Oracle® TimesTen In-Memory Database

ttIsql User's Guide and Reference

Release 26.1
G39011-01
March 2026

Oracle TimesTen In-Memory Database ttIsql User's Guide and Reference, Release 26.1

G39011-01

Contents

1 Get Started with ttIsql

About ttIsql 1
Starting ttIsql 1
Customizing the ttIsql Command Prompt 2
Command Shortcuts 2

2 Use ttIsql

About Using ttIsql in Interactive or Batch Mode 1
Help Commands 3
View and Set Attributes 7
Use, Declare, Set Variables and Parameters 10
- Declaring and Setting Bind Variables 10
- Automatically Creating Bind Variables for Retrieved Columns 13
- Parameters 14

Run SQL Statements 18
- Prepare a SQL Statement for Subsequent Runs 18
- Manage Transactions 22

Display Characters in ttIsql 23
Display Database Structures 24
- describe Command 24
- cachegroups Command 25
- repschemes Command 25
- dssize Command 26
- tablesize Command 26
- monitor Command 27

List Database Objects by Object Type 29
View and Change Query Optimizer Plans 30
- Use the showplan Command 30
- View Commands and Explain Plans from the SQL Command Cache 33
  - View Commands in the SQL Command Cache 33
  - Display Query Plan for Statement in SQL Command Cache 34

Create and Run PL/SQL Blocks Within ttIsql 36
- Pass Data From PL/SQL Using OUT Parameters Within ttIsql 37

Manage ODBC Functions 38
Canceling ODBC Functions 39
Timing ODBC Function Calls 39
'editline' Feature for Linux and UNIX Only 39
Emacs Binding 40
vi Binding 40

3 ttIsql Command Reference

accept 1
allfunctions 1
allindexes 2
allpackages 3
allprocedures 3
allsequences 4
allsynonyms 4
alltables 5
allviews 5
builtins 5
bye 7
cachegroups 7
cachesqlget 8
cd 8
clearhistory 9
clienttimeout 9
close 9
cmdcache 10
commit 11
commitdurable 11
compare 11
connect 12
createandloadfromoraquery 12
define 13
describe 13
dssize 14
e: 14
edit 15
exec 15
execandfetch 16
explain 16
fetchall 17
fetchnext 17

fetchone 18
free 18
help 18
history 20
host 21
if-then-else 21
monitor 23
remark 24
retryconnect 24
rpad 25
run 25
savehistory 26
setjoinorder 26
setuseindex 26
setvariable 27
showjoinorder 27
sleep 27
sqlcolumns 28
sqlgetinfo 28
sqlstatistics 28
sqltables 29
statsclear 30
statsestimate 30
statsupdate 30
tablesize 30
use 31
variable 31
version 32
waitfor 32
waitforresult 33
whenever sqlerror 33
xlabookmarkdelete 36

About This Content

This guide provides background information to help you understand how TimesTen works, as well as step-by-step instructions that show how to perform the most commonly-needed tasks..

Audience

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

Conventions

The following text conventions are used in this document.

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

1

Get Started with ttIsql

The TimesTen `ttIsql` utility is a general tool for working with a TimesTen database.

This chapter includes the following topics:

- About ttIsql
- Starting ttIsql
- Customizing the ttIsql Command Prompt
- Command Shortcuts

About ttIsql

You can use the `ttIsql` command line interface to issue SQL statements and built-in `ttIsql` commands to perform various operations.

Some common tasks that are accomplished using `ttIsql` include:

- Set up and maintain databases: You can create tables and indexes, alter existing tables, and update table statistics quickly and easily using `ttIsql`.
- Connection setup: You can connect or disconnect to the TimesTen and Oracle databases.
- Retrieve information on database structures: You can retrieve the definitions for tables, indexes, and cache groups using built-in `ttIsql` commands. In addition, the current size and state of the database can be displayed.
- Run and optimize database operations: You can use the `ttIsql` utility to run, alter, and display query optimizer plans for the purpose of tuning SQL operations. The time required to run various ODBC function calls can also be displayed.

Starting ttIsql

TimesTen includes the `ttIsql` utility in a TimesTen distribution. Before using `ttIsql`, you can set specific environment variables after you create the TimesTen instance.

You can locate the `ttIsql` utility in the `bin` subdirectory of the TimesTen instance. See Overview of Installations and Instances in the *Installation, Migration, and Upgrade Guide*.

These environment variables include `TIMESTEN_HOME`, `PATH`, and `LD_LIBRARY_PATH`. On Linux, you set the environment variables by sourcing the `ttenv.sh` (for a Bourne or Bourne-compatible shell) or `ttenv.csh` (for a C shell).

The shell script for sourcing the environment variables is located in the `bin` subdirectory of the instance directory.

```
$ source ./<instance_dir>/bin/ttenv.sh
```

After setting the environment variables, you can run the following command:

```
$ ttIsql
Copyright (c) 1996, 2024, Oracle and/or its affiliates. All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

Command>
```

Customizing the ttIsql Command Prompt

You can customize the `ttIsql` command prompt by using the `set` command with the `prompt` attribute.

When you are not connected to the database, the command prompt will be:

```
Command> set prompt MY_DSN;
MY_DSN
```

When you are connected to the database and you want to embed spaces, you must quote the string:

```
Command> set prompt "MY_DSN %c> ";
MY_DSN con1>
```

If you don't have a connection, the command prompt will be:

```
Command> set prompt "MY_DSN %c> ";
MY_DSN >
```

Command Shortcuts

By default, `ttIsql` supports keystroke shortcuts when entering commands.

The `ttIsql` keystroke shortcuts are:

Keystroke	Action
Left Arrow	Moves the insertion point left (back).
Right Arrow	Moves the insertion point right (forward).
Up Arrow	Scroll to the command before the one being displayed. Places the cursor at the end of the line. If the command being added to the history is identical to the most recently added command, it is skipped.
Down Arrow	Scrolls to a more recent command history item and puts the cursor at the end of the line. If the command being added to the history is identical to the most recently added command, it is skipped.
Ctrl-A	Moves the insertion point to the beginning of the line.
Ctrl-E	Moves the insertion point to the end of the line.
Ctrl-K	"Kill" - Saves and erases the characters on the command line from the current position to the end of the line.
Ctrl-Y	"Yank"- Restores the characters previously saved and inserts them at the current insertion point.
Ctrl-F	Forward character - move forward one character. (See Right Arrow.)

Keystroke	Action
Ctrl-B	Backward character - moved back one character. (See Left Arrow.)
Ctrl-P	Previous history. (See Up Arrow.)
Ctrl-N	Next history. (See Down Arrow.)

To turn this feature off:

```
Command>set editline 0;
```

To turn this feature on, replace `0` with `1`.

2

Use ttIsql

You can use the `ttIsql` utility in running various SQL statements and built-in `ttIsql` commands. You can run the `ttIsql` commands either in the interactive mode or in the batch mode.

This chapter includes the following topics:

- About Using ttIsql in Interactive or Batch Mode
- Help Commands
- View and Set Attributes
- Use, Declare, Set Variables and Parameters
- Run SQL Statements
- Display Database Structures
- List Database Objects by Object Type
- View and Change Query Optimizer Plans
- Create and Run PL/SQL Blocks Within ttIsql
- Manage ODBC Functions
- 'editline' Feature for Linux and UNIX Only

About Using ttIsql in Interactive or Batch Mode

You can use the `ttIsql` utility in two different ways: interactive mode or batch mode.

When you use `ttIsql` in the interactive mode, you can type commands directly into `ttIsql` from the console. When you use `ttIsql` in the batch mode, you can run a prepared script of `ttIsql` commands by specifying the name of the file.

Use Interactive mode for the following types of tasks:

- Experimenting with TimesTen features, testing design alternatives, and improving query performance.
- Solving database problems by examining database statistics.

Use Batch mode for the following types of tasks:

- Performing periodic maintenance operations including the updating of table statistics, and purging log files.
- Initializing a database by creating tables, indexes, and cache groups and then populating the tables with data.
- Generating simple reports by running common queries.

When you start `ttIsql` from the shell, `ttIsql` is in the default interactive mode. The `ttIsql` utility prompts you to type in a valid `ttIsql` built-in command or SQL statement at the `Command>` prompt. The following example starts `ttIsql` in the interactive mode and then

connects to a TimesTen database by running the `connect` command using `database1` DSN or without using DSN. There are two ways to connect to the database as shown in the example:

```
$ ttIsql
Command> connect database1;
Connection successful:
DSN=database1;DataStore=/disk1/databases/
database1;DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=AL32UTF8;PermSi
ze=128;(Default setting AutoCommit=1)

Command>connect “DSN=database1”;
Connection successful:
DSN=database1;DataStore=/disk1/databases/
database1;DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=AL32UTF8;PermSi
ze=128;(Default setting AutoCommit=1)
```

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

```
$ ttIsql -connstr "DSN=database1"
connect "DSN=database1";
Connection successful:
DSN=database1;DataStore=/disk1/databases/
database1;DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=AL32UTF8;PermSi
ze=128;(Default setting AutoCommit=1)

Command>quit;
```

You can access the batch mode in two different ways. The most common way is to specify the `-f` option on the `ttIsql` command line followed by the name of file to run.

For example, running a file containing a `CREATE TABLE` statement looks like the following:

```
$ ttIsql -f create.sql -connstr "DSN=database1"

connect "DSN=database1";
Connection successful:
DSN=database1;DataStore=/disk1/databases/
database1;DatabaseCharacterSet=AL32UTF8;
ConnectionCharacterSet=AL32UTF8;PermSize=128;
(Default setting AutoCommit=1)

run "create.sql"

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

exit;
Disconnecting...
Done.
```

The other way to use the batch mode is to enter the `run` command directly from the interactive command prompt. The `run` command is followed by the name of the file containing `ttIsql` built-in commands and SQL statements to run:

```
Command> run "create.sql";
CREATE TABLE LOOKUP (KEY NUMBER NOT NULL PRIMARY KEY, VALUE CHAR (64));
Command>
```

In this example, the `create.sql` file is placed in the current directory of the OS shell from where the interactive `ttIsql` utillity is started. Both relative and absolute paths are allowed for the `run` command.

Help Commands

The `ttIsql` utility has an online version of command syntax definitions and descriptions for all built-in `ttIsql` commands.

To access this online help from within `ttIsql`, use the `help` command from the shell's command prompt:

```
$ ttIsql -help
Usage: ttIsql [-h | -help | -helpcmds | -helpfull | -V]
       ttIsql [-f <filename>] [-v <verbosity>] [-e <commands>]
              [-interactive] [-N <ncharEncoding>] [-wait] [-restrict]
              [{<DSN> | -connstr <connStr>}]

 -h                 Prints this message and exits.

 -help              Prints this message and exits.

 -helpcmds          Prints a brief description of ttIsql commands and exits.

 -helpfull          Prints a full description of ttIsql commands and exits.

 -V                 Prints version information and exits.

 <DSN>, <connStr>   The ODBC connection string or DSN to use as an
                    argument to the connect command. The connect command
                    will then be the first command executed when
                    starting ttIsql.

 -f <filename>      Specifies an input file of ttIsql commands to be
                    executed on start up.

 -v <verbosity>     Specifies the output verbosity level. The output
                    verbosity level argument takes a value of 0, 1, 2,
                    3 or 4.

                    If verbosity = 0 then only error messages are displayed.

                    If verbosity = 1 then the basic output generated by
                    commands is displayed.

                    Verbosity = 2 is the default verbosity level. At this
level
```

```
                    simplified SQL error and information messages are
                    displayed. Command output includes additional information,
                    and commands that are read from an external file are
                    echoed to the display.

                    If verbosity = 3 then more detailed SQL error and
                    information messages are displayed.

                    If verbosity = 4 then the most detailed SQL error and
                    information messages are displayed. At this level
                    additional informational messages are displayed depending
                    upon the command executed.

 -e <commands>      Specifies a semicolon-separated list of ttIsql commands
                    to execute on start up.

 -N <ncharEncoding> Specifies the character encoding method for native
                    character input/output. Valid values are ASCII,
                    and LOCALE. The default encoding method (LOCALE)
                    is derived from the current environment locale settings.

 -wait              Forces the program to wait during a connection attempt
                    until the connection attempt is successful.

 -interactive       Forces interactive mode. This is useful when running from
                    an emacs comint buffer.

-restrict          Limits functionality that interacts with the OS.
                    Prevent running the EDIT, HOST, PUTENV, SPOOL, cachesqlget
                    SET SIGNALBAHAVIOR commands.
                    Limits actions that write to arbitrary files or
                    execute arbitrary processes.

      Default command line options can be set by exporting an environment
      variable called TTISQL. The value of the TTISQL environment variable
      is a string with the same syntax requirements as the ttIsql command
      line. If the same option is present in the TTISQL environment
      variable and the command line, then the command line version always
      takes precedence.

      Examples:

          ttIsql -connStr "DSN=mydsn" -v 4
          ttIsql -helpfull
          ttIsql -connStr "DSN=mydsn" -e "autocommit 0; showplan 1;"
          ttIsql -connStr "DSN=mydsn" -f script.sql -N ASCII
```

To view the list of available commands displayed by the `help` command:

```
Command> help

Use "help all" to get a description of all commands or use "help <cmd>" to

limit it to that command.
```

```
?                           fetchnext                sqlcolumns
!                           free                     sqlgetinfo
@@                          functions                sqlquerytimeout
accept                      getenv                   sqlquerytimeoutmsec
allfunctions                help                     sqlstatistics
allindexes                  history                  sqltables
allpackages                 host                     statsclear
allprocedures               if                       statsestimate
allsequences                indexes                  statsupdate
allsynonyms                 isolation                synonyms
alltables                   monitor                  tables
allviews                    multipleconnections      tblsize
autocommit                  ncharencoding            timing
builtins                    noecho                   tryhash
cachegroups                 optfirstrow              trymaterialize
cachesqlget                 optprofile               trymergejoin
cd                          packages                 trynestedloopjoin
clearhistory                passthrough              tryrowid
clienttimeout               prefetchcount            tryrowlocks
close                       prepare                  tryserial
closeall                    print                    trytbllocks
cmdcache                    procedures               trytmphash
columnlabels                putenv                   trytmptable
commit                      prompt                   trytmprange
commitdurable               remark                   tryrange
compare                     repschemes               undefine
connect                     retryconnect             unsetjoinorder
createandloadfromoraquery   rollback                 unsetuseindex
define                      rpad                     use
define2var                  run                      variable
describe                    savehistory              var2define
disconnect                  sequences                verbosity
dssize                      set                      version
e:                          setjoinorder             vertical
edit                        setuseindex              views
exec                        setvariable              waitfor
execandfetch                show                     waitforresult
exit                        showjoinorder            whenever
explain                     showplan                 xlabookmarkdelete
fetchall                    sleep                    <sql_statement>
fetchone                    spool
```

To view a detailed description of any built-in `ttIsql` commands, type the `help` command followed by one or more `ttIsql` commands to display help for. The example below displays the online description for the `connect` and `disconnect` commands:

```
Command>help connect disconnect

Arguments in <> are required.
Arguments in [] are optional.

Command Usage: connect [DSN|connection_string] [as <connection_id>]
Command Aliases: (none)
Description: Connects to the data source specified by the optional DSN or
connection string argument. If an argument is not given, then the DSN or
```

```
connection string from the last successful connection is used. A connection
ID
may optionally be specified, for use in referring to the connection when
multiple
connections are enabled. The DSN is used as the default connection ID. If
that ID
is already in use, the connection will be assigned the ID "conN", where N is
some
number larger than 0.
Requires an active connection: NO
Requires autocommit turned off: NO
Reports elapsed execution time: YES
Works only with a TimesTen data source: NO
Example: connect; -or- connect RunData; -or- connect "DSN=RunData";
-or- connect RunData as rundata1;

Command Usage: disconnect [all]
Command Aliases: (none)
Description: Disconnects from the currently connected data source or all
connections when the "all" argument is included. If a transaction is active
when
disconnecting then the transaction will be rolled back automatically. If a
connection exists when executing the "bye", "quit" or "exit" commands then
the
"disconnect" command will be executed automatically.
Requires an active connection: NO
Requires autocommit turned off: NO
Reports elapsed execution time: YES
Works only with a TimesTen data source: NO
Example: disconnect;
```

In case of attributes, you can view and set attributes with the `show` and `set` commands and in case of commands, you can run built-in commands from the `ttIsql` utility. See View and Set Attributes.

To view the list of attributes:

```
Command>help attributes
Known attributes:
  - autoprint
  - autovariables
  - autocommit
  - clienttimeout
  - columnlabels
  - connstr
  - define
  - distribution
  - dynamicloadenable
  - dynamicloaderrormode
  - echo
  - editline
  - failovermessage
  - errors
  - feedback
  - isolation
  - long
```

```
  - longchunksize
  - multipleconnections
  - ncharencoding
  - optfirstrow
  - optprofile
  - passthrough
  - prefetchcount
  - prompt
  - querythreshold
  - rowdelimiters
  - sqlquerytimeout
  - sqlquerytimeoutmsec
  - serveroutput
  - session_action
  - session_client_info
  - session_module
  - showcurrenttime
  - showplan
  - termout
  - timing
  - tryhash
  - trymaterialize
  - trymergejoin
  - trynestedloopjoin
  - tryrange
  - tryrowid
  - tryrowlocks
  - tryserial
  - trytbllocks
  - trytmphash
  - trytmprange
  - trytmptable
  - verbosity
  - vertical

For detailed attribute help, do "help attribute <attributename>"
```

To view help for a specific attribute:

```
Command>help attribute showcurrenttime;

Attribute usage: showcurrenttime

Attribute Usage: showcurrenttime [0|off|1|on]
Description: The current time is printed just after the command is read and
before the command is executed.
Used with Set: YES
Used with Show: YES
```

View and Set Attributes

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

Use `show all` command to display the setting for all `set`/`show` attributes.

```
Command>show all;
Connection independent attribute values:
  - autoprint = 0 (OFF)
  - autovariables = 0 (OFF
  - columnlabels = 0 (OFF)
  - define = 0 (OFF)
  - echo 1 (ON)
  - editline = 1 (ON)
  - FEEDBACK ON
  - LONG 80
  - LONGCHUNKSIZE 65536
  - multipleconnections = 0 (OFF)
  - ncharencoding = LOCALE (AL32UTF8)
  - prompt = 'Command> '
  - rowdelimiters = 1 (ON) before:'< ', after:' >', separator:', '
  - showcurrenttime = 0 (OFF)
  - termout = 1 (ON)
  - timing = 0 (OFF)
  - verbosity = 2
  - vertical = 0 (OFF)

Connection specific attribute values:

  - autocommit = 1 (ON)
  - Client timeout = 0
  - Connection String DSN=<database name>;UID=<uid>; DataStore=<database
path>;
 DatabaseCharacterSet=AL32UTF8; ConnectionCharacterSet=US7ASCII;
 DRIVER=/sw/tthome/install/lib/libtten.so; PermSize=20;TempSize=20;
  - No failover messages.
  - PL/SQL Errors: No errors.
  - isolation = READ_COMMITTED
  - Prefetch count = 5
  - Query threshold = 0 seconds (no threshold)
  - Query timeout = 0 seconds (no timeout)
  - Query timeout = 0 seconds (no timeout)
  - serveroutput OFF
  - SESSION_ACTION =
  - SESSION_CLIENT_INFO =
  - SESSION_MODULE = _ttIsql@phoenix92527

Current Optimizer Settings:
  - Scan: 1
  - Hash: 1
  - Range: 1
  - TmpHash: 1
  - TmpRange: 1
  - TmpTable: 1
  - NestedLoop: 1
  - MergeJoin: 1
  - GenPlan: 0
  - TblLock: 1
  - RowLock: 1
  - HashGb: 1
```

```
  - Rowid: 1
  - FirstRow: 0
  - IndexedOr: 1
  - PassThrough: 0
  - BranchAndBound: 1
  - ForceCompile: 0
  - ShowJoinOrder: 0
  - TransparentLoad: 0
  - UseBoyerMooreStringSearch: 0
  - DynamicLoadEnable: 1
  - DynamicLoadErrorMode: 0
  - NoRemRowIdOpt: 0
  - FastPrepare: 1
  - VectorProcess: 1
  - Rewrite: 1
  - DynamicLoadMultiplePKs: 1
  - DynamicLoadRootTbl: 0
  - TmpIdxForMatRes: 0

Current Join Order:
        <>
Command
```

To view the setting for the `Passthrough` attribute, enter:

```
Command>show passthrough;
PassThrough = 0
```

To change the `Passthrough` setting, enter:

```
Command>set passthrough 1;
```

You have the `help attributes` command to list the help for all the attributes that could be set or shown. See Help Commands.

To view the detailed attribute help:

```
Command> help attribute clienttimeout

Attribute usage: clienttimeout

Command Usage: clienttimeout [<timeout seconds>]
Command Aliases: (none)
Description: Sets the client timeout (in seconds) for the current connection.
If the optional argument is omitted then the current client timeout is
reported. The client timeout can take a value between 0 and 99999 inclusive.
Requires an active connection: YES
Requires autocommit turned off: NO
Reports elapsed execution time: YES
Works only with a TimesTen data source: YES
Example: show clienttimeout; -or- set clienttimeout 10;
Used with Set: YES
Used with Show: YES
```

Use, Declare, Set Variables and Parameters

You can declare, set, use bind variables and parameters in ttIsql.

- Declaring and Setting Bind Variables
- Automatically Creating Bind Variables for Retrieved Columns
- Parameters

Declaring and Setting Bind Variables

You can declare and set variables and arrays in ttIsql that can be referenced in a SQL statement, SQL script, or PL/SQL block.

The variables declared using the `variable` and `setvariable` command must be one of the following data types: `NUMBER`, `CHAR`, `NCHAR`, `VARCHAR2`, `NVARCHAR2`, `CLOB`, `NCLOB`, `BLOB`, or `REFCURSOR`. However, when binding arrays, TimesTen supports only binding arrays of the `NUMBER`, `CHAR`, `NCHAR`, `VARCHAR2`, or `NVARCHAR2` data types.

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

To see the help usage on the `variable` or the `var` command:

```
Command> help var;
Arguments in <> are required.
Arguments in [] are optional.

Command Usage: variable [<VariableName> [<DataType>] [:= <Value>]]
Command Aliases: var
Description: Declares a bind variable that can be referenced in a statement,
or
displays the definition of the variable if the type is missing. The type can
be one of the following: (n), NUMBER, CHAR(n), NCHAR(n), VARCHAR2(n) ,
NVARCHAR2(n), BLOB, CLOB, NCLOB, or REFCURSOR. If only '(n)' is supplied, it
is assumed to be VARCHAR2 (n).

The syntax for binding multiple values to an array using the variable command
is as follows: variable ArrayName '[' ArraySize ']' DataType(n) := '['
Value1,... ValueX ']'

Requires an active connection: NO

Requires autocommit turned off: NO

Reports elapsed execution time: NO

Works only with a TimesTen data source: NO
```

```
Example: variable; -or- variable a varchar2(30); -or- var arr[5] number :=
[ 1, 2, 3 ];
```

To see the help usage on the `define2var`. It defines the value and make it a variable that you can use as a parameter variable.

```
Command> help define2var;
Arguments in <> are required.
Arguments in [] are optional.
Command Usage: define2var <alias> <varname>
Command Aliases: (none)
Description: Copies a substitution alias to a variable value. Both the alias
and the variable must exist. The variable must be character based. If it is
an array, you must specify which element to define and you must use a value
that fits within the size of the array. If it is not an array, the variable
may be enlarged to fit the new length if necessary.
Requires an active connection: NO
Requires autocommit turned off: NO
Reports elapsed execution time: NO
Works only with a TimesTen data source: NO
Example: define2var myalias mybindvar; define2var a1 arr[2];
```

To see the help usage on the `var2define`. It takes the value of a named variable and then define it.

```
Command> help var2define;
Arguments ! in <> are required.
Arguments in [] are optional.
Command Usage: var2define <varname> <alias>
Command Aliases: (none)
Description: Copies a variable value to a substitution alias. The variable
must be character based. If it is an array, you must specify which element to
define. The define does not need to exist prior to running this command. A
NULL value in the variable defines an empty string.
Requires an active connection: NO
Requires autocommit turned off: NO
Reports elapsed execution time: NO
Works only with a TimesTen data source: NO
Example: var2define mybindvar myalias; var2define arr[2] alias2;
```

To declare, set the value of variables, and print their values:

```
Command> var a varchar2(100); #declare variable a

Command> var b varchar2(100) := 'This is B'; #declare variable b and assign
value to it

Command> var;  #display available variables

variable   A

datatype   VARCHAR2(100)

variable   B
```

```
datatype   VARCHAR2(100)

Command> print a; #print the value of variable a


A            : <NULL>

Command> print; #print values of all available variables

A            : <NULL>

B            : This is B
```

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

```
Command> VARIABLE house_number NUMBER := 268; #declare variable house_number and assign
the value
Command> PRINT house_number;  #print the value of house_number variable
HOUSE_NUMBER            : 268

Command> VARIABLE street_name VARCHAR2(15); #declare variable street_name
Command> SETVARIABLE street_name := 'Oracle Parkway'; #set the value of the variable
street_name
Command> PRINT street_name;
STREET_NAME      :   Oracle Parkway

Command> VARIABLE occupants[5] VARCHAR2(15);  #declare an array variable
Command> SETVARIABLE occupants[1] := 'Pat'; #set the value at the first position of the
array
Command> SETVARIABLE occupants[2] := 'Terry'; #set the value at the second position of
the array
Command> PRINT occupants;  #print the array
OCCUPANTS            : ARRAY [ 5 ] (Current Size 2)
OCCUPANTS[1] : Pat
OCCUPANTS[2] : Terry
```

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

```
Command> VARIABLE occupants[5] VARCHAR2(15) := ['Pat', 'Terry']; #declare an array
variable with values
Command> PRINT occupants; #print an array
OCCUPANTS : ARRAY [ 5 ] (Current Size 2)
OCCUPANTS[1] : Pat
OCCUPANTS[2] : Terry
```

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

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

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

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

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

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

PL/SQL procedure successfully completed.

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

Automatically Creating Bind Variables for Retrieved Columns

When you set `autovariables` on in `ttIsql`, TimesTen creates an automatic bind variable named after each column fetched. As the rows are fetched, the fetched values are copied into the variables.

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

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

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

```
MANAGER_ID           : 205
DEPARTMENT_ID        : 110
```

As each result set is fetched, each column value is placed into a variable named after the column. If more than one row is fetched, the last value of each column overrides the value of the variable. To prevent the variables from being overwritten, turn off the setting.

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

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

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

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

```
Command> DESCRIBE TTCONFIGURATION;

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

1 procedure found.
```

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

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

Command> call TTCONFIGURATION('LockLevel');

PARAMNAME, PARAMVALUE
< LockLevel, 0 >
1 row found.
Command> print paramname;
PARAMNAME : LockLevel
```

Parameters

Parameters let you bind values that will be used in processing SQL statements. These parameters are marked by using ? or by using `:IDENTIFIER` within the SQL statement.

ttIsql will prepare your SQL statement text. It will then look for variables that match the parameter names. For parameters where there are no matching variables, ttIsql will prompt for the values. The name of the parameter when ? is used is QMARK_N where N starts with 1 and is incremented as each ? is encountered in the statement. For example, QMARK_1, QMARK_2, etc. The name of :PARAM is PARAM.

With dynamic parameters, you are prompted for input for each parameter on a separate line. Values for parameters are specified the same way literals are specified in SQL.

Parameter values must be terminated with a semicolon character.

The possible types of values that can be entered are:

- Numeric literals. For example, 1234.5
- Time, date or timestamp literals within single quotation marks. Examples:

  ```
  '12:30:00'
  '2000-10-29'
  '2000-10-29 12:30:00'
  ```

- Unicode string literals within single quotation marks preceded by 'N'. For example, N'abc'
- A NULL value. For example, NULL
- The '*' character that indicates that the parameter input process should be stopped. For example, *
- The '?' character prints the parameter input help information. For example, ?

This example shows that A and B are dynamic parameters and not variables.

```
Command> select * from dual where :a > 100 and :b < 100;
Type '?' for help on entering parameter values.
Type '*' to end prompting and abort the command.
Type '-' to leave the parameter unbound.
Type '/;' to leave the remaining parameters unbound and execute the command.

Enter Parameter 1 'A' (NUMBER) > 110
Enter Parameter 2 'B' (NUMBER) > 99
< X >
1 row found.

Command> print;
```

The print command does not show result as they are dynamic parameters.

This examples shows that :a and :b are used as variables and also, they are used as parameters but are not dynamic.

```
Command> var a number;
Command> exec :a := 110;

PL/SQL procedure successfully completed.

Command> print a;
A                     : 110
Command> var b number;
Command> exec :b := 99;
```

```
PL/SQL procedure successfully completed.

Command> select * from dual where :a > 100 and :b < 100;
< X >
1 row found.
Command> print;
A                    : 110
B                    : 99
```

This example shows that both `:a` and `:b` are variables and there is a dynamic parameter `:c` that can be combined together.

```
Command> var a number;
Command> exec :a := 110;
Command> var b number;
Command> exec :b := 99;
Command> select * from dual where :a > 100 and :b < 100 and :c > 0;
Enter Parameter 3 'C' (NUMBER) > 1
< X >
1 row found.
Command> print;
A                    : 110
B                    : 99
```

In this example, since `_QMARK_1` is not declared, `ttIsql` will prompt for the parameter value. The `:a` parameter is filled in with the `A` variable whose value is `110`. The `:b` is filled in with the `B` variable whose value is `99`. These values are then sent to the engine, which processes the statement and returns the value `abc11099`.

```
Command> select cast(? as varchar2(10)) || :a || :b from dual;

Type '?' for help on entering parameter values.

Type '*' to end prompting and abort the command.

Type '-' to leave the parameter unbound.

Type '/;' to leave the remaining parameters unbound and execute the command.

Enter Parameter 1 '_QMARK_1' (VARCHAR2) > 'abc';

< abc11099 >

1 row found.
```

If you provide the name of a file when prompted for a parameter, the contents of that file will be used as the value for the parameter.

To view the file content:

```
$cat -n filename
1  'aaaa'bbb
2  12"34
3  xyz
```

```
4
5  '
6
```

You can enter a file name as a parameter after @ or @@ that contains the content. The following output is displayed:

```
Command> select cast(? as varchar2(100)) from dual;
...
Enter Parameter 1 '_QMARK_1' (VARCHAR2) > @filename
< 'aaaa'bbb

12"34

xyz

'
>
1 row found.
```

You can also use @@ to specify files in a relative path to the script being executed with the `run` command.

For diagnostic information, set the verbosity level to 4, `ttIsql` will print a diagnostic message when the file is loaded:

```
Command> select cast(? as varchar2(50)) from dual;
...
Enter Parameter 1 '_QMARK_1' (VARCHAR2) > @test4
File "test4" loaded 31 bytes
< aaaaaa
bbbbbbbb
cccc
ddddddddd
>
1 row found.
```

If you do not enter the file name after @ or @@, the following error message is displayed:

```
Enter Parameter 1 '_QMARK_1' (VARCHAR2) > @@
No file name was provided. Type '?;' for help.
```

If the file is not found, the following output is displayed:

```
Enter Parameter 1 '_QMARK_1' (VARCHAR2) > @filename
File "filename.sql" does not exist.
```

If the file cannot be read, the following error message appears:

```
Enter Parameter 1 '_QMARK_1' (VARCHAR2) > @noread
Unable to open file: "noread.sql"
```

This message appears if the input is a directory.

```
Enter Parameter 1 '_QMARK_1' (VARCHAR2) > @//

"//" is a directory
```

> **Note**
>
> A file as a parameter is not supported for `NCHAR` or `NVARCHAR` data types.

Run SQL Statements

This topic provides information about the SQL statements that are run by the `ttIsql` utility.

SQL statements are generally considered to be either data definition language (DDL) statements or data manipulation language (DML) statements.

DDL statements create or modify the database schema. `CREATE TABLE` and `DROP TABLE` are examples of the DDL statements.

DML statements modify database objects. `INSERT`, `UPDATE`, and `DELETE` are examples of the DML statements. The `SELECT` statement retrieves data from one or more tables or views.

For SQL commands, see SQL Statements in the *Oracle TimesTen In-Memory Database SQL Reference*.

Prepare a SQL Statement for Subsequent Runs

TimesTen application is recommended to prepare a single SQL statement and run it as many times as needed without re-preparing the same SQL statement each time it ran.

The `ttIsql` utility has a set of built-in commands to work with prepared SQL statements.

These commands are summarized below:

- `prepare` - Prepares a SQL statement. Corresponds to a `SQLPrepare` ODBC call.
- `exec` - Runs a prepared statement. Corresponds to a `SQLExecute` ODBC call.
- `execandfetch` - Runs a previously prepared statement and fetches all result rows. Corresponds to a `SQLExecute` call followed by one or more calls to `SQLFetch`.
- `fetchone` - Fetches only one row for a previously run statement. Corresponds to exactly one `SQLFetch` call.
- `fetchall` - Fetches all result rows for a previously run statement. Corresponds to one or more `SQLFetch` calls.
- `close` - Closes the result set cursor on a previously run statement that generated a result set. Corresponds to a `SQLFreeStmt` call with the `SQL_CLOSE` option.
- `free` - Closes and deletes a previously prepared statement. Corresponds to a `SQLFreeStmt` call with the `SQL_DROP` option.
- `describe *` - Describes all the prepared statements including the input parameters and the result columns.

The ttIsql utility prepared statement commands also handle SQL statement parameter markers. When parameter markers are included in a prepared SQL statement, ttIsql automatically prompts by name for the value of each parameter in the statement at runtime.

This example uses the prepared statement commands of the ttIsql utility to prepare an INSERT statement into a table containing a NUMBER and a CHAR column. The statement is prepared and then runs twice with different values for each of the statement's two parameters. The ttIsql utility timing command is used to display the elapsed time required to run the primary ODBC function call associated with each command.

```
Command> connect "DSN=database1";
Connection successful:
DSN=database1;DataStore=/disk1/databases/
database1;DatabaseCharacterSet=AL32UTF8;
ConnectionCharacterSet=AL32UTF8;PermSize=128;
(Default setting AutoCommit=1)

Command> timing 1;
Command> create table t1 (key number not null primary key, value char(20));
Execution time (SQLExecute) = 0.007247 seconds.

Command> prepare insert into t1 values (:f, :g);
Execution time (SQLPrepare) = 0.000603 seconds.

Command> exec;
Type '?' for help on entering parameter values.
Type '*' to end prompting and abort the command.
Type '-' to leave the parameter unbound.
Type '/' to leave the remaining parameters unbound and execute the command.
Enter Parameter 1 'F' (NUMBER) > 1;
Enter Parameter 2 'G' (CHAR) > 'abc';
1 row inserted.
Execution time (SQLExecute) = 0.000454 seconds.

Command> exec;
Type '?' for help on entering parameter values.
Type '*' to end prompting and abort the command.
Type '-' to leave the parameter unbound.
Type '/' to leave the remaining parameters unbound and execute the command.
Enter Parameter 1 'F' (NUMBER) > 2;
Enter Parameter 2 'G' (CHAR) > 'def';
1 row inserted.
Execution time (SQLExecute) = 0.000300 seconds.

Command> free;
Command> select * from t1;
< 1, abc >
< 2, def >
2 rows found.
Execution time (SQLExecute + Fetch Loop) = 0.000226 seconds.
```

The prepare command is immediately followed by the SQL statement to prepare. Whenever a SQL statement is prepared in ttIsql, a unique command ID is assigned to the prepared statement. The ttIsql utility uses this ID to keep track of multiple prepared statements. A maximum of 256 prepared statements per connection can exist in a ttIsql session

simultaneously. When the `free` command runs, the command ID is automatically disassociated from the prepared SQL statement.

To see the command IDs generated by `ttIsql` when using the prepared statement commands, use the `describe *` command to list all prepared statements with their IDs. Command IDs can be referenced explicitly when using `ttIsql`'s prepared statement commands.

This example prepares and runs a `SELECT` statement with a predicate containing one `NUMBER` parameter. The `exec 2` and `exec 3` run the prepared statements. After the execution of the 2 and 3 prepared statements, it shows "`Cursor is open. Ncharencoding is LOCALE.`"

```
Command> prepare 1 select 1 from dual;
Command> prepare 2 select 2 from dual;
Command> prepare 3 select 3 from dual;
Command>exec 2;
Command>exec 3;
Command>describe *;

There are 3 prepared commands.

Prepared Statement [1];
SQL: select 1 from dual

Columns:
EXP        NUMBER(10) Not NULL

Prepared Statement [2];
SQL: select 2 from dual

Cursor is open. Ncharencoding is LOCALE.
Columns:
EXP        NUMBER(10) Not NULL

Prepared Statement [3];
SQL: select 3 from dual

Cursor is open. Ncharencoding is LOCALE.
Columns:
EXP
result columns.
```

This example shows `free 2` command to remove the prepared command, `prepare 2`.

```
Command>free 2;
Command>describe *;

Prepared Statement [1];
SQL: select 1 from dual

Columns:
EXP    NUMBER(10) Not NULL

Prepared Statement [3];
SQL: select 3 from dual

Cursor is open. Ncharencoding is LOCALE.
```

```
Columns:
EXP    NUMBER(10) Not NULL

Command>exec 2;
The prepared command with id=2 was not found.
The command failed.
```

This example uses the prepared command, `prepare 10` and `prepare 11` to `INSERT` values into the table `t2` and run the command to insert two rows into the table. The `fetchone` command is used to fetch only one result row generated by the statement. The `execandfetch 12` command displays all rows of the table `t2`.

```
Command> create table t2 ( a int, b varchar(30));
Command> create sequence s;
Command> set autocommit 0;
Command> prepare 10 insert into t2 values( s.nextval, 'A');
Command> prepare 11 insert into t2 values( s.nextval, 'B');
Command> prepare 12 select * from t2;
Command> describe *;

There are 3 prepared commands.

Prepared Statement [10]:
  SQL: insert into t2 values( s.nextval, 'A')
  Columns:
    (none)

Prepared Statement [11]:
  SQL: insert into t2 values( s.nextval, 'B')
  Columns:
    (none)

Prepared Statement [12]:
  SQL: select * from t2
  Columns:
    A        NUMBER (38)
    B        VARCHAR2 (30)

Command> exec 10;
1 row inserted.
Command> exec 11;
1 row inserted.
Command> exec 12;

Command> fetchone;
< 1, A >
1 row found.

Command> fetchone;
< 2, B >
1 row found.

Command> execandfetch 12;
< 1, A >
< 2, B >
```

```
2 rows found.

Command> exec 10;
1 row inserted.
Command> execandfetch 12;
< 1, A >
< 2, B >
< 3, A >
3 rows found.
Command>
```

Manage Transactions

The `ttIsql` utility has several built-in commands for managing transactions.

These commands are summarized below:

- `autocommit` - Turns on or off the autocommit feature. This can also be set as an attribute of the `set` command.
- `commit` - Commits the current transaction.
- `commitdurable` - Ensures the committed work is durably commit to the transaction log on disk in case of database failure.
- `isolation` - Changes the transaction isolation level. This can also be set as an attribute of the `set` command.
- `rollback` - Rolls back the current transaction.
- `sqlquerytimeout` - Specifies the number of seconds to wait for a SQL statement to run before returning to the application. This can also be set as an attribute of the `set` command.

When starting `ttIsql`, the autocommit feature is turned on by default, even within a SQL script. In this mode, every SQL operation against the database is committed automatically.

To turn the autocommit feature off, run the `autocommit` command with an argument of `0`. When autocommit is turned off, transactions must be committed or rolled back manually by running the `commit`, `commitdurable` or `rollback` commands. The `commitdurable` command ensures that the transaction's effect is preserved to disk in case of database failure. If autocommit is off, the uncommitted statements that are rolled back are not reported when `ttIsql` exits.

The `isolation` command can be used to change the current connection's transaction isolation level. The isolation level can be changed only at the beginning of a transaction. The `isolation` command accepts one of the following constants: `READ_COMMITTED` and `SERIALIZABLE`. If the `isolation` command is modified without an argument then the current isolation level is reported.

The `sqlquerytimeout` command sets the timeout period for SQL statements. If the run time of a SQL statement exceeds the number of seconds set by the `sqlquerytimeout` command, the SQL statement does not run and an 6111 error is generated. See Setting a Timeout Duration for SQL Statements in the *Oracle TimesTen In-Memory Database Java Developer's Guide* and Setting a Timeout Duration for SQL Statements in the *Oracle TimesTen In-Memory Database C Developer's Guide*.

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

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

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

Command> connect "DSN=database1";
Connection successful:
DSN=database1;DataStore=/disk1/databases/database1;DatabaseCharacterSet=AL32UTF8;
ConnectionCharacterSet=AL32UTF8;PermSize=128;
(Default setting AutoCommit=1)
Command> autocommit 0;
Command> CREATE TABLE LOOKUP (KEY NUMBER NOT NULL PRIMARY KEY, VALUE CHAR (64));
Command> commit;
Command> INSERT INTO LOOKUP VALUES (1, 'ABC');
1 row inserted.
Command> SELECT * FROM LOOKUP;
< 1, ABC >
1 row found.
Command> rollback;
Command> SELECT * FROM LOOKUP;
0 rows found.
Command> isolation;
isolation = READ_COMMITTED
Command> commitdurable;
Command> Set sqlquerytimeout 5;
Command> select count (*) from tables, views, columns, all_source;
 6111: SQL statement has reached its timeout limit [timeout msec = 5000, elapsed msec =
5120] and has been terminated
The command failed.
Command> sqlquerytimeout;
Query timeout = 10 seconds
Command> disconnect;
Disconnecting...
Command> exit;
Done.
```

Display Characters in ttIsql

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

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

To override the locale-based output format, use the `ncharencoding` option or the `-N` option. The valid values for these options are `LOCALE` (the default) and `ASCII`. If you choose `ASCII` and `ttIsql` encounters a Unicode character, it displays it in escaped format.

You do not need to have an active connection to change the output method.

Display Database Structures

There are several `ttIsql` commands that display information on database structures.

The most useful commands are summarized below:

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

describe Command

Use the `describe` command to display information on individual database objects. Displays parameters for prepared SQL statements and built-in procedures.

The argument to the `describe` command can be the name of a table, cache group, view, materialized view, sequence, synonym, a built-in procedure, a SQL statement or a command ID for a previously prepared SQL statement, a PL/SQL function, PL/SQL procedure or PL/SQL package.

The `describe` command requires a semicolon character to terminate the command.

```
Command> CREATE TABLE t1 (KEY NUMBER NOT NULL PRIMARY KEY, VALUE CHAR (64));
Command> describe t1;

Table USER.T1:
  Columns:
   *KEY                           NUMBER NOT NULL
    VALUE                         CHAR (64)
1 table found.

(primary key columns are indicated with *)
Command> describe SELECT * FROM T1 WHERE KEY=?;

Prepared Statement:
  Parameters:
    Parameter 1                   NUMBER
  Columns:
    KEY NUMBER                    NOT NULL
    VALUE                         CHAR (64)
Command> describe ttOptUseIndex;

Procedure TTOPTUSEINDEX:
  Parameters:
    Parameter INDOPTION           VARCHAR (1024)
  Columns:
```

```
    (none)

1 procedure found.
Command>
```

cachegroups Command

The `cachegroups` command is used to provide detailed information on cache groups defined in the current database.

The attributes of the root and child tables defined in the cache group are displayed in addition to the `WHERE` clauses associated with the cache group. The argument to the `cachegroups` command is the name of the cache group that you want to display information for and when no argument is provided, it returns information of all the cache groups defined by the connected user in the current database.

```
Command> cachegroups;
Cache Group CACHEUSER.READCACHE:
 Cache Group Type: Read Only
 Autorefresh: Yes
 Autorefresh Mode: Incremental
 Autorefresh State: Paused
 Autorefresh Interval: 5 Seconds
 Autorefresh Status: ok
 Aging: No aging defined
 Root Table: SALES.READTAB
 Table Type: Read Only

 Cache Group CACHEUSER.WRITECACHE:
 Cache Group Type: Asynchronous Writethrough (Dynamic)
 Autorefresh: No
 Aging: LRU on
 Root Table: SALES.WRITETAB
 Table Type: Propagate
2 cache groups found.
```

repschemes Command

The `repschemes` command is used to report information on replication schemes defined in the currently connected data source.

This information includes the attributes of all elements associated with the replication schemes. If the optional argument is omitted then information on all replication schemes defined in the current data source is reported.

```
Command>repschemes [[<scheme_owner_pattern>.]<scheme_name_pattern>];
```

Example:

```
Command>create active standby pair rep1, rep2;
The command succeeded.

Command>repschemes;
```

```
Replication Scheme Active Standby:

  Master Store: REP1 on xxx
  Master Store: REP2 on xxx

  Excluded Tables:
    None

  Excluded Cache Groups:
    None

  Excluded sequences:
    None

  Store: REP1 on xxx
    Port: (auto)
    Log Fail Threshold: (none)
    Retry Timeout: 120 seconds
    Compress Traffic: Disabled

  Store: REP2 on xxx
    Port: (auto)
    Log Fail Threshold: (none)
    Retry Timeout: 120 seconds
    Compress Traffic: Disabled

1 replication scheme found.
```

dssize Command

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

The `dssize` command reports the same information that is displayed in the `SYS.V$MONITOR` and `SYS.GV$MONITOR` system views. The following example uses the `k` option to print the database size information in KB:

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

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

tablesize Command

The `tablesize` command displays the detailed analysis of the amount of space used by a table.

Once you call the `ttComputeTabSizes` built-in procedure, which analyzes the table size of the indicated tables, the `tablesize` command displays the total size data for all analyzed tables.

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

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

The syntax for `tablesize` is as follows:

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

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

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

Sizes of USER1.EMPLOYEES:

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

1 table found.
```

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

For example:

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

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

monitor Command

The monitor command displays all of the information provided by the `dssize` command and additional statistics on the number of connections, checkpoints, lock timeouts, commits,

rollback operations and other information collected since the last time the database was loaded into memory.

```
Command>monitor;
TIME_OF_1ST_CONNECT: Wed Apr 20 10:34:17 2011
DS_CONNECTS: 11
DS_DISCONNECTS: 0
DS_CHECKPOINTS: 0
DS_CHECKPOINTS_FUZZY: 0
DS_COMPACTS: 0
PERM_ALLOCATED_SIZE: 40960
PERM_IN_USE_SIZE: 5174
PERM_IN_USE_HIGH_WATER: 5174
TEMP_ALLOCATED_SIZE: 18432
TEMP_IN_USE_SIZE: 4527
TEMP_IN_USE_HIGH_WATER: 4527
SYS18: 0
TPL_FETCHES: 0
TPL_EXECS: 0
CACHE_HITS: 0
PASSTHROUGH_COUNT: 0
XACT_BEGINS: 2
XACT_COMMITS: 1
XACT_D_COMMITS: 0
XACT_ROLLBACKS: 0
LOG_FORCES: 0
DEADLOCKS: 0
LOCK_TIMEOUTS: 0
LOCK_GRANTS_IMMED: 17
LOCK_GRANTS_WAIT: 0
SYS19: 0
CMD_PREPARES: 1
CMD_REPREPARES: 0
CMD_TEMP_INDEXES: 0
LAST_LOG_FILE: 0
REPHOLD_LOG_FILE: -1
REPHOLD_LOG_OFF: -1
REP_XACT_COUNT: 0
REP_CONFLICT_COUNT: 0
REP_PEER_CONNECTIONS: 0
REP_PEER_RETRIES: 0
FIRST_LOG_FILE: 0
LOG_BYTES_TO_LOG_BUFFER: 64
LOG_FS_READS: 0
LOG_FS_WRITES: 0
LOG_BUFFER_WAITS: 0
CHECKPOINT_BYTES_WRITTEN: 0
CURSOR_OPENS: 1
CURSOR_CLOSES: 1
SYS3: 0
SYS4: 0
SYS5: 0
SYS6: 0
CHECKPOINT_BLOCKS_WRITTEN: 0
CHECKPOINT_WRITES: 0
REQUIRED_RECOVERY: 0
```

```
SYS11: 0
SYS12: 1
TYPE_MODE: 0
SYS13: 0
SYS14: 0
SYS15: 0
SYS16: 0
SYS17: 0
SYS9:
```

List Database Objects by Object Type

You can use `ttIsql` to list tables, indexes, views, sequences, synonyms, PL/SQL functions, procedures, and packages in a database.

Commands prefixed by `all` display all of this type of object. For example, the `functions` command lists PL/SQL functions that are owned by the user, whereas `allfunctions` lists all PL/SQL functions.

> **Note**
>
> To run `all*` commands to see other user's object information, you need to have the permission to do so.

You can optionally specify patterns for object owners and object names.

Use these commands to list database objects:

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

The following example demonstrates the `procedures` and `allprocedures` commands. User `TERRY` creates a procedure called `proc1` while connected to `database1`. Note that a slash character (/) is entered on a new line following the PL/SQL statements.

The `procedures` command and the `allprocedures` command show that it is the only PL/SQL procedure in the database.

```
$ ttIsql database1
Copyright (c) Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.
connect "DSN=database1";
Connection successful:
DSN=database1;UID=Terry;DataStore=/disk1/databases/
database1;DatabaseCharacterSet=AL32UTF8;
ConnectionCharacterSet=AL32UTF8;PermSize=128;
```

```
(Default setting AutoCommit=1)
Command>create or replace procedure proc1 as begin null; end;
       > /
Procedure created.
Command>procedures;
  TERRY.PROC1
1 procedure found.
Command>allprocedures;
  TERRY.PROC1
1 procedure found.
```

Now connect to the same DSN as `PAT` and create a procedure called `q`. The `allprocedures` command shows the PL/SQL procedures created by `Terry` and `PAT`.

```
$ ttIsql "DSN=database1;UID=PAT"
Copyright (c) Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.
connect "DSN=database1;UID=PAT";
Connection successful: DSN=database1;UID=PAT;
DataStore=/disk1/databases/database1;DatabaseCharacterSet=AL32UTF8;
ConnectionCharacterSet=AL32UTF8;PermSize=128;
(Default setting AutoCommit=1)
Command>create or replace procedure q as begin null; end;
       > /
Procedure created.
Command>procedures;
  PAT.Q
1 procedure found.
Command>allprocedures;
  TERRY.PROC1
  PAT.Q
2 procedures found.
```

In this example, `PAT` is able to see the `TERRY's` procedure, it cannot happen unless `PAT` has the permission.

View and Change Query Optimizer Plans

You can view the query optimizer plans, commands in the SQL command cache, or query plans for commands in the SQL command cache.

- Use the showplan Command
- View Commands and Explain Plans from the SQL Command Cache

Use the showplan Command

The built-in `showplan` command is used to display the query optimizer plans used by TimesTen for processing queries.

In addition, `ttIsql` contains built-in query optimizer hint commands for altering the query optimizer plan. By using the `showplan` command in conjunction with the `ttIsql` commands summarized below, the optimum execution plan can be designed.

- `optprofile` - Displays the current optimizer hint settings and join order.
- `setjoinorder` - Sets the join order.
- `setuseindex` - Sets the index hint.
- `tryhash` - Enables or disables the use of hash indexes.

- `trymergejoin` - Enables or disables merge joins.
- `trynestedloopjoin` - Enables or disables nested loop joins.
- `tryserial` - Enables or disables serial scans.
- `trytmphash` - Enables or disables the use of temporary hash indexes.
- `trytmptable` - Enables or disables the use of an intermediate results table.
- `trytmprange` - Enables or disables the use of temporary range indexes.
- `tryrange` - Enables or disables the use of range indexes.
- `tryrowid` - Enables or disables the use of rowid scans.
- `trytbllocks` - Enables or disables the use of table locks.
- `unsetjoinorder` - Clears the join order.
- `unsetuseindex` - Clears the index hint.

When using the `showplan` command and the query optimizer hint commands, the autocommit feature must be turned off. Use the `ttIsql` `autocommit` command to turn off `autocommit`.

This example shows how these commands can be used to change the query optimizer execution plan.

```
Command>CREATE TABLE T1 (A NUMBER);
Table created.
Command>CREATE TABLE T2 (B NUMBER);
Table created.
Command>CREATE TABLE T3 (C NUMBER);
Table created.
Command>INSERT INTO T1 VALUES (3);
1 row inserted.
Command>INSERT INTO T2 VALUES (3);
1 row inserted.
Command>INSERT INTO T3 VALUES (3);
1 row inserted.
Command>INSERT INTO T1 VALUES (4);
1 row inserted.
Command>INSERT INTO T2 VALUES (5);
1 row inserted.
Command>INSERT INTO T3 VALUES (6);
1 row inserted.
Command>autocommit 0;
Command>showplan;
Command>SELECT * FROM T1, T2, T3 WHERE A=B AND B=C;

Query Optimizer Plan:

  STEP:               1
  LEVEL:              3
  OPERATION:          TblLkSerialScan
  TBLNAME:            T2
  IXNAME:             <NULL>
  INDEXED CONDITION:  <NULL>
  NOT INDEXED:        <NULL>


  STEP:               2
  LEVEL:              3
  OPERATION:          TblLkSerialScan
  TBLNAME:            T3
  IXNAME:             <NULL>
```

```
  INDEXED CONDITION:   <NULL>
  NOT INDEXED:         T2.B = T3.C


  STEP:                3
  LEVEL:               2
  OPERATION:           NestedLoop
  TBLNAME:             <NULL>
  IXNAME:              <NULL>
  INDEXED CONDITION:   <NULL>
  NOT INDEXED:         <NULL>


  STEP:                4
  LEVEL:               2
  OPERATION:           TblLkSerialScan
  TBLNAME:             T1
  IXNAME:              <NULL>
  INDEXED CONDITION:   <NULL>
  NOT INDEXED:         T1.A = T2.B AND T1.A = T2.B


  STEP:                5
  LEVEL:               1
  OPERATION:           NestedLoop
  TBLNAME:             <NULL>
  IXNAME:              <NULL>
  INDEXED CONDITION:   <NULL>
  NOT INDEXED:         <NULL>

< 3, 3, 3 >
1 row found.

Command>trytbllocks 0;
Command>tryserial 0;
Command>SELECT * FROM T1, T2, T3 WHERE A=B AND B=C;

Query Optimizer Plan:

  STEP:                1
  LEVEL:               3
  OPERATION:           TmpRangeScan
  TBLNAME:             T2
  IXNAME:              <NULL>
  INDEXED CONDITION:   <NULL>
  NOT INDEXED:         <NULL>


  STEP:                2
  LEVEL:               3
  OPERATION:           RowLkSerialScan
  TBLNAME:             T3
  IXNAME:              <NULL>
  INDEXED CONDITION:   <NULL>
  NOT INDEXED:         T2.B = T3.C


  STEP:                3
  LEVEL:               2
  OPERATION:           NestedLoop
  TBLNAME:             <NULL>
  IXNAME:              <NULL>
```

```
  INDEXED CONDITION:   <NULL>
  NOT INDEXED:         <NULL>


  STEP:                4
  LEVEL:               2
  OPERATION:           RowLkSerialScan
  TBLNAME:             T1
  IXNAME:              <NULL>
  INDEXED CONDITION:   <NULL>
  NOT INDEXED:         T1.A = T2.B AND T1.A = T2.B


  STEP:                5
  LEVEL:               1
  OPERATION:           NestedLoop
  TBLNAME:             <NULL>
  IXNAME:              <NULL>
  INDEXED CONDITION:   <NULL>
  NOT INDEXED:         <NULL>

< 3, 3, 3 >
1 row found.
```

In this example, the query optimizer plan is displayed for a `Select` query. The first version of the query simply uses the query optimizer's default execution plan. However, in the second version, the `trytbllocks` and `tryserial` `ttIsql` built-in hint commands have been used to alter the query optimizer's plan. Instead of using serial scans and nested loop joins, the second version of the query uses temporary index scans, serial scans, and nested loops. The second version uses row lock instead of table lock because the optimizer hint for table lock is turned off.

In this way, the `showplan` command in conjunction with `ttIsql`'s built-in query optimizer hint commands can be used to quickly determine which execution plan should be used to meet the application requirements. The query optimizer generates the best query plan based on the statistics. When you observe the bad query performance, then you need to use the optimizer hints to change the query plan.

View Commands and Explain Plans from the SQL Command Cache

You can view commands and their explain plans.

The following sections describe how to view commands and their explain plans:

- View Commands in the SQL Command Cache
- Display Query Plan for Statement in SQL Command Cache

View Commands in the SQL Command Cache

The `cmdcache` command invokes the `ttSqlCmdCacheInfo` built-in procedure to display the contents of the TimesTen SQL command cache.

If you run the `cmdcache` command without parameters, the full SQL command cache contents are displayed. Identical to the `ttSqlCmdCacheInfo` built-in procedure, you can provide a command ID to specify a specific command to be displayed.

In addition, the `cmdcache` command can filter the results so that only those commands that match a particular owner or query text are displayed.

The syntax for the cmdcache command is as follows:

```
cmdcache [[by {sqlcmdid | querytext | owner}] <query_substring>
```

If you provide the owner parameter, the results are filtered by the owner, identified by the <*query_substring*>, displayed within each returned command. If you provide the querytext parameter, the results are filtered so that all queries are displayed that contain the substring provided within the <*query_substring*>. If only the <*query_substring*> is provided, such as cmdcache <*query_substring*>, the command assumes to filter the query text by the <*query_substring*>. See cmdcache command.

```
 Command> cmdcache profile;
< 552126952, -1, 0, 1, 0, 0, 4784, SYS, select pn.profile# from sys.profname$ pn where
pn.name = :profilename, 0, <NULL>, 0, 0, 0, 0, <NULL>, 0, 0, 0, 0 >
< 552131576, -1, 0, 1, 0, 0, 6088, SYS, select p.resource#, p.limit# from sys.profile$ p
where p.type# = 0 and p.profile# = :profileid, 0, <NULL>, 0, 0, 0, 0, <NULL>, 0, 0, 0, 0
>
< 552110408, -1, 1, 1, 0, 0, 7160, SYS, select u.user#, u.password, u.identification,
u.astatus, u.lcount, u.ltime, u.profile# from sys.user$ u where u.name = :name and
u.type# = 1, 1, 2024-09-24 19:48:13.409000, 0, 0, 0, 0, <NULL>, 1304, 1304, 0, 0 >
3 rows found.
```

Display Query Plan for Statement in SQL Command Cache

The ttIsql explain command displays the query plan for an individual command.

- If you provide a command ID from the SQL command cache, the explain command invokes the ttSqlCmdQueryPlan built-in procedure to display the query plan for an individual command in the TimesTen SQL command cache. If you want the explain plan displayed in a formatted method, run the explain command instead of calling the ttSqlCmdQueryPlan built-in procedure. Both provide the same information, but the ttSqlCmdQueryPlan built-in procedure provides the data in a raw data format.
- If you provide a SQL statement or the history item number, the explain command compiles the SQL statements necessary to display the explain plan for this particular SQL statement.

The syntax for the explain command is as follows:

```
explain [plan for] {[<Connid>.]<ttisqlcmdid> | sqlcmdid <sqlcmdid> | <sqlstmt>
| !<historyitem>}
```

Identical to the ttSqlCmdQueryPlan built-in procedure, you can provide a command ID to specify a specific command to be displayed. The command ID can be retrieved with the cmdcache command, as described in View Commands in the SQL Command Cache.

The following example provides an explain plan for command ID 38001456:

```
Command>EXPLAIN SQLCMDID 38001456;

Query Optimizer Plan:
 Query Text: select * from all_objects where object_name = 'DBMS_OUTPUT'

  STEP:             1
  LEVEL:            12
  OPERATION:        TblLkRangeScan
  TABLENAME:        OBJ$
  TABLEOWNERNAME:   SYS
  INDEXNAME:        USER$.I_OBJ
  INDEXEDPRED:
  NONINDEXEDPRED:   (RTRIM( NAME ))  = DBMS_OUTPUT;NOT( 10 = TYPE#) ;
```

```
(  FLAGS ^ 128 = 0) ;


  STEP:             2
  LEVEL:            12
  OPERATION:        RowLkRangeScan
  TABLENAME:        OBJAUTH$
  TABLEOWNERNAME:   SYS
  INDEXNAME:        OBJAUTH$.I_OBJAUTH1
  INDEXEDPRED:      ( (GRANTEE#=1 )  OR (GRANTEE#=10 ) )  AND ( (PRIVILEGE#=8 ) )
  NONINDEXEDPRED:   OBJ# = OBJ#;


  STEP:             3
  LEVEL:            11
  OPERATION:        NestedLoop(Left OuterJoin)
  TABLENAME:
  TABLEOWNERNAME:
  INDEXNAME:
  INDEXEDPRED:
  NONINDEXEDPRED:
...
 STEP:             21
  LEVEL:            1
  OPERATION:        Project
  TABLENAME:
  TABLEOWNERNAME:
  INDEXNAME:
  INDEXEDPRED:
  NONINDEXEDPRED:

Command>
```

In addition, the `ttIsql` `explain` command can generate an explain plan for any SQL query you provide. For example, the following shows the explain plan for the SQL query: "`SELECT * FROM employees;`"

```
Command>EXPLAIN SELECT * FROM employees;

Query Optimizer Plan:

  STEP:               1
  LEVEL:              1
  OPERATION:          TblLkRangeScan
  TBLNAME:            EMPLOYEES
  IXNAME:             EMP_NAME_IX
  INDEXED CONDITION:  <NULL>
  NOT INDEXED:        <NULL>
```

You can also retrieve explain plans based upon the command history. The following example shows how you explain a previously run SQL statement using the history command ID:

```
Command>SELECT * FROM all_objects WHERE object_name = 'DBMS_OUTPUT';
< SYS, DBMS_OUTPUT, <NULL>, 241, <NULL>, PACKAGE, 2009-10-13 10:41:11, 2009-10-13
10:41:11, 2009-10-13:10:41:11, VALID, N, N, N, 1, <NULL> >
< PUBLIC, DBMS_OUTPUT, <NULL>, 242, <NULL>, SYNONYM, 2009-10-13 10:41:11,
2009-10-13 10:41:11, 2009-10-13:10:41:11, INVALID, N, N, N, 1, <NULL> >
< SYS, DBMS_OUTPUT, <NULL>, 243, <NULL>, PACKAGE BODY, 2009-10-13 10:41:11,
2009-10-13 10:41:11, 2009-10-13:10:41:11, VALID, N, N, N, 2, <NULL> >
3 rows found.
Command>HISTORY;
1     connect "DSN=cache";
```

```
2     help cmdcache;
3     cmdcache;
4     explain select * from dual;
5     select * from all_objects where object_name = 'DBMS_OUTPUT';
Command>EXPLAIN !5;

Query Optimizer Plan:

  STEP:               1
  LEVEL:              10
  OPERATION:          TblLkRangeScan
  TBLNAME:            SYS.OBJ$
  IXNAME:             USER$.I_OBJ
  INDEXED CONDITION:  <NULL>
  NOT INDEXED:        O.FLAGS & 128 = 0 AND CAST(RTRIM (O.NAME) AS VARCHAR2(30
BYTE) INLINE) = 'DBMS_OUTPUT' AND O.TYPE# <> 10

  STEP:               2
  LEVEL:              10
  OPERATION:          RowLkRangeScan
  TBLNAME:            SYS.OBJAUTH$
  IXNAME:             OBJAUTH$.I_OBJAUTH1
  INDEXED CONDITION:  (OA.GRANTEE# = 1 OR OA.GRANTEE# = 10) AND OA.PRIVILEGE# = 8
  NOT INDEXED:        OA.OBJ# = O.OBJ#

  STEP:               3
  LEVEL:              9
  OPERATION:          NestedLoop(Left OuterJoin)
  TBLNAME:            <NULL>
  IXNAME:             <NULL>
  INDEXED CONDITION:  <NULL>
  NOT INDEXED:        <NULL>

  STEP:               4
  LEVEL:              9
  OPERATION:          TblLkRangeScan
  TBLNAME:            SYS.OBJAUTH$
  IXNAME:             OBJAUTH$.I_OBJAUTH1
  INDEXED CONDITION:  (OBJAUTH$.GRANTEE# = 1 OR OBJAUTH$.GRANTEE# = 10) AND
(OBJAUTH$.PRIVILEGE# = 2 OR OBJAUTH$.PRIVILEGE# = 3 OR OBJAUTH$.PRIVILEGE# = 4 OR
OBJAUTH$.PRIVILEGE# = 5 OR OBJAUTH$.PRIVILEGE# = 8)
  NOT INDEXED:        O.OBJ# = OBJAUTH$.OBJ#
...
  STEP:               19
  LEVEL:              1
  OPERATION:          NestedLoop(Left OuterJoin)
  TBLNAME:            <NULL>
  IXNAME:             <NULL>
  INDEXED CONDITION:  <NULL>
  NOT INDEXED:        O.OWNER# = 1 OR (O.TYPE# IN (7,8,9) AND (NOT( ISNULLROW
(SYS.OBJAUTH$.ROWID)) OR NOT( ISNULLROW (SYS.SYSAUTH$.ROWID)))) OR (O.TYPE# IN
(1,2,3,4,5) AND NOT( ISNULLROW (SYS.SYSAUTH$.ROWID))) OR (O.TYPE# = 6 AND NOT(
ISNULLROW (SYS.SYSAUTH$.ROWID))) OR (O.TYPE# = 11 AND NOT( ISNULLROW
(SYS.SYSAUTH$.ROWID))) OR (O.TYPE# NOT IN (7,8,9,11) AND NOT( ISNULLROW
(SYS.OBJAUTH$.ROWID))) OR (O.TYPE# = 28 AND NOT( ISNULLROW (SYS.SYSAUTH$.ROWID)))
OR (O.TYPE# = 23 AND NOT( ISNULLROW (SYS.SYSAUTH$.ROWID))) OR O.OWNER# = 10
```

Create and Run PL/SQL Blocks Within ttIsql

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

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

```
set serveroutput on
```

Create an anonymous block that puts a text line in the output buffer. Note that the block must be terminated with a slash (/).

```
Command> BEGIN
DBMS_OUTPUT.put_line('Welcome!');
END;
/
Welcome!
PL/SQL procedure successfully completed.
Command>
```

See PL/SQL Blocks in the *PL/SQL Developer's Guide*. For information on error handling in `ttIsql` for PL/SQL objects, see Showing Errors in ttIsql in the *PL/SQL Developer's Guide*.

Pass Data From PL/SQL Using OUT Parameters Within ttIsql

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

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

Create the `tt_space_info` PL/SQL procedure and use SQL to provide values for the `permpct`, `permmaxpct`, `temppct`, and `tempmaxpct` parameters.

```
Command> CREATE OR REPLACE PROCEDURE tt_space_info
(permpct    OUT PLS_INTEGER,
permmaxpct OUT PLS_INTEGER,
temppct    OUT PLS_INTEGER,
tempmaxpct OUT PLS_INTEGER) AS
monitor    sys.monitor%ROWTYPE;
BEGIN
SELECT * INTO monitor FROM sys.monitor;
permpct := monitor.perm_in_use_size * 100 /
monitor.perm_allocated_size;
permmaxpct := monitor.perm_in_use_high_water * 100 /
monitor.perm_allocated_size;
temppct := monitor.temp_in_use_size * 100 /
monitor.temp_allocated_size;
tempmaxpct := monitor.temp_in_use_high_water * 100 /
monitor.temp_allocated_size;
END;
/

Procedure created.
```

Declare the variables and call `tt_space_info`. The parameter values are passed back to `ttIsql` so they can be printed:

```
Command> VARIABLE permpct NUMBER
Command> VARIABLE permpctmax NUMBER
Command> VARIABLE temppct NUMBER
Command> VARIABLE temppctmax NUMBER
Command> BEGIN
tt_space_info(:permpct, :permpctmax, :temppct, :temppctmax);
END;
/

PL/SQL procedure successfully completed.
```

```
Command> PRINT permpct;
PERMPCT              : 4

Command> PRINT permpctmax;
PERMPCTMAX           : 4

Command> PRINT temppct;
TEMPPCT              : 11

Command> PRINT temppctmax;
TEMPPCTMAX           : 11
```

You can also pass back a statement handle that can be run by a PL/SQL statement with an `OUT` refcursor parameter. The PL/SQL statement can choose the query associated with the cursor. The following example opens a refcursor, which randomly chooses between ascending or descending order.

```
Command> VARIABLE ref REFCURSOR;
Command> BEGIN
IF (mod(dbms_random.random(), 2) = 0) THEN
open :ref for select object_name from SYS.ALL_OBJECTS order by 1 asc;
ELSE
open :ref for select object_name from SYS.ALL_OBJECTS order by 1 desc;
end if;
END;
/

PL/SQL procedure successfully completed.
```

To fetch the result set from the refcursor, use the `PRINT` command:

```
Command> PRINT ref;
REF           :
< ACCESS$ >
< ALL_ARGUMENTS >
< ALL_COL_PRIVS >
< ALL_DEPENDENCIES >
...
143 rows found.
```

Or if the result set was ordered in descending order, the following would print:

```
Command> PRINT ref;
REF           :
< XLASUBSCRIPTIONS >
< WARNING_SETTINGS$ >
< VIEWS >
...
143 rows found.
```

Manage ODBC Functions

You can perform the following on ODBC functions within `ttIsql`:

- Canceling ODBC Functions
- Timing ODBC Function Calls

Canceling ODBC Functions

The `ttIsql` command attempts to cancel an ongoing ODBC function when the user presses `Ctrl-C`.

Timing ODBC Function Calls

Information on the time required to run common ODBC function calls can be displayed by using the `timing` command.

When the timing feature is enabled, many built-in `ttIsql` commands report the elapsed runtime associated with the primary ODBC function call corresponding to the `ttIsql` command.

In this example, when running the `connect` command several ODBC function calls run, however, the primary ODBC function call associated with `connect` is `SQLDriverConnect` and this is the function call that is timed and reported.

```
Command> timing 1;
Command> connect "DSN=database1";
Connection successful:
DSN=database1;DataStore=/disk1/databases/database1;DatabaseCharacterSet=AL32UTF8;
ConnectionCharacterSet=AL32UTF8;PermSize=128;
(Default setting AutoCommit=1)
Execution time (SQLDriverConnect) = 1.2626 seconds.
Command>
```

The `SQLDriverConnect` call took about 1.26 seconds to run.

When using the `timing` command to measure queries, the time required to run the query plus the time required to fetch the query results is measured. To avoid measuring the time to format and print query results to the display, set the `verbosity` level to `0` before running the query.

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

Command> timing 1;
Command> prepare 2 select 2 from dual;
Execution time (SQLPrepare) = 0.000385 seconds.
Command> exec 2;
Execution time (SQLExecute) = 0.000042 seconds.
Command> verbosity 0;
Command> fetchall 2;
Execution time (Fetch Loop) = 0.000050 seconds.
```

'editline' Feature for Linux and UNIX Only

On Linux and UNIX systems, you can use the 'editline' library to set up emacs (default) or vi bindings that enable you to scroll through previous `ttIsql` commands, as well as edit and resubmit them.

This feature is not available or needed on Windows.

To disable the 'editline' feature in `ttIsql`, use the `ttIsql` command `set editline off`.

The set up and keystroke information is described for each type of editor:

- Emacs Binding
- vi Binding

Emacs Binding

To use the emacs binding, create a file `~/.editrc` and put "`bind`" on the last line of the file, run `ttIsql`. The editline lib prints the current bindings.

The keystrokes when using `ttIsql` with the emacs binding are:

Keystroke	Action
<Left-Arrow>	Move the insertion point left (back up).
<Right-Arrow>	Move the insertion point right (move forward).
<Up-Arrow>	Scroll to the command prior to the one being displayed. Places the cursor at the end of the line.
<Down-Arrow>	Scroll to a more recent command history item and put the cursor at the end of the line.
<Ctrl-A>	Move the insertion point to the beginning of the line.
<Ctrl-E>	Move the insertion point to the end of the line.
<Ctrl-K>	Save and erase the characters on the command line from the current position to the end of the line.
<Ctrl-Y>	"Yank" (Restore) the characters previously saved and insert them at the current insertion point.
<Ctrl-F>	Forward char - move forward 1 (see Right Arrow).
<Ctrl-B>	Backward char - move back 1 (see Left Arrow).
<Ctrl-P>	Previous History (see Up Arrow).
<Ctrl-N>	Next History (see up Down Arrow).
<ESC-k> or <Ctrl-Up-Arrow>	Scroll up one line to edit within a multiple line PL/SQL block.
<ESC-j> or <Ctrl-Down-Arrow>	Scroll down one line to edit within a multiple line PL/SQL block.

vi Binding

To use the vi bindings, create a file `~/.editrc` and put "`bind -v`" in the file, run `ttIsql`.

To get the current settings, create a file `${HOME}/.editrc` and put "`bind`" on the last line of the file. When you run `ttIsql`, the editline lib prints the current bindings.

The keystrokes when using `ttIsql` with the vi binding are:

Keystroke	Action
<Left-Arrow>, h	Move the insertion point left (back up).
<Right-Arrow>, l	Move the insertion point right (forward).
<Up-Arrow>, k	Scroll to the prior command in the history and put the cursor at the end of the line.
<Down-Arrow>, j	Scroll to the next command in the history and put the cursor at the end of the line.

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

3
ttIsql Command Reference

This chapter contains the syntax, description, and examples of the `ttIsql` commands listed alphabetically.

Each command description contains the following parts:

- **Syntax** - Shows how to enter the command.
- **Description** - Provides a brief description of the basic uses of the command.
- **Examples** - Gives one or more examples of the command.

accept

Syntax

```
accept variable [NUM[BER] | CHAR | BINARY_FLOAT | BINARY_DOUBLE] [DEF[AULT] default]
[PROMPT text|NOPR[OMPT]] [HIDE]
```

Description

Gets input from a user and `DEFINES` the variable. If a type is specified then it validates for that type. The default (enclosed in quotes) is assigned if the user just presses enter. The prompt is displayed before waiting for input (or can be suppressed). The `HIDE` option stops the terminal from displaying the entered text (for passwords).

The prompt is displayed before waiting for input, if specified without the `HIDE` option.

Examples

To display the prompt `Password:`, place the reply in a `CHAR` variable named `PSWD`, and suppress the display, enter:

```
accept pswd CHAR PROMPT 'Password:  ' HIDE
```

To display the prompt `Enter weekly salary:` and place the reply in a `NUMBER` variable named `SALARY` with a default of `000.0`, enter:

```
accept salary NUMBER FORMAT '999.99' DEFAULT '000.0' -
PROMPT 'Enter weekly salary:  '
```

To display the prompt `Enter employee lastname:` and place the reply in a `CHAR` variable named `LASTNAME`, enter:

```
accept lastname CHAR FORMAT 'A20' -
PROMPT 'Enter employee lastname:  '
```

allfunctions

Syntax

```
allfunctions [[owner_name_pattern.] table_name_pattern]
```

Description

Lists, in a single column, the names of all the PL/SQL functions that match the given pattern selected from SYS.ALL_OBJECTS. When a pattern is missing, the pattern defaults to "%".

If passthrough is enabled, lists PL/SQL functions matching the pattern in the Oracle database.

For more details, see List Database Objects by Object Type.

Examples

```
Command> allfunctions;
  SYS.ORA_STRING_DISTANCE
  SYS.TT_COMPLEXITY_CHECK
  SYS.TT_STIG_VERIFY_FUNCTION
  SYS.TT_STRONG_VERIFY_FUNCTION
  SYS.TT_VERIFY_FUNCTION
  USER.F
6 functions found.
```

This example displays the functions that matches the given pattern.

```
Command> allfunctions %TT%;
  SYS.TT_COMPLEXITY_CHECK
  SYS.TT_STIG_VERIFY_FUNCTION
  SYS.TT_STRONG_VERIFY_FUNCTION
  SYS.TT_VERIFY_FUNCTION
4 functions found.
```

allindexes

Syntax

```
allindexes [[owner_name_pattern.] table_name_pattern]
```

Description

Describes the indexes that it finds on the tables that match the input pattern selected from SYS.ALL_OBJECTS. When a pattern is missing, the patterns default to "%".

If passthrough is enabled, lists indexes on tables matching the pattern in the Oracle database.

For more details, see List Database Objects by Object Type.

Examples

```
Command> allindexes;
Indexes on system table SYS.ACCESS$:
 I_ACCESS1: non-unique range index on columns:
    D_OBJ#
    ORDER#
  1 index found.
Indexes on system table SYS.ARGUMENT$:
  I_ARGUMENT1: unique range index on columns:
    OBJ#
    PROCEDURE$
    OVERLOAD#
    SEQUENCE#
  I_ARGUMENT2: non-unique range index on columns:
    OBJ#
```

```
    PROCEDURE#
    SEQUENCE#
  2 indexes found.
…..
118 indexes found on 122 tables.
```

allpackages

Syntax

```
allpackages [[owner_name_pattern.] table_name_pattern]
```

Description

Lists, in a single column, the names of all the PL/SQL packages that match the given pattern selected from SYS.ALL_OBJECTS. When a pattern is missing, the patterns default to "%".

If passthrough is enabled, lists PL/SQL packages matching the pattern in the Oracle database.

For more details, see List Database Objects by Object Type.

Examples

```
Command> allpackages;
  SYS.DBMS_LOB
  SYS.DBMS_LOCK
  SYS.DBMS_OUTPUT
  SYS.DBMS_PREPROCESSOR
  SYS.DBMS_RANDOM
  SYS.DBMS_SQL
  SYS.DBMS_STANDARD
  SYS.DBMS_SYS_ERROR
  SYS.DBMS_SYS_SQL
  SYS.DBMS_UTILITY
  SYS.PLITBLM
  SYS.STANDARD
  SYS.SYS_STUB_FOR_PURITY_ANALYSIS
  SYS.TT_DB_VERSION
  SYS.TT_STATS
  SYS.TT_STATS2
  SYS.UTL_FILE
  SYS.UTL_IDENT
  SYS.UTL_RAW
  SYS.UTL_RECOMP
20 packages found.
```

allprocedures

Syntax

```
allprocedures [[owner_name_pattern.] procedure_name_ pattern]
```

Description

Lists, in a single column, the names of all the PL/SQL procedures that match the given pattern selected from SYS.ALL_OBJECTS. When a pattern is missing, the pattern defaults to "%".

If passthrough is enabled, lists PL/SQL procedures matching the pattern in the Oracle database.

For more details, see List Database Objects by Object Type.

Examples

```
Command> allprocedures;
TERRY.PROC1
HERRY.PROC2
2 procedures found.
```

allsequences

Syntax

```
allsequences [[owner_name_pattern.] table_name_pattern]
```

Description

Lists, in a single column, the names of all the sequences that match the given pattern selected from SYS.ALL_OBJECTS. When a pattern is missing, the pattern defaults to "%".

If passthrough is enabled, lists sequences on tables matching the pattern in the Oracle database.

For more details, see List Database Objects by Object Type.

Examples

```
Command> allsequences;
  SYS.OBJECTSEQUENCE
  SYS.OBJECT_GRANT
  SYS.PROFNUM$
  SYS.SPID
  SYS.SYSTEM_GRANT
  SYS.USERSEQUENCE
6 sequences found.
```

allsynonyms

Syntax

```
allsynonyms [[schema_pattern.] object_pattern]
```

Description

Lists, in a single column, the names of all synonyms that match the given pattern. When a pattern is missing, the pattern defaults to "%".

If passthrough is enabled, lists synonyms on tables matching the pattern in the Oracle database.

Examples

```
Command> allsynonyms;
  PUBLIC.ALL_ARGUMENTS
  PUBLIC.ALL_COL_PRIVS
  PUBLIC.ALL_DEPENDENCIES
  PUBLIC.ALL_DIRECTORIES
  PUBLIC.ALL_ERRORS
  PUBLIC.ALL_EXTERNAL_TABLES
  PUBLIC.ALL_IDENTIFIERS
```

```
  PUBLIC.ALL_OBJECTS
….
```

alltables

Syntax

```
alltables [[owner_name_pattern.] table_name_pattern]
```

Description

Lists, in a single column, the names of all the tables that match the given pattern selected from SYS.ALL_OBJECTS. When a pattern is missing, the pattern defaults to "%".

If passthrough is enabled, lists tables matching the pattern in the Oracle database.

For more details, see List Database Objects by Object Type.

Examples

```
Command> alltables;
  SYS.ACCESS$
  SYS.ARGUMENT$
  SYS.CACHE_GROUP
…
```

allviews

Syntax

```
allviews [[owner_name_pattern.] view_name_pattern]
```

Description

Lists, in a single column, the names of all the views that match the specified pattern selected from SYS.ALL_OBJECTS. When a pattern is missing, the pattern defaults to "%".

If passthrough is enabled, lists views matching the pattern in the Oracle database.

For more details, see List Database Objects by Object Type.

Examples

```
Command> allviews;
  SYS.ALL_ARGUMENTS
  SYS.ALL_COL_PRIVS
  SYS.ALL_DEPENDENCIES
```

builtins

Syntax

```
builtins [builtin_name_ pattern]
```

Description

Lists, in a single column, the names of all the TimesTen built-in procedures that match the given pattern. When the pattern is missing, the pattern defaults to "%". See List of Built-In Procedures.

Examples

```
Command> builtins;
  TTAGINGLRUCONFIG
  TTAGINGSCHEDULENOW
  TTAGINGTABLELRUCONFIG
  TTAPPLICATIONCONTEXT
  TTBACKUPSTATUS
  TTBLOCKINFO
  TTBOOKMARK
  TTCACHEADGSTANDBYSTATEGET
  TTCACHEADGSTANDBYSTATESET
  TTCACHEADGSTANDBYTIMEOUTGET
  TTCACHEADGSTANDBYTIMEOUTSET
  TTCACHEALLOWFLUSHAWTSET
  TTCACHEAUTOREFINTERVALSTATSGET
  TTCACHEAUTOREFRESH
  TTCACHEAUTOREFRESHLOGDEFRAG
  TTCACHEAUTOREFRESHSELECTLIMIT
….…...
141 procedures found.

Command> builtins %GET
  TTCACHEADGSTANDBYSTATEGET
  TTCACHEADGSTANDBYTIMEOUTGET
  TTCACHEAUTOREFINTERVALSTATSGET
  TTCACHEAUTOREFRESHSTATSGET
  TTCACHEAWTTHRESHOLDGET
  TTCACHECONNPOOLGET
  TTCACHEPOLICYGET
  TTCACHESQLGET
  TTCACHEUIDGET
  TTCKPTCONFIGGET
  TTDBWRITECONCURRENCYMODEGET
  TTDISTADVICECAPTUREINFOGET
  TTEPOCHSESSIONGET
  TTHOSTNAMEGET
  TTINDEXADVICECAPTUREINFOGET
  TTLATCHSTATSGET
  TTRAMPOLICYAUTORELOADGET
  TTRAMPOLICYGET
  TTREPPOLICYGET
  TTREPQUERYTHRESHOLDGET
  TTREPSTATEGET
  TTREPSYNCGET
  TTREPTRANSMITGET
  TTREPXACTTOKENGET
  TTSQLCMDCACHEINFOGET
  TTSTATSCONFIGGET
  TTTABLESCHEMAFROMORAQUERYGET
  TTXACTIDGET
28 procedures found.
```

bye

Syntax

```
bye or exit
```

Description

Exits `ttIsql`.

Example

```
Command> exit;
Disconnecting...
Done.
```

cachegroups

Syntax

```
cachegroups [[cache_group_owner_pattern.] cache_group_name_pattern]
```

Description

Reports information on cache groups defined in the currently connected data source, including the state of any terminated databases that contain autorefresh cache groups.

If the optional argument is not specified then information on all cache groups in the current data source is reported.

Examples

```
Command> cachegroups;
 Cache Group CACHEUSER.READCACHE:
 Cache Group Type: Read Only
 Autorefresh: Yes
 Autorefresh Mode: Incremental
 Autorefresh State: Paused
 Autorefresh Interval: 5 Seconds
 Autorefresh Status: ok
 Aging: No aging defined
 Root Table: SALES.READTAB
 Table Type: Read Only
1 cache group found.

Command> cachegroups SCOTT.%;
 Cache Group SCOTT.MYCACHEGROUP:
 Cache Group Type: Read Only
 Autorefresh: Yes
 Autorefresh Mode: Incremental
 Autorefresh State: Paused
 Autorefresh Interval: 5 Seconds
 Autorefresh Status: ok
 Aging: No aging defined
 Root Table: SALES.READTAB
 Table Type: Read Only
1 cache group found.
```

For log-based cache group, see Task 3: Create TimesTen Users and Tables in the *Oracle TimesTen In-Memory Database Cache Guide*.

cachesqlget

Syntax

```
cachesqlget [ASYNCHRONOUS_WRITETHROUGH | CACHE_CONFIG |INCREMENTAL_AUTOREFRESH |
ORACLE_DDL_TRACKING ][[<cache_group_owner>.]<cache_group_name>] <INSTALL|UNINSTALL>
[<filename>]
```

Description

Generates an Oracle SQL*Plus compatible script for the installation or uninstallation of Oracle database objects associated with a readonly cache group, a user managed cache group with incremental autorefresh or an AWT cache group.

If `INSTALL` is specified, the Oracle SQL statement to install the Oracle database objects is generated.

If `UNINSTALL` is specified, the Oracle SQL statement used to remove the Oracle objects is generated. If a cache group is not specified with `UNINSTALL`, a SQL statement to remove all Oracle database objects in the autorefresh user's account is generated.

If the optional *`filename`* argument is included, the generated SQL statement is saved to the specified external file. If the external file exists, its contents are destroyed before writing to the file.

Examples

```
Command> cacheSqlGet SCOTT.MYCACHEGROUP INSTALL create.sql;
Command> cacheSqlGet ASYNCHRONOUS_WRITETHROUGH INSTALL create.sql;
Command> cacheSqlGet UNINSTALL;
```

cd

Syntax

```
cd directory
```

Description

Changes the current directory. This is the equivalent of the `cd` command in the interactive shells.

After changing to the directory *`directory`*, the define alias `_CWD` will be set to this directory.

Subsequent commands that rely on relative paths will use this directory as the starting point.

Examples of affected commands are `spool`, `run`, `savehistory`, `host`, and `edit`.

Example

```
Command> host pwd;
/tmp/dir1
Command> cd /tmp/dir2;
Command> host pwd;
/tmp/dir2
```

clearhistory

Syntax

```
clearhistory
```

Description

Clears the history buffer.

Example

```
Command> history
1     connect sampledb;
2     ls
Command> clearhistory;
Command> history;
```

clienttimeout

Syntax

```
clienttimeout [timeout seconds]
```

Description

Sets the client timeout value in seconds for the current connection. If no value is specified, displays the current value. The client timeout can take a value between 0 and 99999 inclusive.

The `clienttimeout` has the same value as the client attribute in the `ttc_timeout`. It is the client timeout for the client server.

Examples

```
Command>show clienttimeout;
Client timeout = 0

Command>set clienttimeout 10;
```

close

Syntax

```
close [[connect_id.] command_id]

closeall
```

Description

Closes the prepared command identified by connection name *`connect_id`* and command ID *`command_id`*. If *`command_id`* is not specified, closes the most recent command. If `closeall` is selected, closes all currently open prepared commands.

For examples, see Prepare a SQL Statement for Subsequent Runs.

cmdcache

Syntax

```
cmdcache [[by {sqlcmdid |querytext|owner}] query_subsstring]
```

Description

Displays the contents of the TimesTen SQL command cache.

Specify the `sqlcmdid`, `querytext` or `owner` column and query substring to search for a specific portion of a SQL query. If no column is specified, searches the `querytext` column.

If passthrough is enabled, the command ID is not passed through to the Oracle database.

Examples

```
Command> cmdcache;
< 147347040, -1, 1, 1, 0, 0, 8336, ADMIN, call ttsqlcmdcacheinfo(), 1, 2024-08-29
03:08:39.370000, 0, 0, 0, 0, <NULL>, 0, 0, 0, 0 >
< 147586432, -1, 2, 2, 0, 1, 3616, SYS, call ttCkptOneElement( , , 'ttCkptLoadPLSQL');,
0, 2024-08-29 03:08:37.465000, 0, 0, 0, 0, <NULL>, 24, 24, 0, 0 >
< 147393976, -1, 0, 1, 0, 0, 5472, SYS, update sys.user$ u set u.lcount = u.lcount + 1
where u.user# = :userid, 0, <NULL>, 0, <NULL>, 0, 0, <NULL>, 0, 0, 0, 0 >
< 147388856, -1, 0, 1, 0, 0, 5760, SYS, update sys.user$ u set u.lcount = 0, u.astatus
= :astatus where u.user# = :userid, 0, <NULL>, 0, <NULL>, 0, 0, <NULL>, 0, 0, 0, 0 >
< 147405768, -1, 0, 1, 0, 0, 5696, SYS, select u.password, u.password_date from
sys.user_history$ u where u.user# = :userid order by u.password_date desc, 0, <NULL>, 0,
<NULL>, 0, 0, <NULL>, 0, 0, 0, 0 >
< 143321728, -1, 0, 1, 0, 0, 5088, SYS, select pn.profile# from sys.profname$ pn where
pn.name = :profilename, 0, <NULL>, 0, <NULL>, 0, 0, <NULL>, 0, 0, 0, 0 >
< 147383224, -1, 0, 1, 0, 0, 6288, SYS, select p.resource#, p.limit# from sys.profile$ p
where p.type# = 0 and p.profile# = :profileid, 0, <NULL>, 0, <NULL>, 0, 0, <NULL>, 0, 0,
0, 0 >
< 147343648, -1, 1, 1, 0, 1, 3792, ADMIN, CALL ttconfiguration ('TTGrid'), 2, 2024-08-29
03:08:39.069000, 0, 0, 0, 0, <NULL>, 1148, 1148, 0, 0 >
< 154652272, -1, 6, 6, 0, 1, 8032, SYS, select
owner#,name,namespace,obj#,type#,ctime,mtime,stime,status,flags from sys.obj$ where
owner#=:1 and name=:2 and namespace=:3, 6, 2024-08-29 03:08:39.370000, 0, 0, 0, 0,
<NULL>, 1022, 1022, 0, 0 >
< 143311952, -1, 0, 1, 0, 0, 9808, SYS, select 1 from sys.sysauth$ s where (s.grantee#
= :userid or s.grantee# = 1) and (s.privilege# = :priv or s.privilege# = 67), 0, <NULL>,
0, <NULL>, 0, 0, <NULL>, 0, 0, 0, 0 >
< 147411432, -1, 0, 1, 0, 0, 5488, SYS, delete from user_history$ where user# = :userid
and password_date < :pdate, 0, <NULL>, 0, <NULL>, 0, 0, <NULL>, 0, 0, 0, 0 >
< 154654448, -1, 1, 1, 0, 1, 3376, ADMIN, CALL ttOptSetFlag ('passthrough', 0), 0,
2024-08-29 03:08:39.069000, 0, 0, 0, 0, <NULL>, 32, 32, 0, 0 >
< 147399464, -1, 0, 1, 0, 0, 6320, SYS, update sys.user$ u set u.lcount = u.lcount + 1,
u.ltime = sysdate, u.astatus = :astatus where u.user# = :userid, 0, <NULL>, 0, <NULL>,
0, 0, <NULL>, 0, 0, 0, 0 >
< 143304432, -1, 1, 1, 0, 0, 7408, SYS, select u.user#, u.password, u.identification,
u.astatus, u.lcount, u.ltime, u.profile# from sys.user$ u where u.name = :name and
u.type# = 1, 1, 2024-08-29 03:08:31.250000, 0, 0, 0, 0, <NULL>, 1312, 1312, 0, 0 >
14 rows found.
```

commit

Syntax

```
commit
```

Description

Commits the current transaction (durably if `Durability = 1` for the connection).

As opposed to Rollback which will undo the most recent uncommitted work.

For examples, see Manage Transactions.

commitdurable

Syntax

```
commitdurable
```

Description

Commits the current transaction durably and ensures that the committed work is recovered in case of database failure. The current transaction (and every previously committed transaction) is saved to the disk and is guaranteed to be recovered when it is required.

For examples, see Manage Transactions.

compare

Syntax

```
compare varA varB
```

Description

Compares the values of two variables and reports if they are different. The first difference is reported.

Examples

```
Command> var a varchar2(10) := 'ABC';
Command> var b varchar2(10) := 'ABC';
Command> var c varchar2(10) := 'ABC1';
Command> compare a a;
Bind variables "A" and "A" match.
Command> compare a b;
Bind variables "A" and "B" match.
Command> compare a c;
Bind variable "A" differs from "C" starting at offset 3.
A    : ABC
C    : ABC1
The command failed.
```

connect

Syntax

```
connect[connection_string |[[DSN][as]connid  [adding] [connection_string | DSN][as
connid]
```

Description

Connects to the database with the specified ODBC *connection_string*.

If no password is supplied in this format, `ttIsql` prompts for the password.

If no user is given, `ttIsql` attempts to connect using the user name of the current user as indicated by the operating system.

If `as` *connid* is specified, you can explicitly name the connection. The `connid` must be only alphanumeric characters, is case sensitive, must start with an alpha character and can only be a maximum of 30 characters in length. The name of `connid` is automatically supplied to the `ConnectionName` general connection attribute. If the connect fails, the current connection is set to a special reserved connection named "none," which is never connected to anything.

When `adding` is specified, it refers to creating a new connection to the DSN specified by *DSN* or by the connection string.

Examples

```
Command> connect "DSN=database1";
Connection successful:
DSN=database1;DataStore=/disk1/databases/database1;DatabaseCharacterSet=AL32UTF8;
ConnectionCharacterSet=AL32UTF8;PermSize=128;
(Default setting AutoCommit=1)
```

createandloadfromoraquery

Syntax

```
createandloadfromoraquery [owner_name.]table_name [num_threads] query
```

Description

Takes a table name, the number of threads for parallel load and an Oracle `SELECT` statement.

Creates the table in TimesTen if the table does not exist. Then, loads the table with the query result from the Oracle database. If the command creates the table, the table column names and types are derived from the query result.

Notes:

- The specified TimesTen table cannot be a system table, a synonym, a view, a materialized view or a detail table of a materialized view, a global temporary table or a cache group table.
- The query cannot have any parameter bindings.
- Any unsupported column types result in a warning being logged. The output issues a comment for the unsupported column data type.
- If you do not supply a value for `num_threads`, defaults to four threads.

- For details and usage information, see Loading Data from an Oracle Database into a TimesTen Table Without Cache in *Oracle TimesTen In-Memory Database Operations Guide*.
- You must rollback or commit after running this operation.
- Also see the NOTES section in the description of the built-in procedure ttLoadFromOracle.

Required Privileges:

Requires `INSERT` privilege on the table specified. Also, requires the `CREATE TABLE` privilege if the table does not exist. The Oracle session user must have all required privileges to run the query on the Oracle database.

Examples

```
Command> createandloadfromoraquery C1 'select * from PAT.C1';
```

define

Syntax

```
define name [= value]
```

Description

Defines a string substitution alias.

If no value is provided, `ttIsql` displays the current definition for the specified name.

You must `set define on` to enable command substitution.

`undefine name` undefines a string substitution alias.

Examples

```
Command> Set define on;
Command> define q = 'myvalue';
```

`define2var` defines the value and make it a variable that you can use as a parameter variable.

```
define2var myalias mybindvar
Command> define2var a1 arr[2];
```

`var2define` it takes the value of a named variable and then define it.

```
var2define mybindvar myalias
Command> var2define arr[2] alias2;
```

describe

Syntax

```
describe [[owner_pattern.] name_pattern | procedure_name_pattern |sql_statement |
[connect_id.]command_id |*]
```

Description

List information on tables, synonyms, views, materialized views, sequences, cache groups, PL/SQL functions, PL/SQL procedures, PL/SQL packages and TimesTen built-in procedures in

that order when the argument is [*owner_pattern*.]*name_pattern*. Otherwise lists the specific objects that match the given pattern.

Describes the parameters and results columns when the argument is *sql_statement*.

If passthrough is set to 3, lists information about the same types of objects in the Oracle database.

If * is specified, reports the prepared statements for all connections.

When describing cache groups, reports information on cache groups defined in the currently connected data source, including the state of any terminated databases that contain autorefresh cache groups.

The command alias is desc.

For examples, see Prepare a SQL Statement for Subsequent Runs and describe Command.

dssize

Syntax

```
dssize [k|m|g|t]
```

Description

Prints database size information in KB, MB, GB or TB.

The default is MB. The output indicates the unit returned.

Examples

```
Command> dssize m;
The following values are in MB:

  PERM_ALLOCATED_SIZE:      32
  PERM_IN_USE_SIZE:         19.085
  PERM_IN_USE_HIGH_WATER:   19.186
  TEMP_ALLOCATED_SIZE:      40
  TEMP_IN_USE_SIZE:         15.088
  TEMP_IN_USE_HIGH_WATER:   21.432
```

e:

Syntax

```
e: msg
PROMPT msg
```

Description

Echoes the specified messages, terminated by the end of the line. A semicolon is not required to end the line. Messages are not echoed if verbosity is set to 0.

Examples

```
Command> e: Hi There John.
Hi There John.
Command> e: Bill
Bill
```

```
Command> e: All done
All  done
Command>
```

edit

Syntax

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

Description

You can use the `edit` command to edit a file or edit in a text editor. The `edit` command starts a text editor such as `emacs`, `gedit`, or `vi`.

If TimesTen does not find an exact file match for the specified *`file`* parameter, it searches for *`file`*`.sql`. If neither file exists, `ttIsql` starts the editor with the file `file`.

You can edit a SQL statement that is stored in the history list of the current `ttIsql` session. When calling the `edit` command specify the `!` character followed by the number of the command or a search string.

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

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

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

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

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

exec

Syntax

```
exec [connect_id.] command_id] | PLSQLSTMT
```

Description

Runs the prepared command *`command_id`* on connection *`connect_id`* or runs a PL/SQL statement.

The *`connect_id`* optionally names a `ttIsql` connection and *`command_id`* is an integer from 1 to 255. If *`PLSQLSTMT`* is supplied, `ttIsql` prepends the statement with `BEGIN` and appends the statement with `END`, thus allowing the PL/SQL statement to run.

If no argument is supplied, runs the most recent command.

For examples, see Prepare a SQL Statement for Subsequent Runs.

execandfetch

Syntax

```
execandfetch [[connect_id.]command_id]
```

Description

Executes the specified prepared SQL statement and fetches all result rows associated with the statement.

The *connect_id* optionally names a `ttIsql` connection and *command_id* is an integer from 1 to 255. If *PLSQLSTMT* is supplied, `ttIsql` prepends the statement with `BEGIN` and appends the statement with `END`, thus allowing the PL/SQL statement to run.

If no argument is supplied, runs the most recent command.

For examples, see Prepare a SQL Statement for Subsequent Runs.

explain

Syntax

```
explain [plan for] {[ Connid.]ttisqlcmdid | sqlcmdid sqlcmdid | sqlstmt |!history}
```

Description

Explains the plan for the specified SQL statement, including prepared `ttIsql` statements, specified in the *ttisqlcmdid* argument, or the *sqlcmdid* argument.

A digit that is not qualified with the `sqlcmdid` argument, is interpreted as a `ttIsql` prepared statement ID.

If passthrough is enabled, the command ID is not passed through to the Oracle database.

Example

```
Command> explain plan for select * from dual;

Query Optimizer Plan (from Query Compilation):

  STEP:                1
  LEVEL:               1
  OPERATION:           RowLkSerialScan
  TBLNAME:             DUAL
  IXNAME:
  INDEXED CONDITION:
  NOT INDEXED:
  MISCELLANEOUS:       cardEst = 1
```

fetchall

Syntax

```
fetchall [connect_id.]command_id]
```

Description

Fetches all results from prepared command *command_id* on connection *connect_id*.

If *command_id* is not specified, fetches all results from the most recent command. The command must already have been run using exec.

For examples, see Prepare a SQL Statement for Subsequent Runs.

fetchnext

Syntax

```
fetchnext num_rows [connect_id.]command_id]
```

Description

Fetches up to *num_rows* rows from prepared command *command_id* on connection *connect_id*.

If *command_id* is not specified, fetches *num_rows* rows from the most recent command. The command must already have been run using exec.

Example

```
Command> create table t1 ( a int, b varchar(30));

Command> create sequence s;

Command> set autocommit 0;

Command> prepare 1 insert into t1 values( s.nextval, 'A');

Command> prepare 2 insert into t1 values( s.nextval, 'B');

Command> prepare 5 select * from t1;

Command> describe *;

There are 3 prepared commands.

Prepared Statement [1]:
  SQL: insert into t1 values( s.nextval, 'A')
  Columns:
    (none)

Prepared Statement [2]:
  SQL: insert into t1 values( s.nextval, 'B')
  Columns:
    (none)

Prepared Statement [5]:
  SQL: select * from t1
  Columns:
```

```
    A         NUMBER (38)
    B         VARCHAR2 (30)


Command> exec 1;
1 row inserted.

Command> exec 2;
1 row inserted.

Command> fetchnext 1;
< 1, A >
1 row found.

Command> fetchnext 1;
< 2, B >
1 row found.
```

fetchone

Syntax

```
fetchone [connect_id.]command_id]
```

Description

Fetches one result from prepared command *command_id* on connection *connect_id*.

If *command_id* is not specified, fetches one result from the most recent command. The command must already have been run using exec.

For examples, see Prepare a SQL Statement for Subsequent Runs.

free

Syntax

```
free [[connect_name.]connect_id.] command_id]
```

Description

Frees prepared command *command_id* on connection *connect_id*.

If no command is specified, frees the most recent command.

Use prepare to create the prepared command.

For examples, see Prepare a SQL Statement for Subsequent Runs.

help

Syntax

```
help [command [command ...]| all | comments | attributes]
```

Description

Prints brief or detailed help information for commands.

If specific commands are given as arguments, then detailed help for each command is printed.

If you do not know the exact name of a command, try typing just a few characters that may be part of the command name. `ttIsql` searches and displays help for any commands that include the characters.

If `all` is given as an argument, then detailed help for all commands is printed.

If `comments` is given as an argument, then information on using `ttIsql` comments within scripts is printed.

If `attributes` is given as an argument, then information on the `set`/`show` attributes is printed.

If no argument is given, then brief help information for all commands is printed. See Help Commands.

Examples

```
Command> help;
 Use "help all" to get a description of all commands or use "help <cmd>" to
limit it to that command.

?                       fetchnext               sqlcolumns
!                       free                    sqlgetinfo
@@                      functions               sqlquerytimeout
accept                  getenv                  sqlquerytimeoutmsec
allfunctions            help                    sqlstatistics
allindexes              history                 sqltables
allpackages             host                    statsclear
allprocedures           if                      statsestimate
allsequences            indexes                 statsupdate
allsynonyms             isolation               synonyms
alltables               monitor                 tables
allviews                multipleconnections     tblsize

….….….…                ….….….…                ….….….

Command> help explain;

Arguments in <> are required.
Arguments in [] are optional.


Command Usage: explain [plan for] {[<connection_name>.]<ttisqlcmdid> | sqlcmdid
<sqlcmdid> | <stmt> | !<historyitem> }
Command Aliases: (none)
Description: Explain a plan for either the prepared ttisql cmd, the sqlcmdid,
<sqlstmt>, or the statement from the given history item..  When the SQL text
is given this is shorthand for "autocommit 0; showplan 1; prepare <sqlstmt>;
restore autocommit and showplan." The literals "plan for" are not required for
TT, but are part of the Oracle RDBMS syntax; ttIsql will insert the keywords
'plan for' when passthrough=3. When a cmdid is given, the plan is retrieved
via the builtin ttsqlcmdqueryplan(). The cmdid form will not work with
passthrough > 0.
Requires an active connection: YES
Requires autocommit turned off: NO
Reports elapsed execution time: YES
Works only with a TimesTen data source: YES
Example: explain select * from dual; -or- explain plan for select * from dual;
```

history

Syntax

```
history [-all] [-h] [-r] [num_commands]
```

Description

`ttIsql` implements a `csh`-like command history.

Lists previously run commands. The *`num_commands`* parameter specifies the number of commands to list. If the *`num_commands`* parameter is omitted then the previous 10 commands are listed by default.

The output of this command omits consecutive duplicate commands. Use the `-all` option to include the consecutive duplicate commands.

Use the `-h` option to omit the command numbers.

Use the `-r` parameter to list the commands in reverse order.

The history list stores up to 100 of the most recently run commands. The history command by default displays the last 10 SQL statements or `ttIsql` built-in commands that were run. To display more than that last 10 commands, specify the maximum number to display as an argument to the history command.

Examples

```
Command> history;
1 INSERT INTO T3 VALUES (3)
2 INSERT INTO T1 VALUES (4)
3 INSERT INTO T2 VALUES (5)
4 INSERT INTO T3 VALUES (6)
5 autocommit 0
6 showplan
7 SELECT * FROM T1, t2, t3 WHERE A=B AND B=C AND A=B
8 trytbllocks 0
9 tryserial 0
10 SELECT * FROM T1, t2, t3 WHERE A=B AND B=C AND A=B
Command>
```

Each entry in the history list is identified by a unique number. The `!` character followed by the number of the command can be used to run the command again. For example:

```
Command> ! 5;
autocommit 0
```

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

```
Command> !!;
autocommit 0
```

To run the last command that begins with a given string type the `!` character followed by the first few letters of the command. For example:

```
Command> ! auto;
autocommit 0
```

host

Syntax

```
host <os_command>
```

Description

Runs an operating system command. The command is run in the same console as `ttIsql`.

This command sets the environment variable `TT_CONNSTR` in the environment of the process it creates.

The value of the variable is the connection string of the current connection.

To see the exit status of the command, use the `define` command with `_EXIT_STATUS`.

Examples

```
Command> host ls -1 *.hh;
Algorithms.hh
Timer.hh
a.hh

Command> host ls /dev/null;
/dev/null

Command> host vi script.sql;
```

if-then-else

Syntax

```
IF [NOT]
   { Literal1 | :BindVariable1 }
   { =  | IN }
   { Literal2 | :BindVariable2 | SelectStatement }
 THEN "ThenCommands"
 [ ELSE "ElseCommands" ] ;
```

The `ttIsql` `IF-THEN-ELSE` command has the parameters:

Parameter	Description
`IF`	The `IF` command must end in a semicolon (`;`). The `IF` command fails if improper syntax is given, the `BindVariables` do not exist or the `SELECT` statement fails to run or does not return just a single column.
`NOT`	Using `NOT` reverses the desired result of the condition.
`Literal1`, *`Literal2`*	A value that can be part of a comparison.
`BindVariable1`, *`BindVariable2`*	A bind variable is equivalent to a parameter. You can use the *`:BindVariable1`* notation for passing bind variables into this construct. The variable can be created and set using the `variable` or `setvariable` `ttIsql` commands.

Parameter	Description
= | IN	You can use the IN operator only with the *SelectStatement*. You can use the IN operator with zero or more returned rows. You can use the equal (=) operator only with a single returned row.
SelectStatement	A provided SELECT statement must start with SELECT. The SELECT statement can return only one column. In addition, it can return only one row when the equal (=) operator is provided. The *SelectStatement* is not available if you are not connected to the database.
ThenCommands, *ElseCommands*	All commands in the THEN or ELSE clauses must be delimited by a semicolon and cannot contain embedded double quotes. These clauses can conditionally run ttIsql commands, such as host or run, which cannot be run through PL/SQL. You can use the CALL statement within the THEN or ELSE clauses. You cannot use PL/SQL blocks.

Description

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

Example

```
Command> if :a = 1 then "e:a is 1"  else "e:a is not 1";
```

The following example creates and tests a bind variable to see which type of locking is enabled for the TimesTen database. It uses the autovariables command to create the bind variable from the result of the call to ttConfiguration. The value can be tested within the IF-THEN-ELSE conditional by testing the paramvalue variable.

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

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

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

Restrictions for the IF-THEN-ELSE construct are as follows:

- You cannot compare variables of the LOB data type.
- The values are compared case-sensitive with strcmp. A character padded value might not match a VARCHAR2 because of the padding.

monitor

Syntax

```
monitor [optional_monitor_column]
```

Description

Formats the contents of the `SYS.MONITOR` table for easy viewing.

If the *`optional_monitor_column`* is specified, only that column is displayed.

For details, see monitor Command.

Example

```
Command> monitor;
  TIME_OF_1ST_CONNECT:      Thu Sep  5 00:48:03 2024
  DS_CONNECTS:                 14
  DS_DISCONNECTS:              1
  DS_CHECKPOINTS:              4
  DS_CHECKPOINTS_FUZZY:        0
  DS_COMPACTS:                 0
  PERM_ALLOCATED_SIZE:         32768
  PERM_IN_USE_SIZE:            19567
  PERM_IN_USE_HIGH_WATER:      19646
  TEMP_ALLOCATED_SIZE:         40960
  TEMP_IN_USE_SIZE:            17665
  TEMP_IN_USE_HIGH_WATER:      21946
  SYS18:                       0
  TPL_FETCHES:                 0
  TPL_EXECS:                   0
  CACHE_HITS:                  0
  PASSTHROUGH_COUNT:           0
  XACT_BEGINS:                 435
  XACT_COMMITS:                434
  XACT_D_COMMITS:              124
  XACT_ROLLBACKS:              0
  LOG_FORCES:                  127
  DEADLOCKS:                   0
  LOCK_TIMEOUTS:               0
  LOCK_GRANTS_IMMED:           134549
  LOCK_GRANTS_WAIT:            0
  SYS19:                       0
  CMD_PREPARES:                396
  CMD_REPREPARES:              0
  CMD_TEMP_INDEXES:            0
  LAST_LOG_FILE:               0
  REPHOLD_LOG_FILE:            -1
  REPHOLD_LOG_OFF:             -1
  REP_XACT_COUNT:              0
  REP_CONFLICT_COUNT:          0
  REP_PEER_CONNECTIONS:        0
  REP_PEER_RETRIES:            0
  FIRST_LOG_FILE:              0
  LOG_BYTES_TO_LOG_BUFFER:     18984608
  LOG_FS_READS:                2
  LOG_FS_WRITES:               229
  LOG_BUFFER_WAITS:            0
  CHECKPOINT_BYTES_WRITTEN:    40875376
```

```
CURSOR_OPENS:                1326
CURSOR_CLOSES:               1326
SYS3:                        0
SYS4:                        0
SYS5:                        0
SYS6:                        0
CHECKPOINT_BLOCKS_WRITTEN:   2182
CHECKPOINT_WRITES:           535
REQUIRED_RECOVERY:           0
SYS11:                       0
SYS12:                       1
TYPE_MODE:                   0
SYS13:                       0
SYS14:                       0
SYS15:                       0
SYS16:                       0
SYS17:                       0
SYS9:
```

remark

Syntax

```
remark msg
```

Description

Allows to embed comments in SQL scripts.

When `rem` or `remark` is the first word on the line, `ttIsql` reads the line and ignores it.

Example

```
Command> remark This is a variable.
```

retryconnect

Syntax

```
retryconnect [0|1]
```

Description

Disables (`0`) or enables (`1`) the wait for connection retry feature.

If the connection retry feature is enabled then connection attempts to a data source that initially fail due to a temporary situation are retried until the connection attempt succeeds. For example, if data source recovery is in progress when attempting to connect, the connection retry feature causes the connect command to continue to attempt a connection until the recovery process is complete.

If the optional argument is omitted then the connection retry feature is enabled by default.

Examples

```
Command> retryconnect;
Command> retryconnect 0;
```

rpad

Syntax

```
rpad varname desiredlength paddingstring
```

Description

The RPAD command acts like the SQL function RPAD( )with some limitations:

- The desired length is in bytes, not characters.
- The padding string is not expanded for string literal escapes, such as unicode escapes.
- The padding string can contain partial unicode characters or full unicode characters and it may split the padding string in the middle of a multibyte character or surrogate pair.

Only variables that are character based (CHAR, VARCHAR) can be padded with the RPAD command.

Examples

```
Command> var myvar varchar2(200) := 'Hi There';
Command> rpad myvar 50 'Timesten';
Command> print myvar;
MYVAR : Hi ThereTimestenTimestenTimestenTimestenTimestenTi
```

run

Syntax

```
run filename [arguments]|
start filename [arguments...]|
@@ filename [arguments...]|
@ filename [arguments...]
```

Description

Reads and runs SQL commands from *filename*. The run command can be nested up to five levels.

The @@ command is identical to the @ command only if the file is specified with an absolute path.

When you specify @ with a relative path, the path is relative to the startup directory of ttIsql. When you specify @@, the path is relative to the currently running input file. Therefore @@ is useful when used in a script that must call other scripts. It does not matter what directory the invoker of ttIsql is in when the script is run.

Examples

```
Command> run hello.sql;
Hi, I am there.
```

savehistory

Syntax

```
savehistory [-all [-h] [-a | -f] outputfile
```

Description

Writes the history buffer to the specified *outputfile.*

Consecutive duplicate commands are omitted.

Use the `-all` option to include the consecutive duplicate commands.

Use the `-h` option to omit the command numbers.

Use `-a` to append to an existing output file. Use `-f` to force the overwriting of an existing output file.

For details, see clearhistory and history.

Examples

```
Command> savehistory myhistory;
Command> savehistory -a myhistory;
```

setjoinorder

Syntax

```
setjoinorder tblNames [...]
```

Description

Specifies the join order for the optimizer. `AutoCommit` must be off. `unsetjoinorder` clears the join order advice to the optimizer.

The join order is specified as a list of tables and table aliases. Use it to influence the query plan.

Examples

```
Command> Select * from TBL1 T1, TBL2 T2, TBL3 T3 where …..;
Command> Setjoinorder T2 T1 T3;
```

`Showjoinorder` reveals the most recent set value. See showjoinorder.

setuseindex

Syntax

```
setuseindex index_name,correlation_name, {0 | 1} [;...]
```

Description

Sets the index hint for the query optimizer. `unsetuseindex` clears the index hint for the query optimizer.

Examples

```
Command> setuseindex _TMPTTREE, T1, 1;
```

setvariable

Syntax

```
setvariable variable_name := value
```

Description

Sets the value of a scalar bind variable or an element of an array bind variable.

For more information, see Declaring and Setting Bind Variables.

Examples

```
Command> setvariable myvar := 'TimesTen';
```

showjoinorder

Syntax

```
showjoinorder {0 | 1}
```

Description

Enables or disables the storing of join orders. It is enabled by default.

`0` - Disables the storing of join orders

`1` - Enables the storing of join orders.

Call the `ttoptshowjoinorder` built-in procedure explicitly to display the join order after `SELECT`, `UPDATE`, `DELETE` or `MERGE` SQL statements.

The `showjoinorder` command must be executed at the beginning of every transaction that you want to see join order information for. As soon as a commit or rollback occurs `showjoinorder` is automatically disabled.

Examples

```
Command> showjoinorder;
```

sleep

Syntax

```
sleep [n] [ms]
```

Description

Suspends operation for *n* seconds or *n* milliseconds, if the unit `ms` is included. If *n* is not specified, then operation is suspended for 1 second.

Examples

```
Command> sleep;
Command> sleep 60;
Command> sleep 500 ms;
```

sqlcolumns

Syntax

```
sqlcolumns [owner_name_pattern.]table_name_pattern
```

Description

Prints results of an ODBC call to `SQLColumns`. If the optional argument is omitted then all columns in the data source are reported.

Examples

```
Command> create table t ( c int, b int);
Command> sqlcolumns T;
< <NULL>, USER, T, C, 3, NUMBER, 38, 39, 0, 10, 1, , <NULL>, 3, <NULL>, <NULL>, 1, YES,
1, <NULL>, 0, 0, 00 >
< <NULL>, USER, T, B, 3, NUMBER, 38, 39, 0, 10, 1, , <NULL>, 3, <NULL>, <NULL>, 2, YES,
1, <NULL>, 0, 0, 00 >
2 rows found.
```

sqlgetinfo

Syntax

```
sqlgetinfo <infotype>
```

Description

Prints results of an ODBC call to `SQLGetInfo`. An `info_type` argument is required for this command.

Examples

```
Command> sqlgetinfo SQL_DRIVER_VER;
SQL_DRIVER_VER = 22.01.0001.0028 Oracle TimesTen IMDB version 22.1.1.28.0

Command> sqlgetinfo SQL_DRIVER_ODBC_VER;
SQL_DRIVER_ODBC_VER = 02.50

Command> sqlgetinfo SQL_ODBC_API_CONFORMANCE;
SQL_ODBC_API_CONFORMANCE = 1 (SQL_OAC_LEVEL1)
```

sqlstatistics

Syntax

```
sqlstatistics [[owner_name_pattern.]table_name_pattern]
```

Description

Prints results of an ODBC call to `SQLStatistics`.

Example

```
Command> sqlstatistics dual;

  TABLE_QUALIFIER:    <NULL>
  TABLE_OWNER:        SYS
  TABLE_NAME:         DUAL
  NON_UNIQUE:         <NULL>
  INDEX_QUALIFIER:    <NULL>
  INDEX_NAME:         <NULL>
  TYPE:               0
  SEQ_IN_INDEX:       <NULL>
  COLUMN_NAME:        <NULL>
  COLLATION:          <NULL>
  CARDINALITY:        1
  PAGES:              <NULL>
  FILTER_CONDITION:   <NULL>
  TT_INDEX_USAGE:     <NULL>

1 row found.
```

sqltables

Syntax

```
sqltables[[owner_name_pattern.]table_name_pattern]
```

Description

Prints results of a call to `SQLTables`. The pattern is a string containing an underscore (_) to match any single character or a percent sign (%) to match zero or more characters.

Examples

```
Command> sqltables;
< <NULL>, SYS, ACCESS$, SYSTEM TABLE,  >
< <NULL>, SYS, ARGUMENT$, SYSTEM TABLE,  >
< <NULL>, SYS, CACHE_GROUP, SYSTEM TABLE,  >
< <NULL>, SYS, CACHE_GROUP_INTERVAL_STATS, SYSTEM TABLE,  >
< <NULL>, SYS, CACHE_GROUP_STATS, SYSTEM TABLE,  >
< <NULL>, SYS, COLUMNS, SYSTEM TABLE,  >
< <NULL>, SYS, COLUMN_HISTORY, SYSTEM TABLE,  >
< <NULL>, SYS, COL_STATS, SYSTEM TABLE,  >
< <NULL>, SYS, DEADLOCKCYCLES, SYSTEM TABLE,  >
< <NULL>, SYS, DEADLOCKS, SYSTEM TABLE,  >
< <NULL>, SYS, DEPENDENCY$, SYSTEM TABLE,  >
< <NULL>, SYS, DIR$, SYSTEM TABLE,  >
….….….….….….….….….….….….
331 rows found.

Command> sqltables SCOTT.T1;
0 rows found.

Command> sqltables SYS.DUAL;
< <NULL>, SYS, DUAL, SYSTEM TABLE,  >
1 row found.
```

statsclear

Syntax

```
statsclear [[<owner_name>.]<table_name>]
```

Description

Clears statistics for the specified table (or all tables if no table is specified).

Examples

```
Command> statsclear;
Command> statsclear SCOTT.T1;
```

statsestimate

Syntax

```
statsestimate [[owner_name.]table_name] {n rows | p percent}
```

Description

Estimates statistics for specified table (or all tables if no table is specified).

If you estimate statistics with an empty table list, statistics on system tables are updated also, if you have privileges to update the system tables.

Examples

```
Command> statsestimate SCOTT.T1 1000 ROWS;
```

statsupdate

Syntax

```
statsupdate [[owner_name_pattern.] table_name_pattern]
```

Description

Updates statistics for the specified table (or all tables if no table is specified).

If `tblName` is an empty string, statistics are estimated for all the current user's tables in the database.

Examples

```
Command> statsupdate;
```

tablesize

Syntax

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

Description

For each table that matches the pattern, lists the contents of the `ALL_TAB_SIZES` view.

Call the TimesTen built-in `ttComputeTabSizes(<owner>.<tblname>)` to update table sizing information in `SYS.ALL_TAB_SIZES`.

Command Aliases: `tablesizes`, `tblsizes`

Example

```
Command> tblsize SYS.TAB%;
```

use

Syntax

```
use [conn_id]
```

Description

Displays the list of current connections and their IDs. If connid is specified, switches to the given connection ID.

To use the name of the first connection, you can specify `con0` for the *`conn_id`*, rather than specifying the full original connection name. You cannot explicitly name a connection `con0`. If the first connection is disconnected, `con0` refers to the connection `none`.

If `use` fails to locate the connection id, the current connection is set to the reserved connection named "none."

See the connect command.

Examples

```
Command> use;
* Connection sampledb: DSN=sampledb
Command> use con0;
sampledb: Command>
```

variable

Syntax

```
variable [variable_name [data_type] [:= value]]
```

The syntax for binding multiple values to an array using the variable command is as follows:

```
variable array_name
'[' array_size ']'
 data_type(n):=
'[' value1, ... valuex ']'
```

Description

Declares a bind variable that can be referenced in a statement or displays the definition of the variable if the type is missing. Type can be one of the following: `(n)`, `NUMBER`, `CHAR(n)`,

NCHAR(n), VARCHAR2(n), NVARCHAR2(n), BLOB, CLOB, NCLOB, or REFCURSOR. If only (n) is supplied, it is assumed to be VARCHAR2 (n).

Assigns a value to a single variable or multiple values if the data type is an array. You can assign a value later with the setvariable command.

For more information, see Declaring and Setting Bind Variables.

Examples

```
Command> variable a varchar2(30);
Command> var arr[5] number := [ 1,2, 3 ];
```

version

Syntax

```
version
```

Description

Reports the TimesTen version information.

Examples

```
Command> version;
TimesTen Release 22.1.1.18.0
```

waitfor

Syntax

```
waitfor expected_result timeoutseconds sqlstatement
```

Description

Runs the given statement once a second until the query returns the expected result or a timeout occurs. The query must have only one column and must return exactly one row. Any errors in the query terminate the loop.

> **Note**
>
> At verbosity level 4, you can retrieve the result values that it is using to compare against in order to diagnose why the matching of the value is not successful.

Examples

```
Command> waitfor X 10 select * from dual;

Command> verbosity 4;
The command succeeded.

Command> waitfor X 10 select * from dual;
Value is 'X'
The command succeeded.
```

waitforresult

Syntax

```
waitforresult expected_result timeoutseconds searchrow searchcol sqlstatement
```

Description

Similar to the `waitfor` command, except that the result can have one or more columns. Also, the result can return 0 rows.

Runs the given statement once a second until the query returns the expected result or a timeout occurs. The *`searchrow`* and *`searchcol`* arguments indicate the ordinal position (`1..N`) of which row or column should be considered. Use `'*'` in *`searchrow`* or *`searchcol`* to indicate any row or column of the result set could have the expected value. See the waitfor command.

Examples

```
Command> waitforresult 1 5 * * select * from dual;
Waitforresult: Time expired.
The command failed.
```

At verbosity level 4, you can retrieve the result values that it is using to compare against in order to diagnose why the matching of the value is not successful.

```
Command> verbosity 4;
The command succeeded.

Command> waitforresult 1 5 * * select * from dual;
Value is 'X'
Value is 'X'
Value is 'X'
Value is 'X'
Value is 'X'
Waitforresult: Time expired.
The command failed.
```

whenever sqlerror

Syntax

```
WHENEVER SQLERROR { ExitClause | ContinueClause | SUPPRESS |
     SLEEP Number | ExecuteClause }
```

When you specify `EXIT`, always exit `ttIsql` if an error occurs. *`ExitClause`* is as follows:

```
EXIT [ SUCCESS | FAILURE | WARNING | Number | :BindVariable ]
 [ COMMIT | COMMIT ALL | ROLLBACK ]
```

When you specify `CONTINUE`, `ttIsql` continues to the next command, even if an error occurs. *`ContinueClause`* is as follows:

```
CONTINUE [ COMMIT | COMMIT ALL | ROLLBACK | NONE ]
```

Run specified commands before continuing. *`ExecuteClause`* is as follows:

```
EXECUTE "Cmd1;Cmd2;...;"
```

The WHENEVER SQLERROR command options are as follows:

- EXIT: Always exit ttIsql if an error occurs. Specify what is performed before ttIsql exits with one of the following. SUCCESS is the default option for EXIT.
  - SUCCESS or FAILURE or WARNING: Return SUCCESS (value 0), FAILURE (value 1), or WARNING (value 2) to the operating system after ttIsql exits for any SQL error.
  - *Number*: Specify a number from 0 to 255 that is returned to the operating system as a return code. Once ttIsql exits, you can retrieve the error return code with the appropriate operating system commands. For example, use echo $status in the C shell (csh) or echo $? in the Bourne shell (sh) to display the return code.

    The return code can be retrieved and processed within batch command files to programmatically detect and respond to unexpected events.
  - *:BindVariable*: Returns the value in a bind variable that was previously created in ttIsql with the variable command. The value of the variable at the time of the error is returned to the operating system in the same manner as the *Number* option.

    > **Note**
    >
    > The bind variable used within the WHENEVER SQLERROR command cannot be defined as a LOB, REFCURSOR, or any array data type.

  In addition, you can specify whether to commit or rollback all changes before exiting ttIsql.
  - COMMIT: Runs a COMMIT and saves changes only in the current connection before exiting. The other connections exit with the normal disconnect processing, which rolls back any uncommitted changes.
  - COMMIT ALL: Runs a COMMIT and saves changes in all connections before exiting.
  - ROLLBACK: Before exiting, runs a ROLLBACK and abandons changes in the current connection and, by default, in all other connections. The other connections exit with the normal disconnect processing, which automatically rolls back any uncommitted changes.
- CONTINUE: Do not exit if an error occurs. The SQL error is displayed, but the error does not cause ttIsql to exit. The following options enable you to specify what is done before continuing to the next ttIsql command:
  - NONE: This is the default. Take no action before continuing.
  - COMMIT: Runs a COMMIT and saves changes in the current connection before continuing.
  - COMMIT ALL: Runs a COMMIT and saves changes in all connections before continuing.
  - ROLLBACK: Before continuing, runs a ROLLBACK and abandons changes in the current connection and, by default, in all other connections. The other connections exit with the normal disconnect processing, which automatically rolls back any uncommitted changes.
- SUPPRESS: Do not show any error messages and continue.
- SLEEP: Sleep for a specified number of seconds before continuing.

- EXECUTE: Run specified commands before continuing. Each command is separated from the other commands by a semicolon (;). If any command triggers additional errors, those errors may cause additional actions that could potentially result in a looping condition.

Description

Provide direction on how to handle errors when in ttIsql. Run the WHENEVER SQLERROR command to prescribe what to do when a SQL error occurs. WHENEVER SQLERROR can be used to set up a recovery action for SQL statements, SQL script, or PL/SQL block.

By default, if a SQL error occurs while in ttIsql, the error information is displayed and ttIsql continues so that you can enter a new command. The default setting is WHENEVER SQLERROR CONTINUE NONE. You can also specify that ttIsql exits each time an error occurs, which may not be the best action for interactive use or when running a SQL script or a PL/SQL block.

Examples

The following example uses EXIT to return an error code of 255 and runs a COMMIT statement to save all changes to the current connection before exiting ttIsql. The example retrieves the error code using the C shell echo $status command.

```
Command> WHENEVER SQLERROR EXIT 255 COMMIT;
Command> SELECT emp_id FROM employee;
 2206: Table PAT.EMPLOYEE not found
WHENEVER SQLERROR exiting.
% echo $status
255
```

The following example demonstrates how the WHENEVER SQLERROR command can run ttIsql commands or TimesTen utilities when an error occurs, even if the error is from another TimesTen utility:

```
Command> WHENEVER SQLERROR EXEC "DSSIZE;CALL TTSQLCMDCACHEINFOGET();";
Command> call TTCACHEPOLICYGET;
 5010: No OracleNetServiceName specified in DSN
The command failed.

DSSIZE;

  PERM_ALLOCATED_SIZE:      32768
  PERM_IN_USE_SIZE:         9204
  PERM_IN_USE_HIGH_WATER:   9204
  TEMP_ALLOCATED_SIZE:      40960
  TEMP_IN_USE_SIZE:         7785
  TEMP_IN_USE_HIGH_WATER:   7848

CALL TTSQLCMDCACHEINFOGET();

CMDCOUNT, FREEABLECOUNT, SIZE
< 10, 7, 41800 >
1 row found.
```

The following demonstrates the SUPPRESS command option. It suppresses all error messages and continues to the next command. The example shows that the error messages can be turned back on in the existing connection with another command option, which in this case is the EXIT command.

```
Command> WHENEVER SQLERROR SUPPRESS;
Command> SELECT *;
Command> WHENEVER SQLERROR EXIT;
Command> SELECT *;
```

```
 1001: Syntax error in SQL statement before or at: "", character position: 9
select *
        ^
WHENEVER SQLERROR exiting.
```

The following example sets a bind variable called `retcode`, the value of which is returned when a SQL error occurs:

```
Command> VARIABLE retcode NUMBER := 111;
Command> WHENEVER SQLERROR EXIT :retcode;
Command> INSERT INTO EMPLOYEES VALUES (
202, 'Pat', 'Fay', 'PFAY', '603.123.6666',
TO_DATE ('17-AUG-1997', 'DD-MON-YYYY'),
'MK_REP', 6000, NULL, 201, 20);
  907: Unique constraint (EMPLOYEES on PAT.EMPLOYEES) violated at Rowid
 <BMUFVUAAACOAAAAIiB>
WHENEVER SQLERROR exiting.
% echo $status;
111
```

xlabookmarkdelete

Syntax

```
xlabookmarkdelete id
```

Description

Deletes a persistent XLA bookmark.

If a bookmark that is to be deleted is not specified then the status of all current XLA bookmarks is reported.

See ttXlaDeleteBookmark in *Oracle TimesTen In-Memory Database C Developer's Guide*.

Requires `ADMIN` privilege or object ownership.

Examples

```
Command> xlabookmarkdelete;
XLA Bookmark: mybookmark
 Read Log File:  0
 Read Offset:    268288
 Purge Log File: 0
 Purge Offset:   268288
 PID:            2004
 In Use:         No
1 bookmark found.

Command> xladeletebookmark mybookmark;

Command> xlabookmarkdelete;
0 bookmarks found.
```