This example revokes the privileges:

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

Example 7–1 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 ttIsq1 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;
Access control for PL/SQL operations

> /
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 his own schema, he can execute it because he owns 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 he cannot execute it yet because he does not own it or have 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:
Access control for PL/SQL operations

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';

See “SQL Statements” in Oracle TimesTen In-Memory Database SQL Reference for information about the ALTER statements.

Example 7–2 Invalidated object

This example shows a sequence that results in an invalidated object, in this case a PL/SQL procedure, as follows:

1. A user is granted CREATE ANY PROCEDURE privilege, creates a procedure in another user’s schema, then creates a procedure in his 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 his 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 he executes his own procedure, it is implicitly recompiled and revalidated.

Instance administrator:
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.
```

**Instance administrator:**

```
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.
```

**Instance administrator:**

**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';
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.

**Instance administrator:**

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, 2009-06-04 14:51:34, 2009-06-04 16:13:00, 2009-06-04 16:13:00, INVALID, 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.

**Access control for SQL operations**

For any query or SQL DML statement executed in an anonymous block, or any SQL DDL statement executed in an **EXECUTE IMMEDIATE** statement, including all such operations discussed in this document or used in any example, it is assumed that the user has appropriate privilege to execute the statement and access the desired objects. SQL executed in a PL/SQL anonymous block requires the same privilege as when executed directly. For example, to insert rows of data into a table you own, no privilege is required. If you want to insert rows of data into a table you do not own, you must be granted **INSERT** privilege on that table or granted **INSERT ANY TABLE**.

Refer to "SQL Statements" in *Oracle TimesTen In-Memory Database SQL Reference* for details SQL statements and their required privileges.

**Definer's rights and invoker's rights**

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**). Similarly, 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*. 
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.

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.

See "Examples using the AUTHID clause" on page 5-18 for examples using definer’s and invoker’s rights.

Refer to "Invoker’s Rights and Definer’s Rights (AUTHID Property)" in Oracle Database PL/SQL Language Reference for additional information.

Additional access control considerations

This section covers the following:

- Access control for connections and connection attributes
- Access control for system views and supplied packages
- Access control for built-in procedures relating to PL/SQL

Access control for connections and connection attributes

Note the following when connecting to the database:

- Privilege to connect to a database must be explicitly granted to every user, other than the instance administrator, through the CREATE SESSION privilege. This is a system privilege so must be granted to the user either by the instance administrator or by a user with ADMIN privilege. This can be accomplished either directly or through the PUBLIC role. Refer to "Managing Access Control" in Oracle TimesTen In-Memory Database Operations Guide for additional information and examples.

- Required privileges for PL/SQL connection attributes are included in "PL/SQL connection attributes" on page 6-2.

Access control for system views and supplied packages

Note the following regarding access to system views and PL/SQL supplied packages.

- SELECT and EXECUTE privileges on various system tables, system views, PL/SQL functions, PL/SQL procedures, and PL/SQL packages are granted by default to all users through the PUBLIC role, of which all users are a member. This role is documented in "Privileges" in Oracle TimesTen In-Memory Database SQL Reference. Use the following command to see the list of these public database objects and the associated privileges:

```sql
SELECT table_name, privilege FROM sys.all_tab_privs
WHERE grantee='PUBLIC';
```
All users have SELECT privilege for the ALL_* and USER_* system views.

- EXECUTE ANY PROCEDURE does not apply to supplied packages; however, most are accessible through the PUBLIC role. Access control for PL/SQL packages provided with TimesTen is noted at the beginning of Chapter 8, "TimesTen Supplied PL/SQL Packages."

**Access control for built-in procedures relating to PL/SQL**

The ttPLSQLMemoryStats built-in procedure, which returns statistics about library cache performance and activity, can be called by any user. This procedure is documented under "ttPLSQLMemoryStats" in Oracle TimesTen In-Memory Database Reference. Also see Example 6–3 on page 6-8.
TimesTen supplies public PL/SQL packages, listed immediately below, to extend database functionality and provide PL/SQL access to SQL features. TimesTen installs these packages automatically for your use. Packages that are part of the PL/SQL language itself or are otherwise for Oracle Database internal use only are not shown here or described in this chapter.

This chapter lists and briefly describes the subprograms that comprise each package. For details on these PL/SQL packages, refer to Oracle TimesTen In-Memory Database PL/SQL Packages Reference.

- DBMS_LOB
- DBMS_LOCK
- DBMS_OUTPUT
- DBMS_PREPROCESSOR
- DBMS_RANDOM
- DBMS_SQL
- DBMS_UTILITY
- TT_DB_VERSION
- TT_STATS
- UTL_FILE
- UTL_IDENT
- UTL_RAW
- UTL_RECOMP
The `DBMS_LOB` package provides subprograms to operate on BLOBs, CLOBs, and NCLOBs, including temporary LOBs.

---

**Notes:**

- TimesTen does not support `DBMS_LOB` subprograms intended specifically for BFILEs, SecureFiles (including Database File System features), or inserting or deleting data fragments in the middle of a LOB.

- `DBMS_LOB` procedures and functions are supported for both TimesTen LOBs and passthrough LOBs, which are LOBs in Oracle Database accessed through TimesTen and exposed as TimesTen LOBs. Note, however, that `CREATETEMPORARY` can only be used to create a temporary LOB in TimesTen. If a temporary passthrough LOB is created using some other mechanism, such as SQL, `ISTEMPORARY` and `FREETEMPORARY` can be used on that LOB.

As with TimesTen local LOBs, a locator for a passthrough LOB does not remain valid past the end of the transaction.

In addition to copying from one TimesTen LOB to another, `COPY` can copy from a TimesTen LOB to a passthrough LOB, from a passthrough LOB to a TimesTen LOB, or from one passthrough LOB to another passthrough LOB. An attempt to copy a passthrough LOB to a TimesTen LOB when the passthrough LOB is larger than the TimesTen LOB size limit results in an error. (“`COPY` procedures” in *Oracle TimesTen In-Memory Database PL/SQL Packages Reference* provides examples for copying LOBs.)

See "Passthrough LOBs" on page 2-20 for related information.

---

**Table 8–1** describes the supported `DBMS_LOB` subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPEND procedures</td>
<td>Appends the contents of the source LOB to the destination LOB.</td>
</tr>
<tr>
<td>CLOSE procedures</td>
<td>Closes a previously opened LOB.</td>
</tr>
<tr>
<td>COMPARE functions</td>
<td>Compares two entire LOBs or parts of two LOBs.</td>
</tr>
<tr>
<td>CONVERTTOBLOB procedure</td>
<td>Reads character data from a source CLOB or NCLOB instance, converts the character data to the specified character set, writes the converted data to a destination BLOB instance in binary format, and returns the new offsets.</td>
</tr>
<tr>
<td>CONVERTTOCLOB procedure</td>
<td>Takes a source BLOB instance, converts the binary data in the source instance to character data using the specified character set, writes the character data to a destination CLOB or NCLOB instance, and returns the new offsets.</td>
</tr>
<tr>
<td>COPY procedures</td>
<td>Copies all or part of the source LOB to the destination LOB.</td>
</tr>
</tbody>
</table>
### Table 8–1 (Cont.) DBMS_LOB subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATETEMPORARY</td>
<td>Creates a temporary LOB in the temporary data region. Any of the durations supported by Oracle Database is permitted (SESSION, TRANSACTION, or CALL), but in TimesTen LOB duration cannot extend past the end of the transaction.</td>
</tr>
<tr>
<td>ERASE</td>
<td>Erases all or part of a LOB.</td>
</tr>
<tr>
<td>FREETEMPORARY</td>
<td>Frees a temporary LOB in the temporary data region.</td>
</tr>
<tr>
<td>GET_STORAGE_LIMIT</td>
<td>Returns the storage limit for the LOB type of the specified LOB.</td>
</tr>
<tr>
<td>GETCHUNKSIZE</td>
<td>In TimesTen, this simply returns the value 32 KB for interoperability. Do not rely on this value for performance tuning.</td>
</tr>
<tr>
<td>GETLENGTH</td>
<td>Returns the length of the LOB value, in bytes for a BLOB or characters for a CLOB or NCLOB.</td>
</tr>
<tr>
<td>INSTR</td>
<td>Returns the matching position of the ( n )th occurrence of the pattern in the LOB.</td>
</tr>
<tr>
<td>ISOPEN</td>
<td>Checks to see if the LOB was already opened using the input locator.</td>
</tr>
<tr>
<td>ISTEMPORARY</td>
<td>Checks whether the locator is pointing to a temporary LOB.</td>
</tr>
<tr>
<td>OPEN</td>
<td>Opens a LOB (persistent or temporary) in the indicated mode, read/write or read-only.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Opening a LOB is similar conceptually, but not technically, to opening a file. Opening a LOB is more like a hint regarding resources to be required.</td>
</tr>
<tr>
<td>READ</td>
<td>Reads data from the LOB starting at the specified offset.</td>
</tr>
<tr>
<td>SUBSTR</td>
<td>Returns part of the LOB value starting at the specified offset.</td>
</tr>
<tr>
<td>TRIM</td>
<td>Trims the LOB value to the specified shorter length.</td>
</tr>
<tr>
<td>WRITE</td>
<td>Writes data to the LOB from a specified offset.</td>
</tr>
<tr>
<td>WRITETEXTAPPEND</td>
<td>Writes a buffer to the end of a LOB.</td>
</tr>
</tbody>
</table>
The `DBMS_LOCK` package provides an interface to lock-management services. In the current release, TimesTen supports only the sleep feature. 

Table 8–2 describes the supported `DBMS_LOCK` subprogram.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLEEP procedure</td>
<td>This procedure suspends the session for a given duration. Specify the amount of time in seconds. The smallest supported increment is a hundredth of a second. For example:</td>
</tr>
<tr>
<td></td>
<td><code>DBMS_LOCK.SLEEP(1.95);</code></td>
</tr>
</tbody>
</table>

Notes:
- The actual sleep time may be somewhat longer than specified, depending on system activity.
- If `PLSQL_TIMEOUT` is set to a positive value that is less than this sleep time, the timeout takes effect first. Be sure that either the sleep value is less than the timeout value, or `PLSQL_TIMEOUT=0` (no timeout). See "PL/SQL connection attributes" on page 6-2 for information about `PLSQL_TIMEOUT`. 
The DBMS_OUTPUT package enables you to send messages from stored procedures and packages. The package is useful for displaying PL/SQL debugging information. Table 8–3 describes the DBMS_OUTPUT subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISABLE</td>
<td>Disables message output.</td>
</tr>
<tr>
<td>ENABLE</td>
<td>Enables message output.</td>
</tr>
<tr>
<td>GET_LINE</td>
<td>Retrieves one line from the buffer.</td>
</tr>
<tr>
<td>GET_LINES</td>
<td>Retrieves an array of lines from the buffer.</td>
</tr>
<tr>
<td>NEW_LINE</td>
<td>Terminates a line created with PUT.</td>
</tr>
<tr>
<td>PUT</td>
<td>Places a line in the buffer.</td>
</tr>
<tr>
<td>PUT_LINE</td>
<td>Places a partial line in the buffer.</td>
</tr>
</tbody>
</table>

TimesTen Supplied PL/SQL Packages   8-5
The `DBMS_PREPROCESSOR` package provides an interface to print or retrieve the source text of a PL/SQL unit after processing of conditional compilation directives. Table 8–4 describes the `DBMS_PREPROCESSOR` subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GET_POST_PROCESSED_SOURCE</code> function</td>
<td>Returns post-processed source text.</td>
</tr>
<tr>
<td><code>PRINT_POST_PROCESSED_SOURCE</code> procedure</td>
<td>Prints post-processed source text.</td>
</tr>
</tbody>
</table>
The DBMS_RANDOM package provides a built-in random number generator. 

Table 8–5 describes the DBMS_RANDOM subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INITIALIZE procedure</td>
<td>Initializes the package with a seed value (deprecated).</td>
</tr>
<tr>
<td>NORMAL function</td>
<td>Returns random numbers in a normal distribution.</td>
</tr>
<tr>
<td>RANDOM procedure</td>
<td>Generates a random number (deprecated).</td>
</tr>
<tr>
<td>SEED procedure</td>
<td>Resets the seed.</td>
</tr>
<tr>
<td>STRING function</td>
<td>Gets a random string.</td>
</tr>
<tr>
<td>TERMINATE procedure</td>
<td>Terminates the package (deprecated).</td>
</tr>
<tr>
<td>VALUE function</td>
<td>There are two overloaded versions. In the first, it gets a random number greater than or equal to 0 and less than 1, with 38 digits to the right of the decimal point (38-digit precision). In the second, it gets a random number within specified low and high limits.</td>
</tr>
</tbody>
</table>
The `DBMS_SQL` package provides an interface for using dynamic SQL to accomplish any of the following:

- Execute data manipulation language (DML) and data definition language (DDL) statements.
- Execute PL/SQL anonymous blocks.
- Call PL/SQL stored procedures and functions.

This package does not support pre-defined data types and overloads with data types that are not supported in TimesTen, such as `UROWID`, time zone features, ADT, database-level collections, and edition overloads. For more information on the supported data types in TimesTen PL/SQL, see "Understanding the data type environments" on page 3-1.

Table 8–6 describes the `DBMS_SQL` subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BIND_ARRAY</code> procedure</td>
<td>Binds a given value to a given collection.</td>
</tr>
<tr>
<td><code>BIND_VARIABLE</code> procedure</td>
<td>Binds a given value to a given variable.</td>
</tr>
<tr>
<td><code>CLOSE_CURSOR</code> procedure</td>
<td>Closes a given cursor and frees memory.</td>
</tr>
<tr>
<td><code>COLUMN_VALUE</code> procedure</td>
<td>Returns the value of the cursor element for a given position in a cursor.</td>
</tr>
<tr>
<td><code>COLUMN_VALUE_LONG</code> procedure</td>
<td>Returns a selected part of a <code>LONG</code> column that has been defined using <code>DEFINE_COLUMN_LONG</code>.</td>
</tr>
<tr>
<td></td>
<td><strong>Important</strong>: Because TimesTen does not support the <code>LONG</code> data type, attempting to use this procedure in TimesTen results in an <code>ORA-01018</code> error at runtime.</td>
</tr>
<tr>
<td><code>DEFINE_ARRAY</code> procedure</td>
<td>Defines a collection to be selected from the given cursor. Use with <code>SELECT</code> statements.</td>
</tr>
<tr>
<td><code>DEFINE_COLUMN</code> procedure</td>
<td>Defines a column to be selected from the given cursor. Use with <code>SELECT</code> statements.</td>
</tr>
<tr>
<td><code>DEFINE_COLUMN_LONG</code> procedure</td>
<td>Defines a <code>LONG</code> column to be selected from the given cursor. Use with <code>SELECT</code> statements.</td>
</tr>
<tr>
<td></td>
<td><strong>Important</strong>: Because TimesTen does not support the <code>LONG</code> data type, attempting to use the <code>COLUMN_VALUE_LONG</code> procedure in TimesTen results in an <code>ORA-01018</code> error at runtime. <code>DEFINE_COLUMN_LONG</code> would be used in conjunction with <code>COLUMN_VALUE_LONG</code>.</td>
</tr>
<tr>
<td><code>DESCRIBE_COLUMNS</code> procedure</td>
<td>Describes the columns for a cursor opened and parsed through the <code>DBMS_SQL</code> package.</td>
</tr>
<tr>
<td><code>DESCRIBE_COLUMNS2</code> procedure</td>
<td>Describes the specified column. Use as an alternative to <code>DESCRIBE_COLUMNS</code> procedure.</td>
</tr>
<tr>
<td><code>DESCRIBE_COLUMNS3</code> procedure</td>
<td>Describes the specified column. Use as an alternative to <code>DESCRIBE_COLUMNS</code> procedure.</td>
</tr>
<tr>
<td><code>EXECUTE</code> function</td>
<td>Executes a given cursor.</td>
</tr>
<tr>
<td><code>EXECUTE_AND_FETCH</code> function</td>
<td>Executes a given cursor and fetches rows.</td>
</tr>
<tr>
<td><code>FETCH_ROWS</code> function</td>
<td>Fetches a row from a given cursor.</td>
</tr>
</tbody>
</table>
### Table 8–6  (Cont.) DBMS_SQL subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IS_OPEN</strong> function</td>
<td>Returns <strong>TRUE</strong> if a given cursor is open.</td>
</tr>
<tr>
<td><strong>LAST_ERROR_POSITION</strong> function</td>
<td>Returns the byte offset in the SQL statement text where the error occurred.</td>
</tr>
<tr>
<td><strong>LAST_ROW_COUNT</strong> function</td>
<td>Returns a cumulative count of the number of rows fetched.</td>
</tr>
<tr>
<td><strong>LAST_ROW_ID</strong> function</td>
<td>Returns <strong>NULL</strong>. TimesTen does not support ROWID of the last row operated on by a DML statement.</td>
</tr>
<tr>
<td><strong>LAST_SQL_FUNCTION_CODE</strong> function</td>
<td>Returns the SQL function code for the statement.</td>
</tr>
<tr>
<td><strong>OPEN_CURSOR</strong> function</td>
<td>Returns the cursor ID number of a new cursor.</td>
</tr>
<tr>
<td><strong>PARSE</strong> procedures</td>
<td>Parses a given statement.</td>
</tr>
<tr>
<td><strong>TO_CURSOR_NUMBER</strong> function</td>
<td>Takes an opened (by <strong>OPEN</strong>) strongly or weakly typed REF CURSOR and transforms it into a <strong>DBMS_SQL</strong> cursor number.</td>
</tr>
<tr>
<td><strong>TO_REFCURSOR</strong> function</td>
<td>Takes an opened, parsed, and executed cursor (by <strong>OPEN</strong>, <strong>PARSE</strong>, and <strong>EXECUTE</strong>) and transforms or migrates it into a <strong>PL/SQL</strong> manageable REF CURSOR (a weakly typed cursor) that can be consumed by <strong>PL/SQL</strong> native dynamic SQL and switched to use native dynamic SQL.</td>
</tr>
<tr>
<td><strong>VARIABLE_VALUE</strong> procedures</td>
<td>Returns value of a named variable for a given cursor.</td>
</tr>
</tbody>
</table>
The **DBMS_UTILITY** package provides a variety of utility subprograms. Subprograms are not supported (and not listed here) for features that TimesTen does not support.

*Table 8–7* describes **DBMS_UTILITY** subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANONICALIZE procedure</td>
<td>Canonicalizes a given string.</td>
</tr>
<tr>
<td>COMMA_TO_TABLE procedure</td>
<td>Converts a comma-delimited list of names into an associative array (index-by table) of names.</td>
</tr>
<tr>
<td>COMPILE_SCHEMA</td>
<td>Compiles all procedures, functions, packages, and views in the specified database schema.</td>
</tr>
<tr>
<td>DB_VERSION procedure</td>
<td>Returns version information for the database. The procedure returns NULL for the compatibility setting because TimesTen does not support the system parameter COMPATIBLE.</td>
</tr>
<tr>
<td>FORMAT_CALL_STACK function</td>
<td>Formats the current call stack.</td>
</tr>
<tr>
<td>FORMAT_ERROR_BACKTRACE function</td>
<td>Formats the backtrace from the point of the current error to the exception handler where the error is caught.</td>
</tr>
<tr>
<td>FORMAT_ERROR_STACK function</td>
<td>Formats the current error stack.</td>
</tr>
<tr>
<td>GET_CPU_TIME function</td>
<td>Returns the current CPU time in hundredths of a second.</td>
</tr>
<tr>
<td>GET_DEPENDENCY procedure</td>
<td>Shows the dependencies on the objects passed in.</td>
</tr>
<tr>
<td>GET_ENDIANNESS function</td>
<td>Returns the endianness of your database platform.</td>
</tr>
<tr>
<td>GET_HASH_VALUE function</td>
<td>Computes a hash value for a given string.</td>
</tr>
<tr>
<td>GET_SQL_HASH function</td>
<td>Computes the hash value for a given string using the MD5 algorithm.</td>
</tr>
<tr>
<td>GET_TIME function</td>
<td>Returns the current time in hundredths of a second.</td>
</tr>
<tr>
<td>INVALIDATE procedure</td>
<td>Invalidates a database object and optionally modifies the PL/SQL compiler parameter settings for the object.</td>
</tr>
<tr>
<td>IS_BIT_SET function</td>
<td>Checks the setting of a specified bit in a RAM value.</td>
</tr>
<tr>
<td>NAME_RESOLVE procedure</td>
<td>Resolves the given name of the following form: ([a.])b.)c[dblink] Where a, b, and c are SQL identifiers and dblink is a dblink (database link). Do not use @dblink. TimesTen does not support dblinks.</td>
</tr>
</tbody>
</table>
NAME_TOKENIZE procedure

Calls the parser to parse the given name of the following form:

"a [.b [.c ]]@dblink"

Strips double quotes or converts to uppercase if there are no quotes. Ignores comments and does not perform semantic analysis. Missing values are NULL.

Do not use @dblink. TimesTen does not support dblinks.

TABLE_TO_COMMA procedures

Converts an associative array (index-by table) of names into a comma-delimited list of names.

VALIDATE procedure

Validates the object described by either owner, name and namespace, or object ID.
The \texttt{TT\_DB\_VERSION} package contains boolean constants indicating the current TimesTen release.

Table 8–8 describes the \texttt{TT\_DB\_VERSION} constants.

The primary use case for the \texttt{TT\_DB\_VERSION} and \texttt{UTL\_IDENT} packages is for conditional compilation.

\begin{table}[h]
\centering
\caption{\texttt{TT\_DB\_VERSION} constants}
\begin{tabular}{|l|l|}
\hline
Name & Description \\
\hline
\texttt{VER\_LR\_1121} & Boolean that is \texttt{TRUE} if the TimesTen version this package ships with is 11.2.1 or prior. \texttt{FALSE} for TimesTen 11g Release 2 (11.2.2) versions. \\
\hline
\texttt{VER\_LR\_1122} & Boolean that is \texttt{TRUE} if the TimesTen version this package ships with is 11.2.2 or prior. \texttt{TRUE} for TimesTen 11g Release 2 (11.2.2) versions. \\
\hline
\end{tabular}
\end{table}

See "Examples" in Oracle TimesTen In-Memory Database PL/SQL Packages Reference for an example that uses \texttt{TT\_DB\_VERSION} and \texttt{UTL\_IDENT}. 
The `TT_STATS` package provides features for collecting and comparing snapshots of TimesTen system metrics, according to the capture level. Each snapshot can consist of what TimesTen considers to be basic metrics, typical metrics, or all available metrics.

For those familiar with Oracle Database performance analysis tools, these reports are similar in nature to Oracle Automatic Workload Repository (AWR) reports.

### Table 8–9  **TT_STATS subprograms**

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CAPTURE_SNAPSHOT</code> procedure and function</td>
<td>Takes a snapshot of TimesTen metrics. The function also returns the snapshot ID.</td>
</tr>
<tr>
<td><code>DROP_SNAPSHOTS_RANGE</code> function</td>
<td>Deletes snapshots according to a specified range of snapshot IDs or timestamps.</td>
</tr>
<tr>
<td><code>GENERATE_REPORT_HTML</code> procedure</td>
<td>Produces a report in HTML format based on the data from two specified snapshots.</td>
</tr>
<tr>
<td><code>GENERATE_REPORT_TEXT</code> procedure</td>
<td>Produces a report in plain text format based on the data from two specified snapshots.</td>
</tr>
<tr>
<td><code>GET_CONFIG</code> function</td>
<td>Retrieves the value of a specified <code>TT_STATS</code> configuration parameter or the values of all configuration parameters.</td>
</tr>
<tr>
<td><code>SET_CONFIG</code> procedure</td>
<td>Sets a specified value for a specified <code>TT_STATS</code> configuration parameter.</td>
</tr>
<tr>
<td><code>SHOW_SNAPSHOTS</code> function</td>
<td>Shows the snapshot IDs and timestamps of all snapshots currently stored in the database.</td>
</tr>
</tbody>
</table>
The UTL_FILE package enables PL/SQL programs the ability to read and write operating system text files.

In the current release, this package is restricted to access of a pre-defined temporary directory only. Refer to the Oracle TimesTen In-Memory Database Release Notes for details.

**Note:** Users do not have execute permission on UTL_FILE by default. To use UTL_FILE in TimesTen, an ADMIN user or instance administrator must explicitly grant EXECUTE permission on it, such as in the following example:

```
GRANT EXECUTE ON SYS.UTL_FILE TO scott;
```

Table 8–10 describes the UTL_FILE subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCLOSE procedure</td>
<td>Closes a file.</td>
</tr>
<tr>
<td>FCLOSE_ALL procedure</td>
<td>Closes all file handles.</td>
</tr>
<tr>
<td>FCOPY procedure</td>
<td>Copies a contiguous portion of a file to a newly created file.</td>
</tr>
<tr>
<td>FFLUSH procedure</td>
<td>Physically writes all pending output to a file.</td>
</tr>
<tr>
<td>FGETATTR procedure</td>
<td>Reads and returns the attributes of a disk file.</td>
</tr>
<tr>
<td>FGETPOS procedure</td>
<td>Returns the current relative offset position (in bytes) within a file.</td>
</tr>
<tr>
<td>FOPEN function</td>
<td>Opens a file for input or output.</td>
</tr>
<tr>
<td>FOPEN_NCHAR function</td>
<td>Opens a file in Unicode for input or output.</td>
</tr>
<tr>
<td>FREMOVE procedure</td>
<td>With sufficient privilege, deletes a disk file.</td>
</tr>
<tr>
<td>FRENAME procedure</td>
<td>Renames an existing file to a new name (similar to the UNIX mv command).</td>
</tr>
<tr>
<td>FSEEK procedure</td>
<td>Adjusts the file pointer forward or backward within the file by the number of bytes specified.</td>
</tr>
<tr>
<td>GET_LINE procedure</td>
<td>Reads text from an open file.</td>
</tr>
<tr>
<td>GET_LINE_NCHAR procedure</td>
<td>Reads text in Unicode from an open file.</td>
</tr>
<tr>
<td>GET_RAW function</td>
<td>Reads a RAW string value from a file and adjusts the file pointer ahead by the number of bytes read.</td>
</tr>
<tr>
<td>IS_OPEN function</td>
<td>Determines if a file handle refers to an open file.</td>
</tr>
<tr>
<td>NEW_LINE procedure</td>
<td>Writes one or more operating system-specific line terminators to a file.</td>
</tr>
<tr>
<td>PUT procedure</td>
<td>Writes a string to a file.</td>
</tr>
<tr>
<td>PUT_LINE procedure</td>
<td>Writes a line to a file and appends an operating system-specific line terminator.</td>
</tr>
<tr>
<td>PUT_LINE_NCHAR procedure</td>
<td>Writes a Unicode line to a file.</td>
</tr>
<tr>
<td>PUT_NCHAR procedure</td>
<td>Writes a Unicode string to a file.</td>
</tr>
</tbody>
</table>
Table 8–10 (Cont.) UTL_FILE subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUT_RAW function</td>
<td>Accepts as input a RAW data value and writes the value to the output buffer.</td>
</tr>
<tr>
<td>PUTF procedure</td>
<td>This is similar to the PUT procedure, but with formatting.</td>
</tr>
<tr>
<td>PUTF_NCHAR procedure</td>
<td>This is similar to the PUT_NCHAR procedure, but with formatting.</td>
</tr>
<tr>
<td></td>
<td>Writes a Unicode string to a file with formatting.</td>
</tr>
</tbody>
</table>
The `UTL_IDENT` package indicates whether PL/SQL is running on TimesTen, an Oracle database client, an Oracle database server, or Oracle Forms. Each of these has its own version of `UTL_IDENT` with appropriate settings for the constants.

Table 8–11 shows the `UTL_IDENT` settings for TimesTen.

The primary use case for the `UTL_IDENT` package is for conditional compilation, resembling the following:

```sql
$if utl_ident.is_oracle_server $then
    [...Run code supported for Oracle Database...]
$elsif utl_ident.is_timesten $then
    [...code supported for TimesTen Database...]
$end
```

See "Examples" in Oracle TimesTen In-Memory Database PL/SQL Packages Reference for an example that uses `TT_DB_VERSION` and `UTL_IDENT`.

### Table 8–11  `UTL_IDENT` Constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS_ORACLE_CLIENT</td>
<td>BOOLEAN set to FALSE</td>
</tr>
<tr>
<td>IS_ORACLE_SERVER</td>
<td>BOOLEAN set to FALSE</td>
</tr>
<tr>
<td>IS_ORACLE_FORMS</td>
<td>BOOLEAN set to FALSE</td>
</tr>
<tr>
<td>IS_TIMESTEN</td>
<td>BOOLEAN set to TRUE</td>
</tr>
</tbody>
</table>
The UTL_RAW package provides SQL functions for manipulating RAW data types. Table 8–12 describes the UTL_RAW subprograms.

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT_AND function</td>
<td>Performs bitwise &quot;and&quot; of two RAW values and returns the resulting RAW.</td>
</tr>
<tr>
<td>BIT_COMPLEMENT function</td>
<td>Performs bitwise &quot;complement&quot; of a RAW value and returns the resulting RAW.</td>
</tr>
<tr>
<td>BIT_OR function</td>
<td>Performs bitwise &quot;or&quot; of two RAW values and returns the resulting RAW.</td>
</tr>
<tr>
<td>BIT_XOR function</td>
<td>Performs bitwise &quot;exclusive or&quot; of two RAW values and returns the resulting RAW.</td>
</tr>
<tr>
<td>CAST_FROM_BINARY_DOUBLE function</td>
<td>Returns the RAW binary representation of a BINARY_DOUBLE value.</td>
</tr>
<tr>
<td>CAST_FROM_BINARY_FLOAT function</td>
<td>Returns the RAW binary representation of a BINARY_FLOAT value.</td>
</tr>
<tr>
<td>CAST_FROM_BINARY_INTEGER function</td>
<td>Returns the RAW binary representation of a BINARY_INTEGER value.</td>
</tr>
<tr>
<td>CAST_FROM_NUMBER function</td>
<td>Returns the RAW binary representation of a NUMBER value.</td>
</tr>
<tr>
<td>CAST_TO_BINARY_DOUBLE function</td>
<td>Casts the RAW binary representation of a BINARY_DOUBLE value into a BINARY_DOUBLE.</td>
</tr>
<tr>
<td>CAST_TO_BINARY_FLOAT function</td>
<td>Casts the RAW binary representation of a BINARY_FLOAT value into a BINARY_FLOAT.</td>
</tr>
<tr>
<td>CAST_TO_BINARY_INTEGER function</td>
<td>Casts the RAW binary representation of a BINARY_INTEGER value into a BINARY_INTEGER.</td>
</tr>
<tr>
<td>CAST_TO_NUMBER function</td>
<td>Casts the RAW binary representation of a NUMBER value into a NUMBER.</td>
</tr>
<tr>
<td>CAST_TO_NVARCHAR2 function</td>
<td>Casts a RAW value represented using n data bytes into an NVARCHAR2 value with n data bytes.</td>
</tr>
<tr>
<td>CAST_TO_RAW function</td>
<td>Casts a VARCHAR2 value represented using n data bytes into a RAW with n data bytes.</td>
</tr>
<tr>
<td>CAST_TO_VARCHAR2 function</td>
<td>Casts a RAW value represented using n data bytes into a VARCHAR2 value with n data bytes.</td>
</tr>
<tr>
<td>COMPARE function</td>
<td>Compares two RAW values.</td>
</tr>
<tr>
<td>CONCAT function</td>
<td>Concatenates up to 12 RAW values into a single RAW value.</td>
</tr>
<tr>
<td>CONVERT function</td>
<td>Converts a RAW value from one character set to another and returns the resulting RAW.</td>
</tr>
<tr>
<td>COPIES function</td>
<td>Copies a RAW value a specified number of times and returns the concatenated RAW value.</td>
</tr>
<tr>
<td>LENGTH function</td>
<td>Returns the length in bytes of a RAW value.</td>
</tr>
</tbody>
</table>
# Table 8–12  (Cont.) UTL_RAW subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVERLAY function</td>
<td>Overlays the specified portion of a target RAW value with an overlay RAW value, starting from a specified byte position and proceeding for a specified number of bytes.</td>
</tr>
<tr>
<td>REVERSE function</td>
<td>Reverses a byte-sequence in a RAW value.</td>
</tr>
<tr>
<td>SUBSTR function</td>
<td>Returns a substring of a RAW value for a specified number of bytes from a specified starting position.</td>
</tr>
<tr>
<td>TRANSLATE function</td>
<td>Translates the specified bytes from an input RAW value according to the bytes in a specified translation RAW value.</td>
</tr>
<tr>
<td>TRANSLITERATE function</td>
<td>Converts the specified bytes from an input RAW value according to the bytes in a specified transliteration RAW value.</td>
</tr>
<tr>
<td>XRANGE function</td>
<td>Returns a RAW value containing the succession of one-byte encodings beginning and ending with the specified byte-codes.</td>
</tr>
</tbody>
</table>

UTL_RECOMP package recompiles invalid PL/SQL modules. This is particularly useful after a major-version upgrade that typically invalidates all PL/SQL objects. Table 8–13 describes the UTL_RECOMP subprograms.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECOMP_PARALLEL</td>
<td>Recompiles invalid objects in a given schema, or all invalid objects in the database, in parallel.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: Because TimesTen does not support DBMS_SCHEDULER, the number of recompile threads to run in parallel is always 1, regardless of what the user specifies. Therefore there is no effective difference between RECOMP_PARALLEL and RECOMP_SERIAL in TimesTen.</td>
</tr>
<tr>
<td>RECOMP_SERIAL</td>
<td>Recompiles invalid objects in a given schema, or all invalid objects in the database, serially.</td>
</tr>
</tbody>
</table>

Important: To use this package, you must be the instance administrator and specify SYS.UTL_RECOMP.
The purpose of this chapter is to summarize PL/SQL language elements and features and compare their support in TimesTen to their support in Oracle Database. In the Oracle Database documentation, many of these features are covered in "PL/SQL Language Elements" in Oracle Database PL/SQL Language Reference.

Table 9–1 PL/SQL language element and feature support in TimesTen

<table>
<thead>
<tr>
<th>Feature name</th>
<th>Description</th>
<th>Supported?</th>
<th>Example/comment</th>
</tr>
</thead>
</table>
| ALTER {PROCEDURE | FUNCTION | PACKAGE} statements                                | Recompiles a PL/SQL procedure, function, or package. | Y Syntax and semantics are the same as in Oracle Database.  
For information about these statements, see "SQL Statements" in Oracle TimesTen In-Memory Database SQL Reference. |
| ALTER SESSION statement          | Changes session parameters dynamically.                | Y          | In TimesTen you can use ALTER SESSION to set some PL/SQL connection attributes as discussed in "PL/SQL connection attributes" on page 6-2.  
For more information on this statement in TimesTen, see "ALTER SESSION" in Oracle TimesTen In-Memory Database SQL Reference. |
| Assignment statement             | Sets current value of a variable, parameter, or element. | Y          | See "PL/SQL variables and constants" on page 2-2.                               |
| AUTONOMOUS_TRANSACTION pragma    | Marks a routine as autonomous.                        | N          | TimesTen does not support autonomous transactions.                              |
| Block declaration                | Declares a block, the basic unit of a PL/SQL source program. | Y          | See "PL/SQL blocks" on page 2-1.                                                |
| BULK COLLECT clause             | Select multiple rows.                                 | Y          | This clause can be used with the SELECT statement in PL/SQL to retrieve rows without using a cursor.  
See "FORALL and BULK COLLECT operations" on page 2-14 and "Examples using FORALL and BULK COLLECT" on page 5-9. |
<table>
<thead>
<tr>
<th>Feature name</th>
<th>Description</th>
<th>Supported?</th>
<th>Example/comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL statement</td>
<td>Executes a routine from within SQL.</td>
<td>Y</td>
<td>In TimesTen, use the CALL statement to execute PL/SQL stored procedures and functions, as in Oracle Database, or TimesTen built-in procedures. (For TimesTen built-in procedures, use EXECUTE IMMEDIATE if CALL is inside PL/SQL.) See &quot;How to execute PL/SQL procedures and functions&quot; on page 2-8 and Example 5–19, &quot;Using EXECUTE IMMEDIATE to call ttConfiguration&quot; on page 5-16.</td>
</tr>
<tr>
<td>CASE statement</td>
<td>Evaluates an expression, compares it against several values, and takes action according to the comparison that is true.</td>
<td>Y</td>
<td>See &quot;PL/SQL control structures&quot; on page 2-6.</td>
</tr>
<tr>
<td>CLOSE statement</td>
<td>Closes a cursor or cursor variable.</td>
<td>Y</td>
<td>See Example 2–13, &quot;Using a cursor to retrieve information about an employee&quot; on page 2-21 (among others).</td>
</tr>
<tr>
<td>Collection definition</td>
<td>Specifies a collection, which is an ordered group of elements that are all of the same type.</td>
<td>Y</td>
<td>Examples include associative arrays (index-by tables or PL/SQL tables), nested tables, and varrays. TimesTen supports all three of these collection types in PL/SQL programs, but supports only associative arrays as bound parameters between PL/SQL and applications written in other languages (such as OCI or JDBC). See &quot;Using collections in PL/SQL&quot; on page 3-5.</td>
</tr>
<tr>
<td>Collection methods</td>
<td>Built-in subprograms that operate on collections and are called using &quot;dot&quot; notation.</td>
<td>Y</td>
<td>See &quot;Collection Methods&quot; in Oracle Database PL/SQL Language Reference. Examples include COUNT, DELETE, EXISTS, EXTEND, FIRST, LAST, LIMIT, NEXT, PRIOR, and TRIM.</td>
</tr>
<tr>
<td>Comments</td>
<td>Text included within your code for explanatory purposes.</td>
<td>Y</td>
<td>Single-line and multi-line comments are supported.</td>
</tr>
<tr>
<td>COMMIT statement</td>
<td>Ends the current transaction and makes permanent all changes performed in the transaction.</td>
<td>Y</td>
<td>See &quot;COMMIT&quot; in Oracle TimesTen In-Memory Database SQL Reference. Important: COMMIT and ROLLBACK statements close all cursors in TimesTen.</td>
</tr>
<tr>
<td>Connection attributes</td>
<td>Equivalent to initialization parameters in Oracle Database.</td>
<td>Y</td>
<td>See &quot;PL/SQL connection attributes&quot; on page 6-2. Also see &quot;PL/SQL first connection attributes&quot; and &quot;PL/SQL general connection attributes&quot; in Oracle TimesTen In-Memory Database Reference.</td>
</tr>
<tr>
<td>Constant and variable declarations</td>
<td>Specify constants and variables to be used in PL/SQL code, in the declarative part of any PL/SQL block, subprogram, or package.</td>
<td>Y</td>
<td>See &quot;PL/SQL variables and constants&quot; on page 2-2.</td>
</tr>
<tr>
<td>Feature name</td>
<td>Description</td>
<td>Supported?</td>
<td>Example/comment</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CONTINUE statement</td>
<td>Exits the current iteration of a loop and transfers control to the next iteration.</td>
<td>Y</td>
<td>See “CONTINUE statement” on page 2-7.</td>
</tr>
<tr>
<td>CREATE FUNCTION statement</td>
<td>Creates a PL/SQL function.</td>
<td>Y</td>
<td>CREATE FUNCTION is supported in TimesTen, but the AS LANGUAGE, AS EXTERNAL, and PIPELINED clauses are not supported. See ”PL/SQL procedures and functions” on page 2-21. Also see “CREATE FUNCTION” in Oracle TimesTen In-Memory Database SQL Reference. You are not required to run DBMSSTDX.SQL in TimesTen.</td>
</tr>
<tr>
<td>CREATE LIBRARY statement</td>
<td>Creates a schema object associated with an operating system shared library.</td>
<td>N</td>
<td>CREATE LIBRARY is not supported in TimesTen.</td>
</tr>
<tr>
<td>CREATE PACKAGE statement</td>
<td>These statements are used together to create a PL/SQL package definition and package body.</td>
<td>Y</td>
<td>Syntax and semantics are the same as in Oracle Database. See ”PL/SQL packages” on page 2-25. Also see “CREATE PACKAGE” and “CREATE PACKAGE BODY” in Oracle TimesTen In-Memory Database SQL Reference. You are not required to run DBMSSTDX.SQL in TimesTen.</td>
</tr>
<tr>
<td>CREATE PROCEDURE statement</td>
<td>Creates a PL/SQL procedure.</td>
<td>Y</td>
<td>CREATE PROCEDURE is supported in TimesTen, but the AS LANGUAGE and AS EXTERNAL clauses are not supported. See ”PL/SQL procedures and functions” on page 2-21. Also see ”CREATE PROCEDURE” in Oracle TimesTen In-Memory Database SQL Reference. Note: You are not required to run DBMSSTDX.SQL in TimesTen.</td>
</tr>
<tr>
<td>CREATE TYPE statement</td>
<td>Creates a user-defined object type or collection type.</td>
<td>N</td>
<td>TimesTen does not support CREATE TYPE.</td>
</tr>
<tr>
<td>CURRENT_DATE function</td>
<td>Returns the current date in the session time zone.</td>
<td>Y</td>
<td>In TimesTen this returns the current date in UTC (universal time). TimesTen does not support local time zones.</td>
</tr>
</tbody>
</table>
Cursor attributes

Append to the cursor or
cursor variable to return
useful information about
the execution of a data
manipulation statement.

The implicit cursor (SQL) has
additional attributes: %BULK_ROWCOUNT
and %BULK_EXCEPTIONS.
See "Using the %ROWCOUNT and
%NOTFOUND attributes" on page 5-8
and "Using FORALL with
SQL%BULK_ROWCOUNT" on
page 5-9. Also see "Named Cursor
Attribute" in
Oracle Database PL/SQL
Language Reference.

Cursor declaration

Declares a cursor. To
execute a multi-row query,
TimesTen opens an
unnamed work area that
stores processing
information. A cursor lets
you name the work area,
access the information, and
process the rows
individually.

See "Use of cursors in PL/SQL
programs" on page 2-21.

Cursor variables (REF CURSORs)

Act as handles to cursors
over SQL result sets.

TimesTen supports OUT REF
CURSORs, one per statement.
See "PL/SQL REF CURSORs" on
page 3-7.

Database links (dblinks)

A pointer that defines a
one-way communication
path from an Oracle
database server to another
database server.

TimesTen does not support database
links.

DELETE statement

Deletes rows from a table.

See "DELETE" in
Oracle TimesTen
In-Memory Database SQL Reference.

DROP ( PROCEDURE | FUNCTION | PACKAGE )
statement

Removes a PL/SQL
procedure, function, or
package, as specified.

Syntax and semantics are the same as
in Oracle Database.
You can refer to information about
these statements in "SQL Statements"
in
Oracle TimesTen In-Memory Database
SQL Reference.

Error reporting

(This is self-explanatory.)

TimesTen applications report errors
using Oracle Database error codes
instead of TimesTen error codes. The
error messages that accompany the
error codes are either TimesTen error
messages or Oracle Database error
messages.

EXCEPTION_INIT pragma

Associates a user-defined
exception with a TimesTen
error number.

See "EXCEPTION_INIT Pragma" in
Oracle Database PL/SQL Language
Reference.
## Table 9–1 (Cont.) PL/SQL language element and feature support in TimesTen

<table>
<thead>
<tr>
<th>Feature name</th>
<th>Description</th>
<th>Supported?</th>
<th>Example/comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exception definition</td>
<td>Specifies an exception, which is a runtime error or warning condition. Can be predefined or user-defined.</td>
<td>Y</td>
<td>Predefined conditions are raised implicitly. User-defined exceptions are raised explicitly by the RAISE statement. To handle raised exceptions, write separate routines called exception handlers. See Chapter 4, &quot;Errors and Exception Handling&quot;.</td>
</tr>
<tr>
<td><strong>EXECUTE IMMEDIATE statement</strong></td>
<td>Builds and executes a dynamic SQL statement.</td>
<td>Y</td>
<td>TimesTen supports this as Oracle Database does to execute a SQL DML or DDL statement, execute a PL/SQL anonymous block, or call a PL/SQL stored procedure or function. See &quot;Dynamic SQL in PL/SQL (EXECUTE IMMEDIATE statement)&quot; on page 2-12. In TimesTen, the EXECUTE IMMEDIATE statement can also be used to execute TimesTen built-in procedures and TimesTen-specific SQL features (such as SELECT FIRST).</td>
</tr>
<tr>
<td>Executing PL/SQL from client applications</td>
<td>(This is self-explanatory.)</td>
<td>Y</td>
<td>TimesTen supports ODBC, OCI, Pro*C/C++, TTClasses (a set of TimesTen C++ classes), JDBC, and ODP.NET.</td>
</tr>
<tr>
<td>Executing PL/SQL from SQL</td>
<td>(This is self-explanatory.)</td>
<td>N</td>
<td>In TimesTen, you cannot execute PL/SQL from a SQL statement (static or dynamic).</td>
</tr>
<tr>
<td><strong>EXIT statement</strong></td>
<td>Exits a loop and transfers control to the end of the loop.</td>
<td>Y</td>
<td>See Example 6–2, &quot;Use ALTER SESSION to change attribute settings&quot; on page 6-7 (among others).</td>
</tr>
<tr>
<td>Expression definition</td>
<td>Specifies an expression, which is a combination of operands (variables, constants, literals, operators, and so on) and operators. The simplest expression is a single variable.</td>
<td>Y</td>
<td>See &quot;Expressions&quot; in Oracle Database PL/SQL Language Reference.</td>
</tr>
<tr>
<td><strong>FETCH statement</strong></td>
<td>Retrieves rows of data from the result set of a multi-row query.</td>
<td>Y</td>
<td>See Example 2–13, &quot;Using a cursor to retrieve information about an employee&quot; on page 2-21 (among others).</td>
</tr>
<tr>
<td><strong>FORALL statement</strong></td>
<td>Bulk-binds input collections before sending them to the SQL engine.</td>
<td>Y</td>
<td>See &quot;FORALL and BULK COLLECT operations&quot; on page 2-14.</td>
</tr>
</tbody>
</table>
**Table 9–1 (Cont.) PL/SQL language element and feature support in TimesTen**

<table>
<thead>
<tr>
<th>Feature name</th>
<th>Description</th>
<th>Supported?</th>
<th>Example/comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function declaration and definition</td>
<td>Specifies a subprogram or stored program that can be declared and defined in a PL/SQL block or package and returns a single value.</td>
<td>Y</td>
<td>In TimesTen, a stored function or procedure can be executed in an anonymous block or through a CALL statement, but not from any other SQL statement. See &quot;How to execute PL/SQL procedures and functions&quot; on page 2-8. Use the <code>CREATE FUNCTION</code> statement in TimesTen SQL to create stored functions. See &quot;PL/SQL procedures and functions&quot; on page 2-21. Also see &quot;CREATE FUNCTION&quot; in Oracle TimesTen In-Memory Database SQL Reference. Also refer to the table entry below for &quot;Procedure declaration and definition&quot;.</td>
</tr>
<tr>
<td>GOTO statement</td>
<td>Branches unconditionally to a statement label or block label.</td>
<td>Y</td>
<td>See “GOTO Statement” in Oracle Database PL/SQL Language Reference.</td>
</tr>
<tr>
<td>IF statement</td>
<td>Executes or skips a sequence of statements depending on the value of the associated boolean expression.</td>
<td>Y</td>
<td>See &quot;Conditional control&quot; on page 2-6.</td>
</tr>
<tr>
<td>Initialization parameters</td>
<td>Initial parameter settings for an Oracle database.</td>
<td>n/a</td>
<td>TimesTen connection attributes are equivalent. See that entry above.</td>
</tr>
<tr>
<td>INLINE pragma</td>
<td>Specifies whether a subprogram call is to be inline.</td>
<td>Y</td>
<td>See &quot;INLINE Pragma&quot; in Oracle Database PL/SQL Language Reference.</td>
</tr>
<tr>
<td>INSERT statement</td>
<td>Inserts one or more rows of data into a table.</td>
<td>Y</td>
<td>See &quot;Example using the INSERT statement&quot; on page 5-2. Also see &quot;INSERT&quot; in Oracle TimesTen In-Memory Database SQL Reference.</td>
</tr>
<tr>
<td>Literal declaration</td>
<td>Specifies a numeric, character string, or boolean value.</td>
<td>Y</td>
<td>Examples: Numeric literal: 135 String literal: 'TimesTen'</td>
</tr>
<tr>
<td>LOCK TABLE statement</td>
<td>Locks database tables in a specified lock mode.</td>
<td>N</td>
<td>TimesTen does not support the LOCK TABLE statement.</td>
</tr>
<tr>
<td>LOOP statement</td>
<td>Executes a sequence of statements multiple times. Can be used, for example, in implementing a FOR loop or WHILE loop.</td>
<td>Y</td>
<td>See Example 2-8, &quot;Using a WHILE loop&quot; on page 2-7. Also see &quot;Basic LOOP Statement&quot; in Oracle Database PL/SQL Language Reference.</td>
</tr>
<tr>
<td>MERGE statement</td>
<td>Allows you to select rows from one or more sources for update or insertion into a target table.</td>
<td>Y</td>
<td>See &quot;MERGE&quot; in Oracle TimesTen In-Memory Database SQL Reference.</td>
</tr>
<tr>
<td>Native dynamic SQL execution</td>
<td>Processes most dynamic SQL statements through the EXECUTE IMMEDIATE statement.</td>
<td>Y</td>
<td>See the EXECUTE IMMEDIATE entry above.</td>
</tr>
</tbody>
</table>
### Table 9–1 (Cont.) PL/SQL language element and feature support in TimesTen

<table>
<thead>
<tr>
<th>Feature name</th>
<th>Description</th>
<th>Supported?</th>
<th>Example/comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-ASCII names</td>
<td>Use of non-ASCII character sets in names of tables, columns, procedures, functions, and other database objects.</td>
<td>N</td>
<td>This is not supported in TimesTen.</td>
</tr>
<tr>
<td>Non-uppercase names</td>
<td>Use of quoted non-uppercase names of tables, columns, procedures, functions, and other database objects.</td>
<td>N</td>
<td>This is not supported in TimesTen (such as lowercase and MixedCase). For example, you cannot have the following: create or replace procedure 'MixedCase' as begin ... end;</td>
</tr>
<tr>
<td>NULL statement</td>
<td>This is a no-operation statement. Control is passed to the next statement without any action.</td>
<td>Y</td>
<td>See &quot;NULL Statement&quot; in Oracle Database PL/SQL Language Reference.</td>
</tr>
<tr>
<td>Object type declaration</td>
<td>Specifies a custom object type, which is created in SQL and stored in the database.</td>
<td>N</td>
<td>TimesTen does not support object types at the database level. For example, CREATE TYPE is not supported.</td>
</tr>
<tr>
<td>OPEN statement</td>
<td>Executes the query associated with a cursor. Allocates database resources to process the query, and identifies the result set.</td>
<td>Y</td>
<td>See Example 2–13, &quot;Using a cursor to retrieve information about an employee&quot; on page 2-21.</td>
</tr>
<tr>
<td>OPEN-FOR statement</td>
<td>Executes the SELECT statement associated with a cursor variable (REF CURSOR). Positions the cursor variable before the first row in the result set.</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Package declaration</td>
<td>Specifies a package, which is a database object that groups logically related PL/SQL types, items, and subprograms.</td>
<td>Y</td>
<td>Use SQL statements CREATE PACKAGE and CREATE PACKAGE BODY. See &quot;PL/SQL packages&quot; on page 2-25. Also see &quot;SQL Statements&quot; in Oracle TimesTen In-Memory Database SQL Reference for information about the CREATE statements.</td>
</tr>
</tbody>
</table>
**Table 9–1 (Cont.) PL/SQL language element and feature support in TimesTen**

<table>
<thead>
<tr>
<th>Feature name</th>
<th>Description</th>
<th>Supported?</th>
<th>Example/comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedure declaration and definition</td>
<td>Specifies a subprogram or stored program that can be declared and defined in a PL/SQL block or package and performs a specific action.</td>
<td>Y</td>
<td>In TimesTen, a stored procedure or function can be executed in an anonymous block or through a CALL statement, but not from any other SQL statement. See &quot;How to execute PL/SQL procedures and functions&quot; on page 2-8. Use the CREATE PROCEDURE statement in TimesTen SQL to create stored procedures. See &quot;PL/SQL procedures and functions&quot; on page 2-21. Also see &quot;CREATE PROCEDURE&quot; in Oracle TimesTen In-Memory Database SQL Reference. Also refer to the table entry above for &quot;Function declaration and definition&quot;.</td>
</tr>
<tr>
<td>RAISE statement</td>
<td>Stops normal execution of a PL/SQL block or subprogram and transfers control to an exception handler.</td>
<td>Y</td>
<td>See &quot;Using the RAISE statement&quot; on page 4-4.</td>
</tr>
<tr>
<td>Record definition</td>
<td>Defines a record, which is a composite variable that stores data values of different types (similar to a database row).</td>
<td>Y</td>
<td>See &quot;Using records in PL/SQL&quot; on page 3-5.</td>
</tr>
<tr>
<td>RESTRICT_REFERENCES pragma</td>
<td>Asserts that a subprogram (usually a function) in a package specification or object type specification does not read or write database tables or package variables.</td>
<td>N</td>
<td>TimesTen ignores this.</td>
</tr>
<tr>
<td>Result cache</td>
<td>This is a mechanism for caching the results of PL/SQL functions in a shared global area (SGA) that is available to every session that runs your application.</td>
<td>N</td>
<td>TimesTen does not support this.</td>
</tr>
<tr>
<td>RETURN statement</td>
<td>Immediately completes the execution of a subprogram and returns control to the invoker. Execution resumes with the statement following the subprogram call.</td>
<td>Y</td>
<td>See &quot;RETURN Statement&quot; in Oracle Database PL/SQL Language Reference.</td>
</tr>
<tr>
<td>RETURNING INTO clause</td>
<td>Specifies the variables in which to store the values returned by the statement to which the clause belongs.</td>
<td>Y</td>
<td>See &quot;RETURNING INTO clause&quot; on page 2-15 and &quot;Examples using RETURNING INTO&quot; on page 5-16.</td>
</tr>
<tr>
<td>Feature name</td>
<td>Description</td>
<td>Supported?</td>
<td>Example/comment</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ROLLBACK statement</td>
<td>Undoes database changes made during the current transaction.</td>
<td>Y</td>
<td>See &quot;ROLLBACK&quot; in Oracle TimesTen In-Memory Database SQL Reference. Important: COMMIT and ROLLBACK statements close all cursors in TimesTen.</td>
</tr>
<tr>
<td>%ROWTYPE attribute</td>
<td>Provides a record type that represents a row in a database table.</td>
<td>Y</td>
<td>See Example 2–2, &quot;Assigning values to variables with the assignment operator&quot; on page 2-3.</td>
</tr>
<tr>
<td>SAVEPOINT statement</td>
<td>Names and marks the current point in the processing of a transaction.</td>
<td>N</td>
<td>TimesTen does not support savepoints.</td>
</tr>
<tr>
<td>SELECT INTO statement</td>
<td>Retrieves values from one row of a table (SELECT) and then stores the values in either variables or a record. With the BULK COLLECT clause (discussed above), this statement retrieves an entire result set at one time.</td>
<td>Y</td>
<td>See Example 2–3, &quot;Using SELECT INTO to assign values to variables&quot; on page 2-4. Also see &quot;Query Result Set Processing&quot; in Oracle Database PL/SQL Language Reference.</td>
</tr>
<tr>
<td>SERIALLY_REUSABLE pragma</td>
<td>Indicates that package state is required only for the duration of one call to the server.</td>
<td>N</td>
<td>TimesTen does not support the SERIALLY_REUSABLE pragma.</td>
</tr>
<tr>
<td>SET TRANSACTION statement</td>
<td>Begins a read-only or read and write transaction.</td>
<td>N</td>
<td>TimesTen does not support the SET TRANSACTION statement.</td>
</tr>
<tr>
<td>SOUNDEX SQL function</td>
<td>Returns a character string containing the phonetic representation of a char.</td>
<td>Y</td>
<td>See &quot;SOUNDEX&quot; in Oracle TimesTen In-Memory Database SQL Reference.</td>
</tr>
<tr>
<td>SQL cursor</td>
<td>Either explicit or implicit, is used to handle the result set of a SELECT statement.</td>
<td>Y</td>
<td>See &quot;Use of cursors in PL/SQL programs&quot; on page 2-21.</td>
</tr>
<tr>
<td>SQLCODE function</td>
<td>Returns number code of the most recent exception.</td>
<td>Y</td>
<td>Given the same error condition, error codes returned by the built-in function SQLCODE are the same in TimesTen as in Oracle Database, although the SQLERRM returns may be different. This is also noted in &quot;TimesTen error messages and SQL codes&quot; on page 4-9.</td>
</tr>
<tr>
<td>SQLERRM function</td>
<td>Returns the error message associated with the error-number argument.</td>
<td>Y</td>
<td>Given the same error condition, error messages returned by the built-in function SQLERRM are not necessarily the same in TimesTen as in Oracle Database, although SQLCODE returns are the same. This is also noted in &quot;TimesTen error messages and SQL codes&quot; on page 4-9.</td>
</tr>
<tr>
<td>Supplied packages</td>
<td>These are PL/SQL packages supplied with the database.</td>
<td>Y</td>
<td>TimesTen provides a subset of the Oracle Database PL/SQL supplied packages.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>See Chapter 8, &quot;TimesTen Supplied PL/SQL Packages&quot;</td>
</tr>
<tr>
<td>Feature name</td>
<td>Description</td>
<td>Supported?</td>
<td>Example/comment</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>System tables and views</td>
<td>These are tables and views provided with the database for administrative purposes.</td>
<td>Y</td>
<td>TimesTen supports a subset of the Oracle Database system tables and views.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>See &quot;System Tables&quot; in Oracle TimesTen In-Memory Database System Tables and Views Reference.</td>
</tr>
<tr>
<td>SYSTIMESTAMP</td>
<td>Returns the system date, including fractional seconds and time zone, of the system on which the database resides.</td>
<td>N</td>
<td>TimesTen cannot support this because the return type, TIMESTAMP WITH TIME ZONE, is not currently supported. As an alternative, you can use SELECT tt_sysdate FROM dual using dynamic SQL: declare ts timestamp; begin execute immediate 'select tt_sysdate from dual' into ts; -- ts has millisecond resolution end;</td>
</tr>
<tr>
<td>Triggers</td>
<td>These are procedures that are stored in the database and activated when specific conditions occur, such as adding a row to a table.</td>
<td>N</td>
<td>TimesTen does not support triggers, but you can achieve similar functionality using XLA. See &quot;XLA and TimesTen Event Management&quot; in Oracle TimesTen In-Memory Database C Developer's Guide.</td>
</tr>
<tr>
<td>ttPLSQLMemoryStats built-in procedure</td>
<td>Returns statistics about library cache performance and activity.</td>
<td>Y</td>
<td>See &quot;ttPLSQLMemoryStats&quot; in Oracle TimesTen In-Memory Database Reference.</td>
</tr>
<tr>
<td>%TYPE attribute</td>
<td>Lets you use the data type of a field, record, nested table, database column, or variable in your own declarations, rather than hardcoding the data type. Particularly useful when declaring variables, fields, and parameters that refer to database columns.</td>
<td>Y</td>
<td>See &quot;PL/SQL variables and constants&quot; on page 2-2.</td>
</tr>
<tr>
<td>UPDATE statement</td>
<td>Updates the values of one or more columns in all rows of a table or in rows that satisfy a search condition.</td>
<td>Y</td>
<td>See &quot;UPDATE&quot; in Oracle TimesTen In-Memory Database SQL Reference.</td>
</tr>
<tr>
<td>V$LIBRARYCACHE system view</td>
<td>In Oracle Database, use this system view to return statistics about library cache performance and activity.</td>
<td>n/a</td>
<td>In TimesTen, use the ttPLSQLMemoryStats built-in procedure to retrieve the same statistical information. See that entry above.</td>
</tr>
</tbody>
</table>
## Index

### Symbols

- `%BULK_EXCEPTIONS attribute`, 2-14, 5-12
- `%BULK_ROWCOUNT attribute`, 5-9
- `%FOUND attribute`, 9-4
- `%ISOPEN attribute`, 9-4
- `%NOTFOUND attribute`, 5-7, 9-4
- `%ROWCOUNT attribute`, 5-8, 9-4
- `%ROWTYPE attribute`, 2-4
- `%TYPE attribute`, 2-2

### A

- access control
  - connections, 7-8
  - for TimesTen built-in functions, 7-9
  - granting and revoking privileges, 7-3
  - impact for PL/SQL, 7-1
  - privileges for procedures, functions, packages, 7-1
  - SQL operations, 7-7
  - supplied packages and system views, 7-8
- ALTER SESSION, 6-7
- anonymous blocks, 2-1
- array binding
  - associative array binding, 3-6
  - bulk binding, 2-14
- associative arrays (index-by tables)
  - binding, 3-6
  - example, 5-5
  - using, 3-5
- audiences for this document, 1-4
- AUTHID clause, 5-18, 7-7

### B

- BINARY_INTEGER type, 3-3
- bind parameter—see parameters
- bind variables—see parameters
- blocks, 2-1
- built-in functions (TimesTen)
  - access control, 7-9
  - calling via EXECUTE IMMEDIATE, 2-12, 5-16
- bulk binding
  - `%BULK_EXCEPTIONS attribute`, 2-14, 5-12
  - BULK COLLECT INTO with cursors, 5-11
  - BULK COLLECT INTO with queries, 5-10
  - BULK COLLECT INTO with RETURNING INTO, 5-17
  - examples, 5-9
- FORALL statement, 2-14
- overview, 2-14
- SAVE EXCEPTIONS clause, 2-14, 5-12

### C

- cache features, use from PL/SQL, 2-16
- CALL statement, calling functions and procedures, 2-8
- CASE statement, 2-6
- collections, 3-5
- compilation
  - conditional compilation, use of PLSQL_CCFLAGS, 6-4
  - conditional, use of UTL_IDENT and TT_DB_VERSION, 8-16
  - DBMS_UTILITY.COMPILE_SCHEMA, 8-10
  - DBMS_UTILITY.INVALIDATE, optionally modify compiler parameter settings, 8-10
  - implicit, 7-3
  - PLSCOPE_SETTINGS for compilation
  - cross-reference information, 6-3
  - PLSQL_OPTIMIZE_LEVEL for optimization level, 6-5
  - UTL_RECOMP package to recompile invalid modules, 8-19
- components of PL/SQL, overview, 1-2
- composite data types, 3-4
- conditional control, 2-6
- connection attributes
  - first connection attributes, 6-2
  - general connection attributes, 6-2
  - PLSCOPE_SETTINGS attribute, 6-3
  - PLSQL attribute, 6-2
  - PLSQL_CCFLAGS attribute, 6-4
  - PLSQL_CONN_MEM_LIMIT attribute, 6-5
  - PLSQL_MEMORY_ADDRESS attribute, 6-3
  - PLSQL_MEMORY_SIZE attribute, 6-3
  - PLSQL_OPTIMIZE_LEVEL attribute, 6-5
  - PLSQL_TIMEOUT attribute, 6-6
- constants and variables, 2-2
- CONTINUE statement, 2-7
conversion—see type conversion

cursors
- closed at end of transaction, 2-31
- cursor attributes, 9-4
- cursor FOR loop, example, 5-8
- examples, 5-6
- REF CURSORs, 3-7
- use in PL/SQL, 2-21

data types
- associative arrays (index-by tables), 3-5
- categories, 3-2
- collections, 3-5
- composite data types, 3-4
- conversion between application types and PL/SQL or SQL types, 3-9
- conversion between PL/SQL and SQL, 3-11
- conversion between PL/SQL types, 3-9
- differences in TimesTen, 3-10
- index-by tables (associative arrays), 3-5
- nested tables, 3-5
- non-supported types, 3-13
- overview of what is supported, 3-1
- PL/SQL types, 3-2
- records, 3-5
- REF CURSORs, 3-7
- ROWID, 3-4
- scalar types, 3-2
- SIMPLE_INTEGER, 3-4
- type environments, 3-1
- varrays, 3-5
- DBMS_LOB package, 8-2
- DBMS_LOCK package, 8-4
- DBMS_OUTPUT package, 8-5
- DBMS_PREPROCESSOR package, 8-6
- DBMS_RANDOM package, 8-7
- DBMS_SQL package, 8-8
- DBMS_UTILITY package, 8-10
- DDL statements, 2-12
- definer's rights, 5-18, 7-7
- demo applications, 1-5
- differences in TimesTen
  - data type considerations, 3-10
  - exception handling and behavior, 4-7
  - execution of PL/SQL from SQL, 1-4
  - PL/SQL language element and feature support, 9-1
  - SQL statements in PL/SQL blocks, 1-3
  - transaction behavior, 2-31
- DML returning, 2-15, 5-16
- DML statements, 2-11
- duplicate parameters, 2-10
- dynamic SQL
  - DBMS_SQL package, 8-8
  - EXECUTE IMMEDIATE examples, 5-13
  - EXECUTE IMMEDIATE usage, 2-12

enabling PL/SQL
- checking whether it is enabled, 6-1

errors
- error messages, differences vs. Oracle, 4-9
- exception types, 4-2
- RAISE statement, 4-4
- RAISE_APPLICATION_ERROR procedure, 4-5
- show errors in ttsql, 4-6
- SQLCODE built-in function, 4-4, 4-9
- SQLERRM built-in function, 4-4, 4-9
- transaction and rollback behavior, differences vs. Oracle, 4-7
- trapping predefined exceptions, 4-3
- trapping user-defined exceptions, 4-4
- understanding exceptions, 4-1
- warnings (not supported), 4-9

examples
- bind variables, 5-3
- bulk binding, 5-9
- cursor FOR loop, 5-8
- cursors, 5-6
- dynamic SQL, 5-13
- FETCH statement, 5-6
- INSERT statement, 5-2
- query a system view, 5-22
- RETURNING INTO, 5-16
- SELECT statement, 5-1

exceptions—see errors

EXECUTE IMMEDIATE statement
- examples, 5-13
- usage, 2-12

features, overview, 1-1
- FETCH statement, example, 5-6
- first connection attributes, 6-2
- FOR loop, 2-7
- FORALL statement, 2-14, 5-9
- functions
  - access control, 7-1
  - basic usage and example, 2-21
  - SQL functions, from PL/SQL, 2-5
  - supported ways to execute, 2-8

general connection attributes, 6-2

granted privileges, 7-3

IF-THEN-ELSE statement, 2-6
- IN OUT parameters, 2-10
- IN parameters, 2-10
- index-by tables (associative arrays)
  - binding, 3-6
  - using, 3-5
- INSERT statement, example, 5-2
installing PL/SQL (ttmodinstall), 6-1
integer types
  BINARY_INTEGER, 3-3
  PLS_INTEGER, 3-3
  SIMPLE_INTEGER, 3-4
invoker’s rights, 5-18, 7-7
iterative control, 2-7

L
language elements and features, support, 9-1
LOBs
  DBMS_LOB package, 8-2
  overview, 2-17
  using in PL/SQL, 2-17

N
nested tables, 3-5
NLS_DATE_FORMAT, 3-12
NLS_TIMESTAMP_FORMAT, 3-12
non-ASCII names (not supported), 9-7
non-uppercase names (not supported), 9-7

O
operations of PL/SQL, overview, 1-2
OUT parameters, 2-10
  overview
    components and operations, 1-2
    features, 1-1

P
packages
  access control, 7-1
  concepts, 2-25
  creating and using, 2-25
  TimesTen-supplied packages, 8-1
parameters
  binding, 2-9
  duplicate parameters, 2-10
  examples using bind variables, 5-3
  IN, 2-10
  IN OUT, 2-10
  OUT, 2-10
PLS_INTEGER type, 3-3
PLSCOPE_SETTINGS connection attribute, 6-3
PLSQL connection attribute, 6-2
PLSQL_CCFLAGS connection attribute, 6-4
PLSQL_CONN_MEM_LIMIT connection attribute, 6-5
PLSQL_MEMORYADDRESS connection attribute, 6-3
PLSQL_MEMORY_SIZE connection attribute, 6-3
PLSQL_OPTIMIZE_LEVEL connection attribute, 6-5
PLSQL_TIMEOUT connection attribute, 6-6
predefined exceptions
  not supported by TimesTen, 4-9
  supported by TimesTen, 4-3
privileges
  for procedures, functions, packages, 7-1
  granting and revoking, 7-3
privileges—also see access control
  procedures
    access control, 7-1
    basic usage and example, 2-21
    supported ways to execute, 2-8
programming features
  conditional control, 2-6
    continue, 2-7
    iterative control, 2-7
public objects, 2-25

Q
queries, 2-11
Quick Start demo applications, 1-5

R
RAISE statement (exceptions), 4-4
RAISE_APPLICATION_ERROR procedure, 4-5
records, 3-5
REF CURSORs, 3-7
replication, PL/SQL DDL, not supported, 2-22
RETURNING INTO clause, 2-15, 5-16
revoking privileges, 7-3
ROWID type, 3-4

S
samples—see examples
SAVE EXCEPTIONS clause, 2-14, 5-12
security—see access control
SELECT statement, 2-11
SELECT statement, example, 5-1
show errors, ttIsql, 4-6
SIMPLE_INTEGER type, 3-4
sleep functionality, 8-4
SQL
  DDL statements, 2-12
    dynamic SQL, 2-12
    static SQL, 2-11
SQL functions, from PL/SQL, 2-5
SQLCODE built-in function, 4-4, 4-9
SQLERRM built-in function, 4-4, 4-9
standalone subprograms (procedures and functions), 2-21
static SQL, 2-11
stored functions
  access control, 7-1
    basic usage and example, 2-21
    supported ways to execute, 2-8
stored procedures
  access control, 7-1
    basic usage and example, 2-21
    supported ways to execute, 2-8
subprograms
  access control, 7-1
    basic usage and example, 2-21
    supported ways to execute, 2-8
supplied packages and system views
  access control, 7-8
  system view, querying, 5-22
synonyms
  for packages, 2-28
  for procedures and functions, 2-24

T
TimesTen Application-Tier Database Cache
  (TimesTen Cache), use from PL/SQL, 2-16
TimesTen built-in functions
  access control, 7-9
    calling via EXECUTE IMMEDIATE, 2-12, 5-16
TO_CHAR function, 3-12
TO_DATE function, 3-12
transaction and rollback behavior, differences vs.
  Oracle, 4-7
transaction behavior, 2-31
trapping exceptions
  predefined exceptions, 4-3
  user-defined exceptions, 4-4
TT_DB_VERSION package, 8-12
TT_DECIMAL data type (unsupported), 3-12
TT_STATS package, 8-13
ttIsql, show errors, 4-6
ttmodinstall utility (install PL/SQL), 6-1
ttPLSQLMemoryStats built-in procedure, 6-8, 7-9,
  9-10
ttSrcScan utility (check for unsupported
  features), 6-8
type conversion
  between application types and PL/SQL or SQL
    types, 3-9
  between PL/SQL and SQL, 3-11
  between PL/SQL types, 3-9
  differences in TimesTen, 3-10

U
unsupported features, check with ttSrcScan, 6-8
UTL_FILE package, 8-14
UTL_IDENT package, 8-16
UTL_RAW package, 8-17
UTL_RECOMP package, 8-19

V
variables and constants, 2-2
varrays, 3-5

W
warnings (not supported), 4-9
WHILE loop, 2-7
wrapping PL/SQL source code, 2-29