

# Testnext

PERFORMANCE  
TEST SOLUTIONS  
FOR ORACLE

## **Database2test** Release 3

## **User's Guide**

---

## > Table of Contents

<b>Preface</b>	<b>3</b>
<b>Preparing to use Database2test</b>	<b>4</b>
Requirements	4
Installation	4
Running Database2test	4
Configuring Database2test	5
<b>The Testing Process</b>	<b>6</b>
<b>How Database2test Works</b>	<b>7</b>
<b>Creating a Database Connection Pool</b>	<b>9</b>
<b>Creating a Database Test Script</b>	<b>14</b>
Create a SQL database statement	15
Compound SQL database statement	16
Pause statement	16
Create a PL*SQL database statement	17
Establishing the baseline execution time	17
Defining the target load	19
<b>Parameterize a Test Script</b>	<b>20</b>
Adding script parameters	20
Defining database statement parameters	22
Parameter types	22
Parameter Properties	22
<b>Running a Performance Test</b>	<b>25</b>
Defining the test scenario	25
Defining the overall test goal	26
<b>Analyzing The Test Results</b>	<b>31</b>
<b>What It Means</b>	<b>33</b>

---

## > Preface

Welcome to the TestNext Database2test User's Guide. This guide provides a step-by-step overview to using TestNext Database2test.

Database2test is a performance test tool dedicated to the Oracle Database Server. It can be used to simulate a (heavy) load on a single or multiple database instances to test its strength or to analyze overall performance under different load types. You can use it to make a graphical analysis of performance or to test your database server behavior under heavy concurrent load.

With Database2test you can define and run a performance test in a few steps using the powerful wizards.

During the performance test a lot of metrics, such as executions times and database connections are captured and logged.

At the end of a test the test results are presented in various automatically generated graphical reports; the starting point for analysis.

---

## ➤ Preparing to use Database2test

### Requirements

Database2test requires your computing environment meets some minimum requirements. Database2test should be installed on a reasonably modern Windows desktop with at least 1.2GHz processor and 512Mb of RAM (memory).

Database2test is a 100% Java application and requires a fully compliant JVM 1.6 or higher. We recommend you to download the distribution of Database2test including the Java 6 runtime.

Database2test has been tested and should run correctly under Windows (NT, 2000, XP, Vista and Windows 7).

Database2test contains some sample scripts and these sample scripts are a good starting point for getting familiar with Database2test.

### Certified Oracle Database Servers

Database2test is build on top of the *Oracle 11g JDBC Drivers* and therefore certified with the following database releases:

- Oracle Server 11g,
- Oracle Server 10.2.0.2 and higher, and
- Oracle Server 9.2.0.4 and higher.

### Installation

Installing Database2test is a snap. Database2test is available as a Windows installer. Launch the installer and install the full functional version into the directory where you want Database2test to be installed.

Please note that installing Database2test *will overwrite* your existing installation. However, scripts, settings and registrations won't be lost.

### Running Database2test

After the installation has completed the installer has created start menu items to launch (or uninstall) Database2test.

---

**Note:** *If the start-up fails, ensure that the JAVA\_HOME variable points to the location where you have installed Java 6 or higher. This variable is set in the <database2test home>/bin/setenv.bat OS script.*

---

## Configuring Database2test

If Database2test is started for the first time you are automatically directed to the Setup section (see 1).

The only required configuration setting for Database2test is at least one valid database connection pool (see 2).

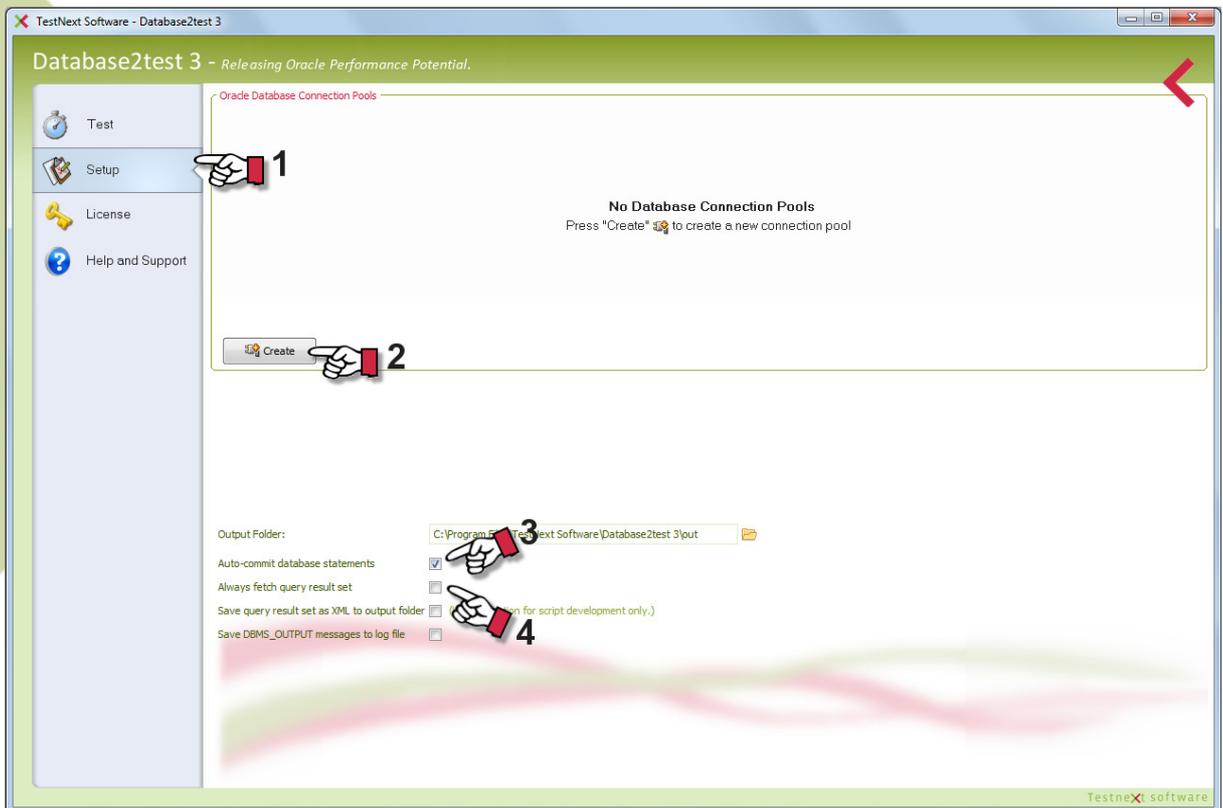
Optionally you can save the output to a local output folder. This option is especially interesting for testing scripts.

By default all database statements are committed immediately (auto committed). However, when you create a compound SQL statement, i.e. a statement containing multiple sub statements, you can execute this statement as a single transaction by disabling the auto commit flag (see 3).

You can instruct Database2test to always fetch the complete result set (see 4)

Sometimes it can be handy to save the query result set to a XML file. We strongly encourage you to disable this option during the 'real' test.

The same applies to the option to enable the DBMS\_OUTPUT messages and save the database messages to the log file.



---

## ➤ The Testing Process

Typically a performance test is a two step process and consists of the following tests:

- Load test, and
- Stress test (*optional*)

The difference between a load and stress test is the objective of the test. For a stress test you are interested in the maximum load the Oracle Database server can handle.

For a load test you are interested in the performance and resource usage of the Oracle Database during typical production load. Database2test is suitable for both type of tests. Although the stress test is an optional test it does provide some useful input for future capacity planning.

The first step for conducting a load test is to develop one or more test scenarios or test cases. A test scenario defines the typical working conditions. A test scenario defines (1) *a set of database statements* (SQL and/or PL\*SQL) and (2) the *total number of statement executions*, i.e. the target load.

The creation process for a stress test is similar to a load test. The test scenario is often based on the same set of scripts only the number of statement executions, in other words, the target load is different.

---

## > How Database2test Works

The load on the database is generated through Database2test by executing a set of database statements concurrently; simulating multiple users working together.

For the load you can choose between two types of database statements, SQL statements and PL\*SQL statements. These database statements are captured into *test scripts*. A script holds the *definition* of the database statement.

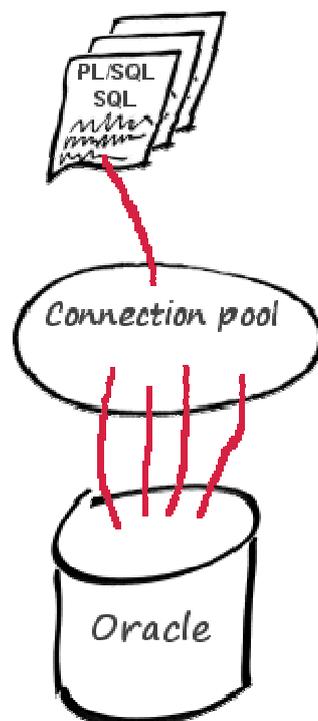
During the test one or more *instances* of the script are executed on the database server through one or more connection pools. These instances are called *statement executions*.

Each script is associated with a *connection pool*. A connection pool is a set of physical database connections.

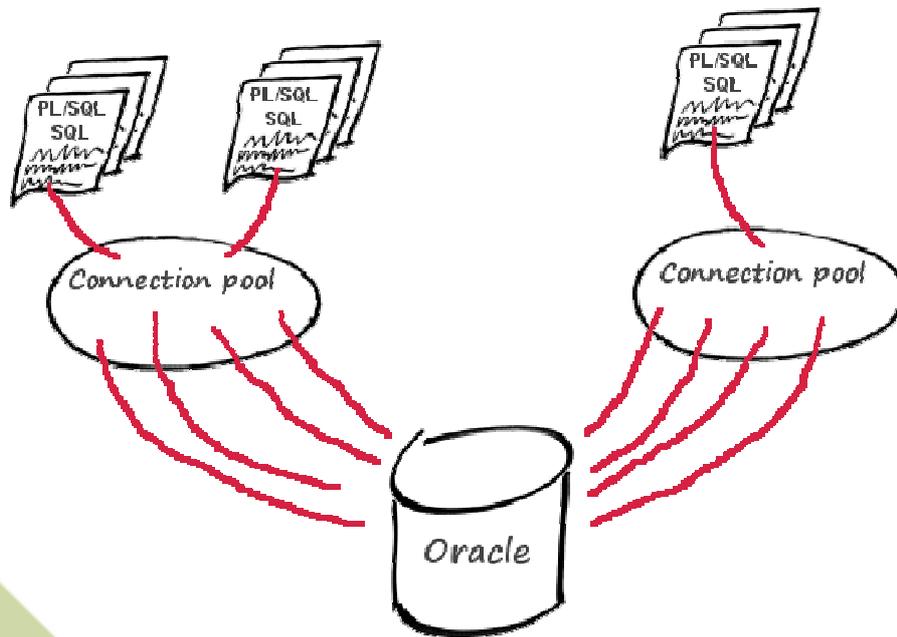
You can specify the desired pool size, that is, the number of database connections that will be established to generate the load. These database connections are created during the initialization of the load test.

When you choose to manage the connection pool by Database2test the number of database connections of the pool can vary during the test. This depends on the scheduled load.

The following figure shows a very simple test scenario involving a single script, a connection pool and the database. To execute a single SQL statement multiple times concurrently on an Oracle database.



A more realistic figure is a test scenario involving multiple test scripts using multiple connection pools.



You can even opt for multiple database instances and so called *compound SQL statements*. A compound SQL statement consist of multiple database statements. In other words, Database2test has a very flexible architecture.

The first step to develop a load test is to create one or more connection pools.

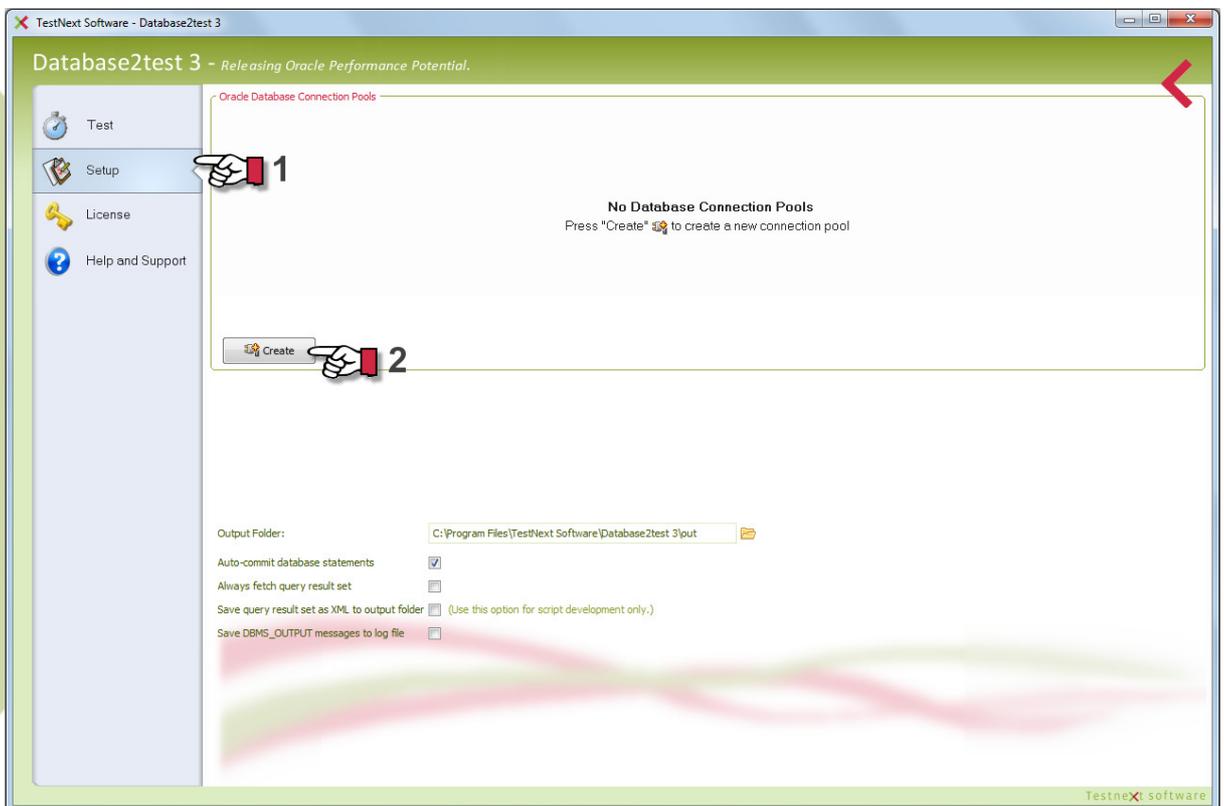
## > Creating a database Connection Pool

In this section you will learn how to create a Connection Pool. A connection pool is a cache of database connections maintained so that the connections can be reused by the database2test engine when future requests to the database are required. The connection pool is created during initialization of the performance test.

The database connections of the pool are used for each script execution. Each test script is assigned a connection pool.

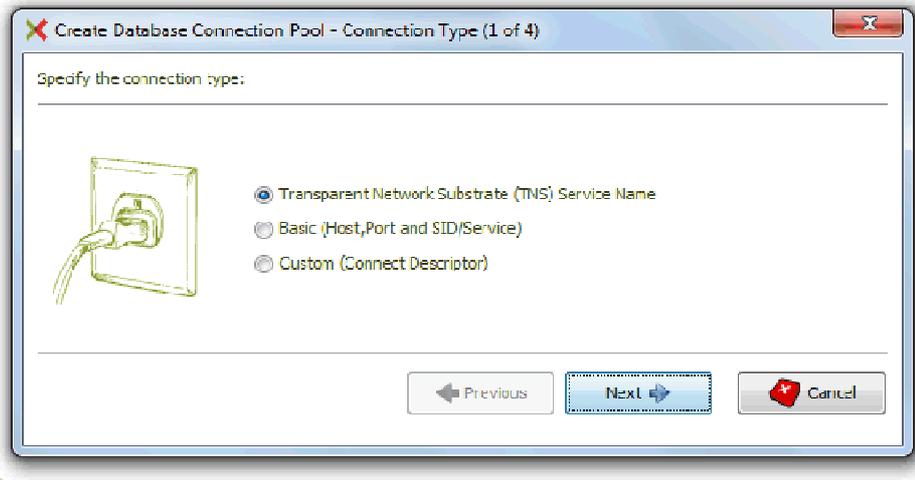
You can share a connection pool by multiple scripts but you can also choose to create multiple connection pools for multiple scripts.

To create a connection pool navigate to the Setup section (see 1). Then press the “Create” button to launch the wizard (see 2).

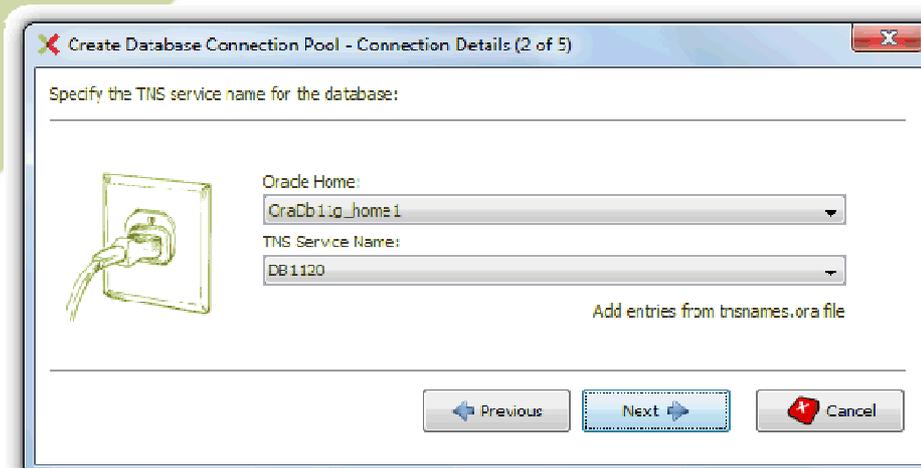


The creation process of a database connection pool is a 4-step wizard. The first step is to specify the connection type. The connection type tells Database2test how to gather the required connect descriptor for opening the database connections.

You can select a predefined TNS service name. Database2test will show you a list with all available predefined connect descriptors available on the local machine. You can select the 'Basic' connection type. Database2test will offer you a form to fill out the minimum (basic) information to establish a database connection. Alternatively, you can provide your own (custom) connect descriptor.



When you choose to select a TNS entry you can select the Oracle Home and the corresponding TNS entry. When the environment variable TNS\_ADMIN is set, also the TNS entries from that tnsnames.ora file will be selectable. Alternatively, you can add the entries from a tnsnames.ora configuration file manually.



When you choose to fill out the basic information to establish a database connection you have to provide the database host, the listener port and the Service Name or System identifier (SID):

Create Database Connection Pool - Connection Details (2 of 5)

Specify the host, port and service:

Database Host: rdbms.testnext.local Port: 1521

Service Name: db1120

System ID (SID):

Previous Next Cancel

The next step is to provide the credentials, i.e. username, password and the default role (if applicable). You are strongly encouraged to verify the credentials. When the test succeeds you know that the provided information is valid.

Create Database Connection Pool - Authentication (3 of 5)

Specify a username, password and role to establish the connection:

Username: SCOTT

Password: .....

Role: DEFAULT

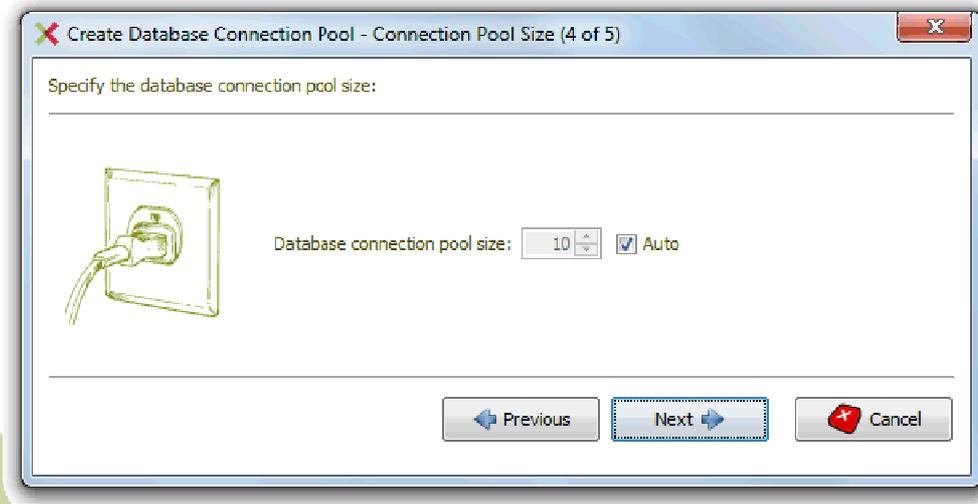
Test

Previous Next Cancel

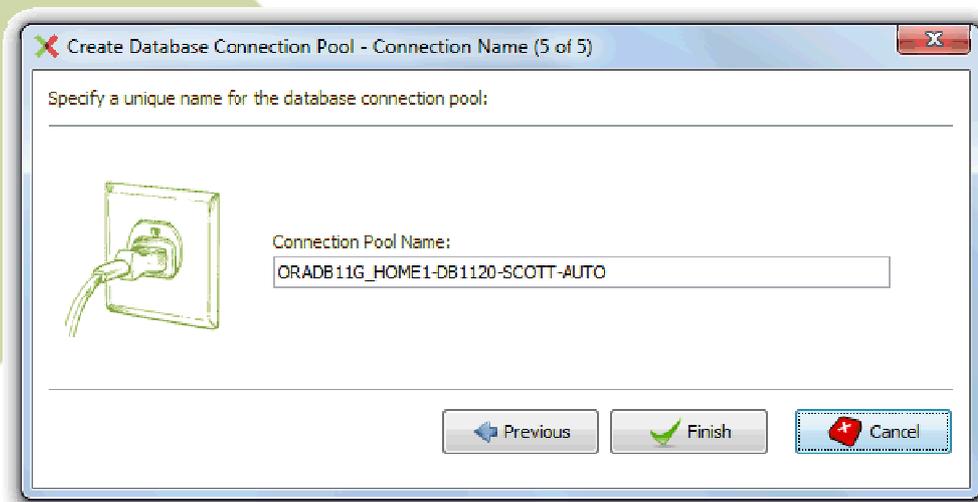
The next step is to specify the size of the connection pool. You can choose to specify a fixed size or you can let Database2test to manage the connection pool size for you. When you let Database2test to manage the connection pool size it will always try to minimize the size of the connection pool. During a load test Database2test will increment the pool size when necessary.

During the initialization of the load test Database2test will open all database connections first.

During the test Database2test will pick an inactive connection from the pool to execute a database statement and release the connection as soon as the database statement is executed and so on.

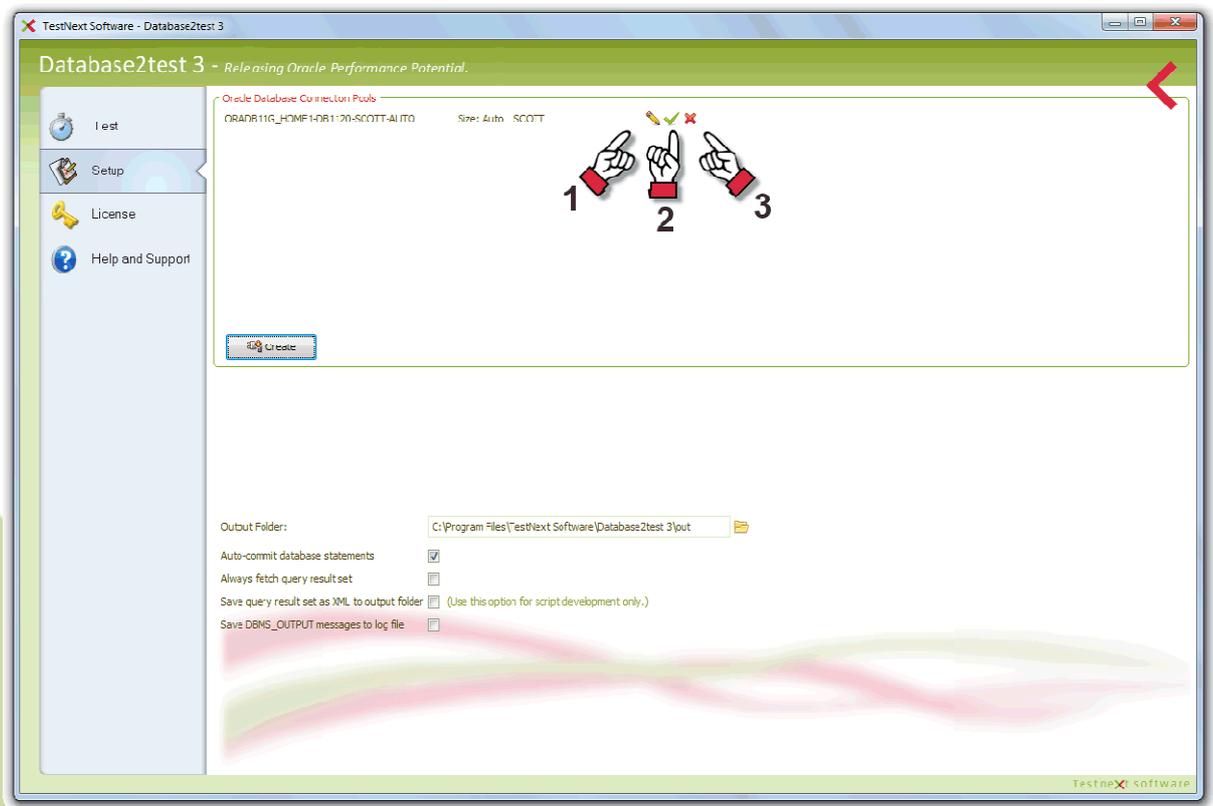


The last step is to provide a unique name for the connection pool. If possible Database2test will suggest a name based on all information provided in the previous steps.



Select 'Finish' to create the connection pool and close the wizard.

The created connection pool is now added to the available connection pools. You can always edit the connection pool (see 1), test the connection pool (see 2) or remove the connection pool (see 3).



At this point you can create another connection pool or start to create a test script.

## ➤ Creating a Database Test Script

In this section you will learn how to create a Database2test test script.

**Note:** Before you start to create a script make sure that you have created a valid database connection pool first (see [Creating a Connection Pool](#)).

Launch Database2test and select the Test section (see 1).

Database2test let you define two types of database scripts. SQL statements and/or PL\*SQL statements.

Select the preferred script type to add a new SQL statement (see 2) or a new PL\*SQL statement (see 3) to the Test Suite.

Database2test 3 - Releasing Oracle Performance Potential.

Test Database Connection Pool (default)  
ORADB11G\_HOME1 DE1120 SCOTT AUTO

Test Suite

Test Item	Script Type	Options	Icons
<input checked="" type="checkbox"/> PLSQL Example - DBMS_OUTPUT	PL*SQL Example - DBMS_OUTPUT	-- -- --	Icons
<input checked="" type="checkbox"/> SQL Example - Spatial Query	SQL Example - Spatial Query	-- -- --	Icons
<input checked="" type="checkbox"/> SQL Example Bind Variable	SQL Example: Query With Parameter	-- -- --	Icons

Test Goal  
Run the load test for 3 minutes with a maximum of 20 active (simultaneous) statement executions.

Test Monitor

Elapsed (mm:ss): 00:00  
Remaining (mm:ss): 00:00  
Active Statements: 0  
Pending Statements: 0  
Statements Passed/Failed: 0 / 0  
Database Performance: 0%  
Avg. Exec. Time Delay: 0 msec.  
Database Connections: 0  
Test status: Idle

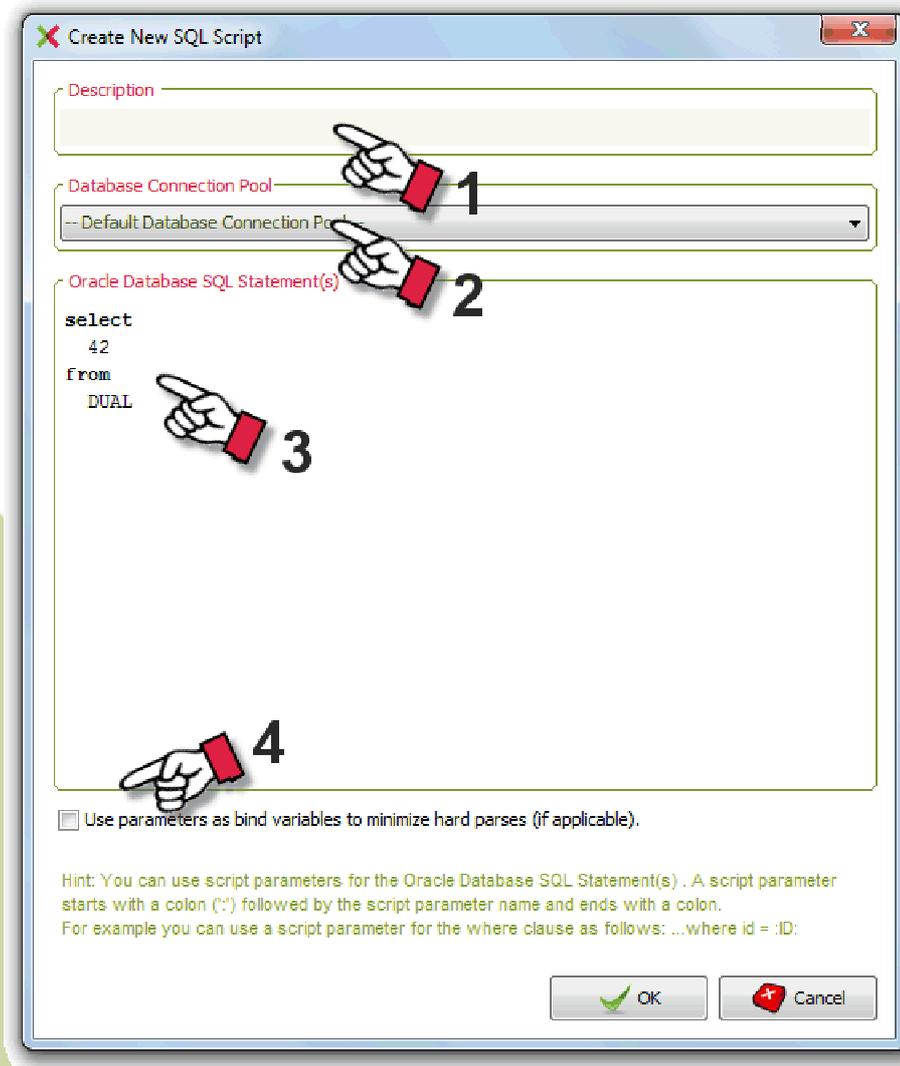
Performance (%)  
Exec. Time Delay (ms)  
Load (stmts/min)  
Active Statements

Start Press Start to run the performance test.

TestNext software

## Create a SQL database statement

When you select to create a SQL statement script the following editor pops up:



First, you may enter an appropriate description for the script (*see 1*). The description will be visible in the Test Suite.

When you omit the description the file location of the script will be shown instead.

The second step is to select the database connection pool (see 2). You can either select the default database connection pool or select a database connection pool explicitly. The latter option is mandatory when you decide to use more than one database connection pool for the test using multiple scripts (see How Database2test works).

The next step is to enter a SQL-statement (see 3). Make sure that you enter a valid SQL statement. The SQL statement should be valid for the selected database connection pool.

### **Compound SQL-statement**

You can create a simple SQL-statement or a compound SQL-statement. A simple SQL-statement is a single database statement. A compound SQL-statement is a SQL statement that consists of multiple (DML) SQL sub statements. The sub statements are separated by a semicolon.

During the test the system will execute the sub statements consecutively.

By setting the global setting *auto-commit* to false (see settings tab) the compound SQL-statement can be executed as a single transaction.

### **Pause statement**

You can also insert a special 'pause statement' to a compound SQL-statement. The syntax is **PAUSE <seconds>;**

The system will pause the execution of the compound statement for the given number of seconds.

Note that the duration of a pause statement will *not* be added up to the overall execution time of the SQL-statement.

Example pause statement:

```
insert into <table> values (<value>); --insert a record
PAUSE 10; -- wait for 10 seconds
delete from <table>; -- delete all records
```

---

**Hint:** You can use script parameters for the Oracle Database SQL Statement . A script parameter starts with a colon (':') followed by the script parameter name and ends with a colon.

For example you can use a script parameter for the where clause as follows: *select xxx from table where id = :ID;*

---

When you parameterize the where-clause of the SQL statement you have the option to use the parameter as a bind variable (see 4). This will minimize the number of hard parses during the test and may better reflect the application behavior. Consult your database administrator for details.

## Create a PL\*SQL database statement

Creating a PL\*SQL statement is equal to creating a SQL statement.

The only difference is that you cannot use a parameter as a bind variable and you do not have the option to create a compound PL\*SQL statement.

## Establishing the baseline execution time

The next step after creating a database test script is to capture the baseline execution time for the database statement in the script.

The baseline execution time is the (average) execution time for the database statement on the database server.

This is a useful and mandatory metric for establishing a point of reference. During the test the performance is determined relative to this metric.

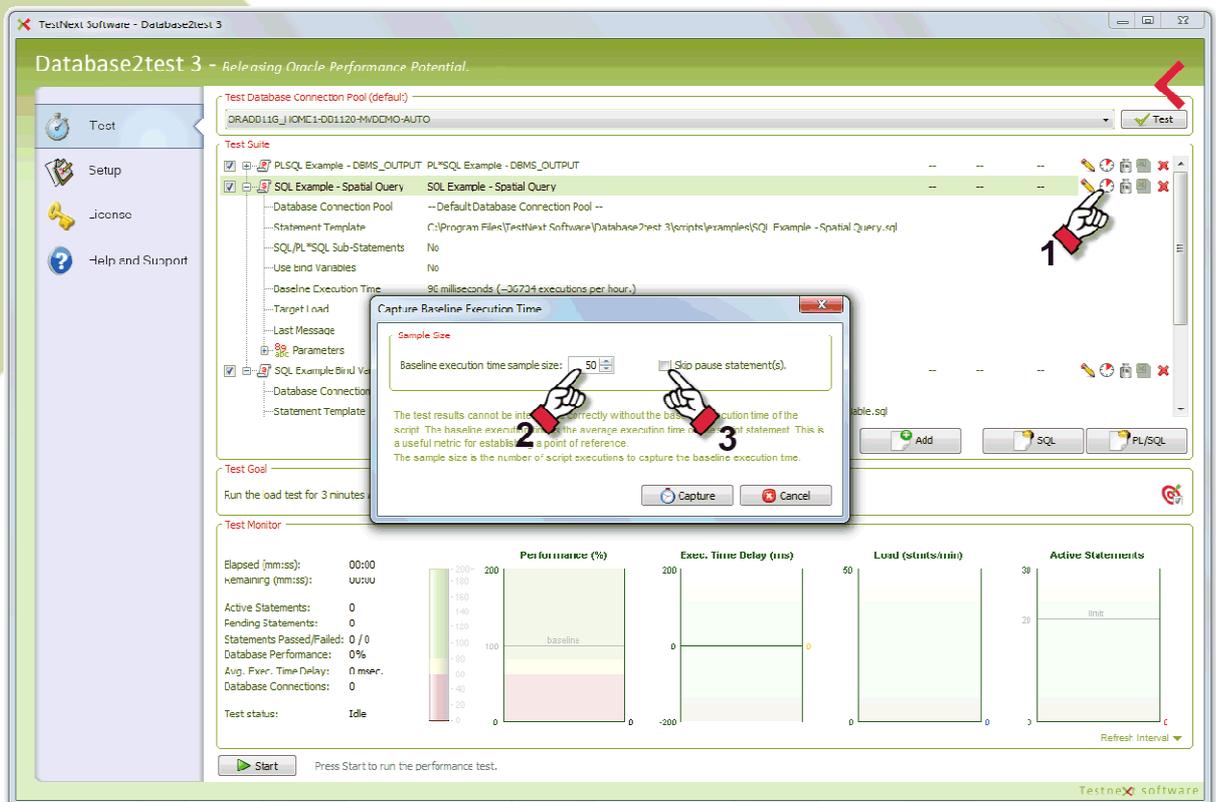
---

**Note:** The baseline execution time is a mandatory metric. You cannot run a performance test without establishing the baseline execution time first!

---

After saving a new database test script a dialog will popup automatically to capture the baseline execution time.

Alternatively, you can always (re)capture the baseline execution time by clicking the corresponding button (see 1)

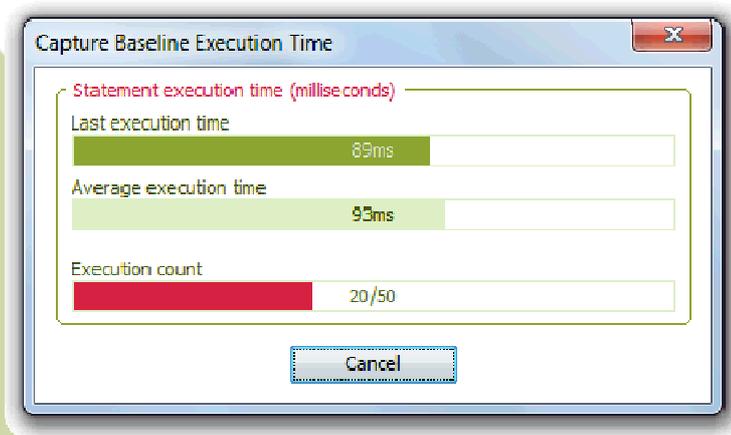


A dialog pops up with the option to set the sample size. The sample size tells Database2test how many times the statement should be executed on the Oracle Database to establish a good average execution time.

The sample size should depend on whether the database script has script parameters or not. If the statement is static, i.e. has no parameters, the sample size could be low (default 10). However, when the database script is dynamic, i.e. has script parameters, the sample size should be sufficiently high to get a representative performance baseline low (default 25).

When you have included one or more pause statements you can opt to ignore the pause statements during establishing the baseline execution time by selecting the checkbox (see 3). Note that this option is only valid for establishing the baseline execution time. During the test the pause statements will never be ignored.

When the baseline execution time is being captured the following dialog is visible showing the intermediate results.



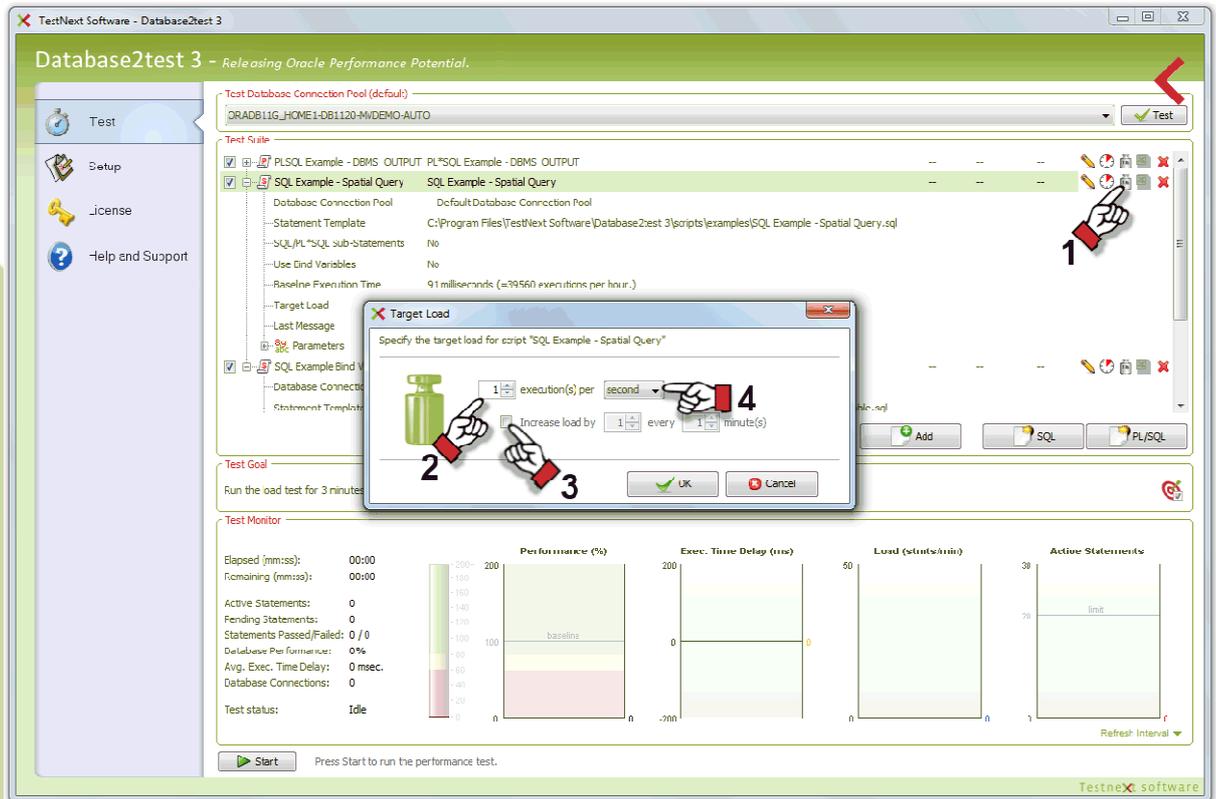
## Defining the target load

After establishing the baseline execution time you should define the target load for the script during the performance test.

To define the target load you should press the corresponding button (see 1). A dialog shows up to specify the target load for the test script.

You should specify the number of statement executions (see 2) and the corresponding time interval (see 3).

Optionally, you can dynamically increase the load during the test (see 4). The latter option is especially interesting for conducting a stress test.



Press OK to save the target load.

---

## > Parameterize a Test Script

When you create a test script, you can also parameterize the database statement by replacing literal values by parameters. At runtime the dynamic parameter values will be generated based on the corresponding parameter definition. Parameterization is very useful to conduct a realistic performance test since you bypass the caching mechanism of the database as much as possible.

---

*Tip: Before you parameterize a database statement make sure that the statement runs fine.*

---

### Adding script parameters

The first step to parameterization is to add one or more parameters to a database statement.

To add a parameter to a database statement you should replace a literal value by the parameter name enclosed by colons, i.e. **:<parameter name>**.

The script editor will automatically recognize this parameter construct.

For example, to replace the literal ID of a where-clause you should replace the SQL statement:

```
select OBJ.object_name from OBJ where OBJ.id = 2517
```

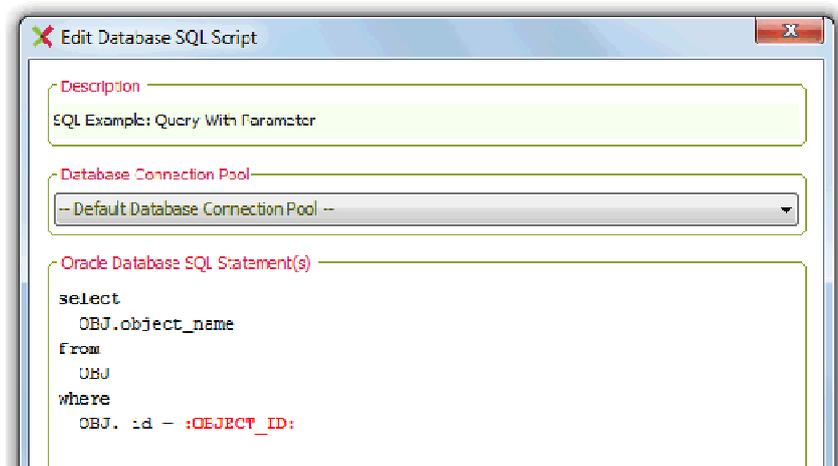
by

```
select OBJ.object_name from OBJ where OBJ. id = :OBJECT_ID:
```

---

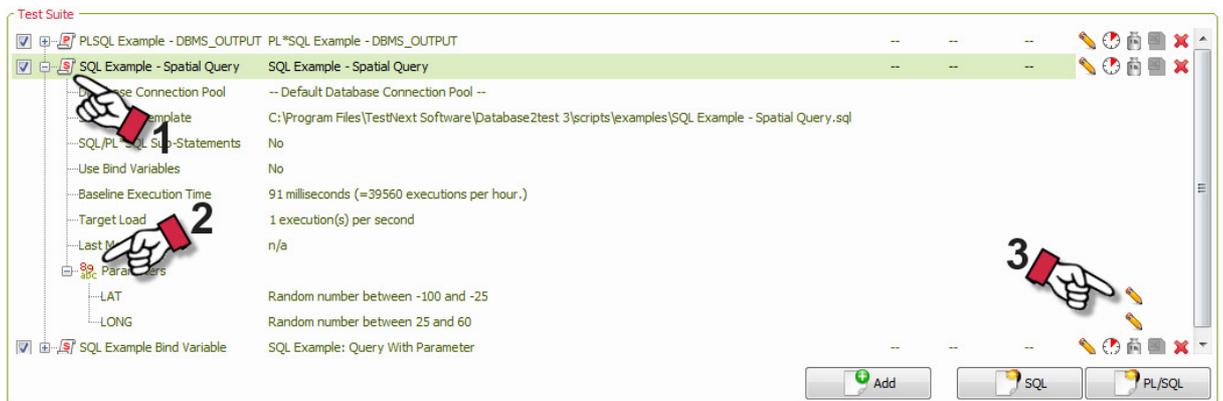
*Note: The parameter name is limited to 32 characters including underscores.*

---



When you create a script including one or more parameters the script type indication of the script icon changes from black into red (see 1).

All script parameters are visible (see 2) and you can modify the parameter definition by clicking the corresponding edit button (see 3) or by double clicking the parameter name.



---

**Tip:** The included sample scripts are a good starting point for getting familiar with parameterization.

---

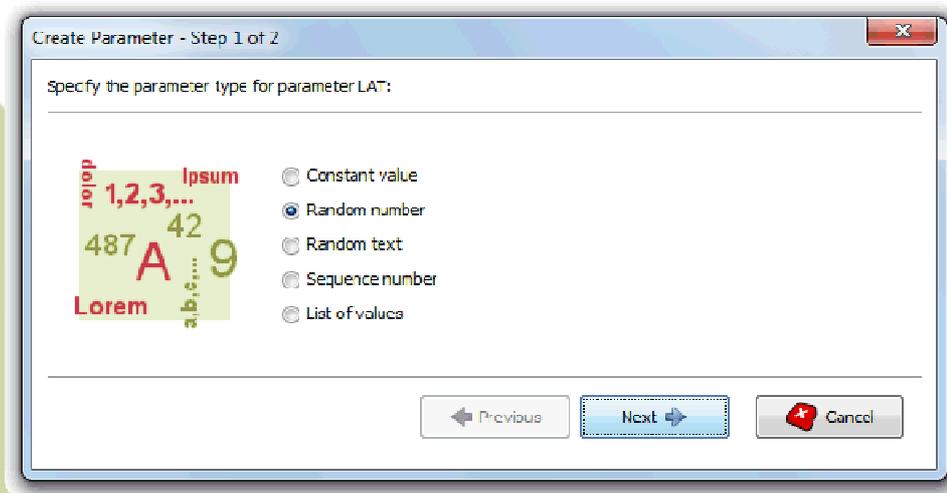
### Defining database statement parameters

After adding parameters to a database statement you have to define the parameter. For each parameter added to the database statement the parameter wizard will open automatically to help you define in two steps the parameter type and corresponding parameter properties.

### Parameter types

Each parameter should have a parameter type. You can choose between five different parameter types:

- Constant value
- Random number
- Random text
- Sequence number
- List of values (LOV)

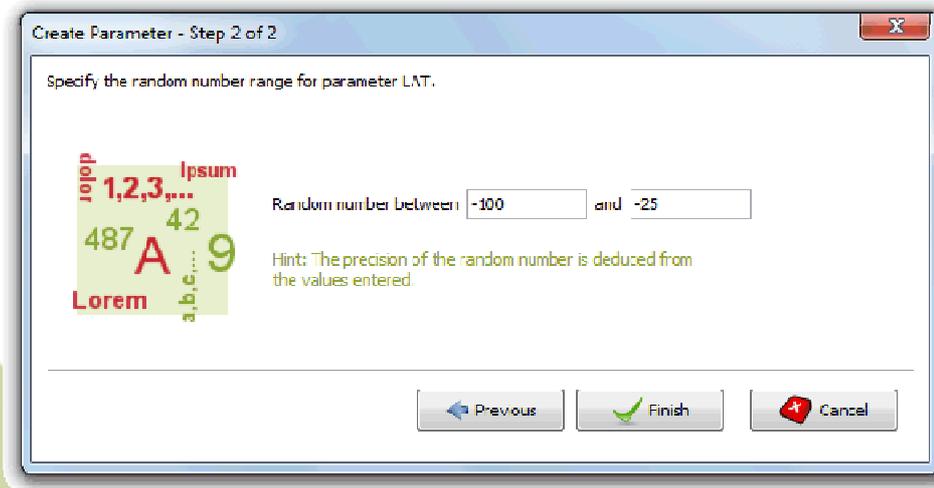


Select the preferred parameter type and press Next.

## Parameter Properties

The next step is to define the parameter properties for the selected parameter type of the previous step.

For a *random number* you should define the minimum (inclusive) and maximum (inclusive) for the random number parameter. Note that the precision (that is, digits to the right of the decimal point) of the random values during the test are deduced from the values entered.



For a random alphanumeric value (*random text*) you have to provide the minimum length and the maximum length. At runtime the parameter value will consist of at least minimum length and no more than maximum length arbitrary ASCII characters.

For a *sequence number* you have to provide the start and increment value. Then for each virtual user the parameter value will be increased by the increment value.

For a *list of values* you have to select the file containing the list of values. The list of values file is a flat ASCII file with the file extension .lov.

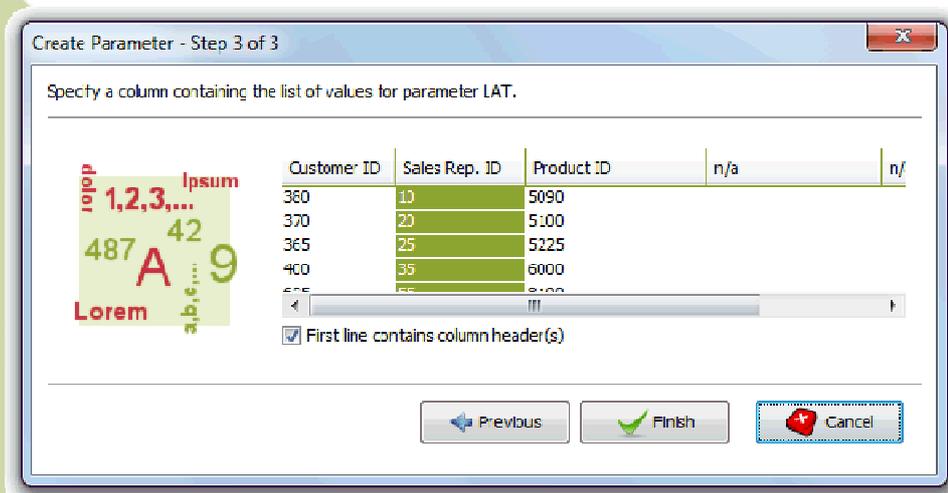
The file consists of an optional header followed by records. Each record is one line of the text file and each value of a record is separated from the next by a semicolon or a tab stop.

The values will be cyclically reused as often as necessary.

Example list of values:

```
Customer ID;Sales Rep. ID;Product ID
380;10;5090
370;20;5100
365;25;5225
400;35;6000
625;55;8100
700;42;7500
etc...
```

The next wizard step after selecting the lov file is to select the preferred column. For following screenshot the column containing the Customer ID's is selected.



---

## > Running a Performance Test

Once you have finished creating and parameterizing the test scripts, you are ready to define and run a test scenario.

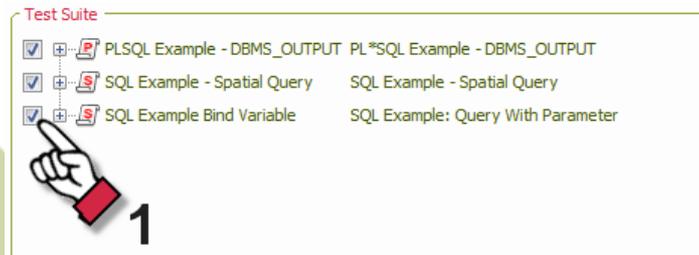
Make sure that the connection pools are valid.

### Defining the test scenario

A typical test scenario consists out of one or more scripts, i.e. database statements.

You can add a test script to the test scenario by enabling the script in the Test Suite (see 1).

Note that all parameters of the script should be defined and that the baseline execution time is captured before you can add the script to the test scenario.



---

**Warning:** Schedule a reasonable/realistic load behavior!

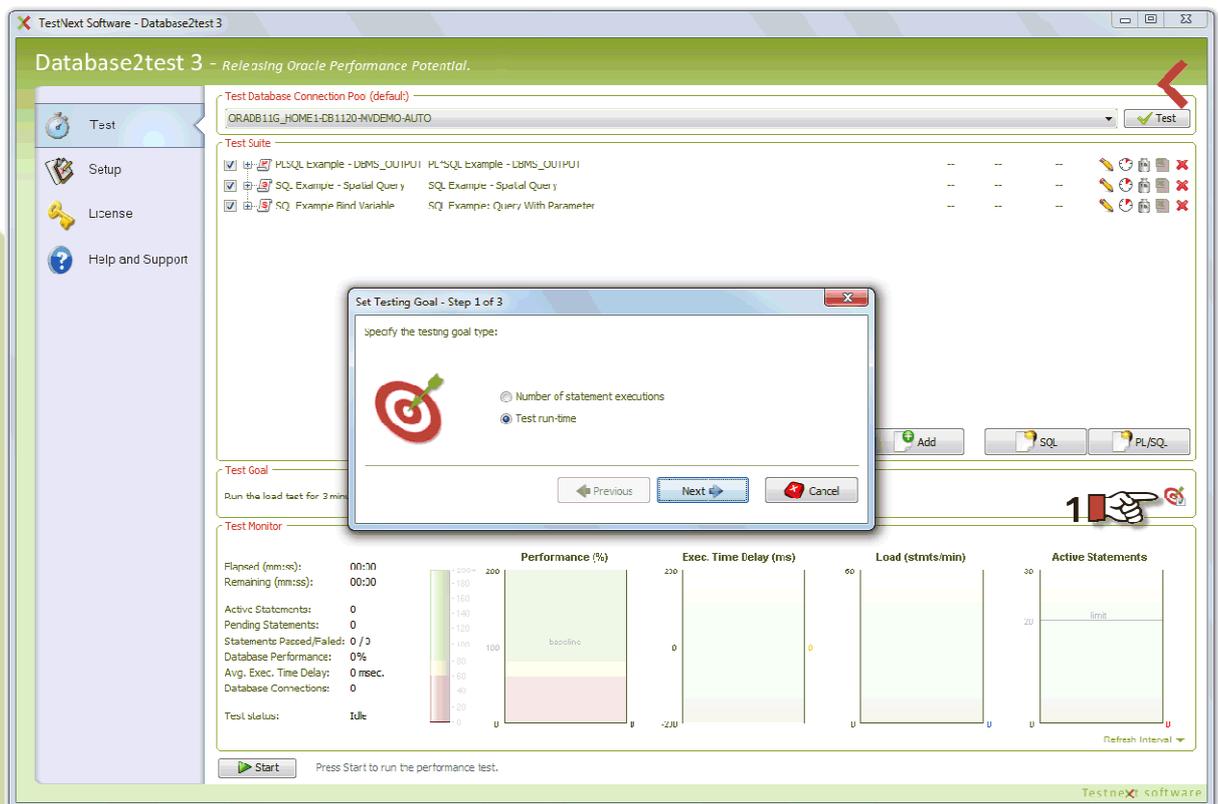
---

## Defining the overall test goal

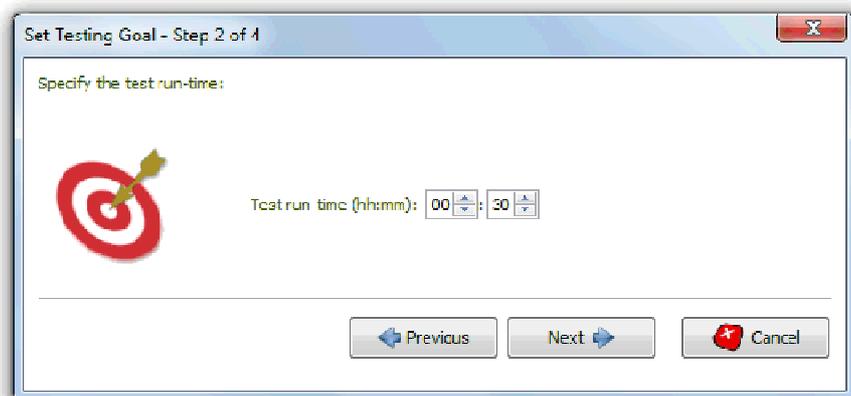
The next step is to define the test goal. Press the test goal button (see 1) in the Test Goal pane to invoke the wizard.

In three steps you can define the overall test goal.

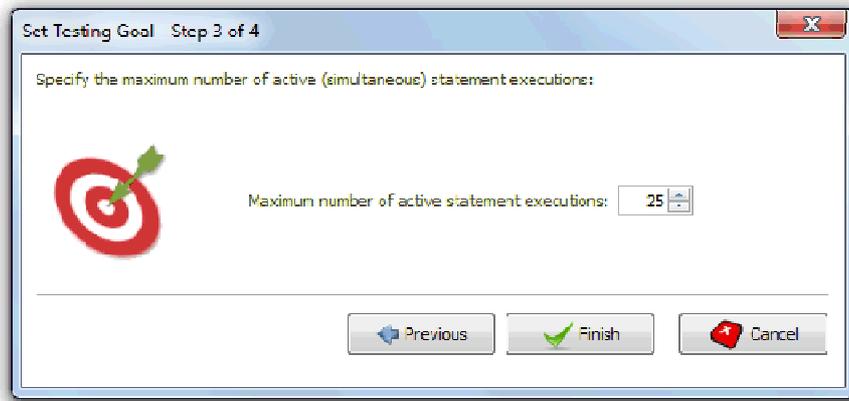
First, you have to specify the goal type. Then the overall number of database statements to execute or the test duration and finally the maximum number of active (concurrent) database statements allowed. The latter option is also dependent on the active license. Database2test has support for two test goal types. A test based on the *test runtime* and a test on the *number of statement executions*. Select the preferred test goal and press Next.



Note that when either test goal is reached, the system will not terminate immediately but wait for the active database statements to stop gracefully. This is the ramp-down phase of the test.



The last step is to define the maximum number of active statements allowed during the test. Setting this value to 25 means that at any moment no more than 25 concurrent database statements will be executed on the database server irrespective of the target load at script level.



---

**Note:** Database2test allows you to purchase a license to execute as many database statements as you need to effectively test your Oracle Database. For the (full functional) evaluation version, however, you are licensed to set a maximum of 2 active database statements only.

---

Before you start the test you should familiarize yourself with the Test page. While the test is running you can see how the Oracle Database performs in real time. You can view performance information on the online performance monitors and in tabular form.

The Test section contains four panes. The *Default Database Connection Pool*, *Test Suite*, *Test Goal* and the *Test Monitor*.

The Default Database Connection Pool shows the connection pool that will be used by all scripts that are defined to use the default database connection pool.

The Test Suite pane lets you define the test scenario and view the run status, i.e. passed statement executions (see 1), failed statement executions (see 2) and the performance for the *passed* statement executions (see 3).

The screenshot displays the TestNext Software interface for 'Database2test 3'. The interface is divided into several panes:

- Test Suite:** Shows a list of test scenarios with their status and performance metrics. Three hand icons labeled 1, 2, and 3 point to specific rows in the list.
- Test Goal:** Displays the current testing goal: 'Run the load test for 30 minutes with a maximum of 25 active (simultaneous) statement executions.'
- Test Monitor:** Shows real-time performance statistics and graphical monitors. A hand icon labeled 4 points to the Test Monitor pane.

Test Suite	Passed	Failed	Performance
PL/SQL Example - DBMS_OUTPUT	81	33%	
SQL Example - Spatial Query	78	54%	
SQL Example Bind Variable	78	33%	

**Test Monitor Statistics:**

- Elapsed (mm:ss): 01:25
- Time Remaining: 28:35
- Active Statements: 3
- Hending >statements: 0
- Statements Passed/Failed: 237 / 17
- Database Performance: 81%
- Avg. Exec. Time Delay: 67 msec.
- Database Connections: 5
- Test status: Running

**Graphical Monitors:**

- Performance (%):** Shows a bar chart with a value of 81%.
- Exec. Time Delay (ms):** Shows a line graph with a value of 73ms.
- Load (stmts/min):** Shows a line graph with a value of 180.
- Active Statements:** Shows a line graph with a value of 3.

The Test Goal pane shows the current testing goal.

The Test Monitor (see 4) shows four online (graphical) overall performance monitors together with the tabular overall performance statistics for the running test scenario.

The following performance monitors are displayed during the test:

**Performance (%)** - shows the performance, i.e. the relation between the baseline execution time and actual execution time (see *What it means*).

**Load/Statements Per Minute** – shows the actual and target number of database statements per minute.

**Execution Time Delay (ms)** – shows the difference between the baseline execution time and the actual execution times for the last 5 seconds.

**Active Statements** - shows the number of simultaneous running database statements.

The following online statistics are displayed during the test:

**Elapsed (mm:ss)** – The elapsed time for the running test.

**Time/Users remaining** – The remaining time or database statements.

**Active Statements** – The number of currently running database statements.

**Pending Statements** – The number of pending database statements.

**Statements Passed/Failed** – The total number of passed statement executions and the total number of failed statement executions.

**Performance** – The actual performance. The performance is the relation between the baseline execution time and the actual execution time.

**Avg. Exec. Time Delay** – the average difference between the baseline execution time and the actual execution time in milliseconds for each database statement execution over the last 5 seconds.

**Database Connections** – total number of opened database connections.

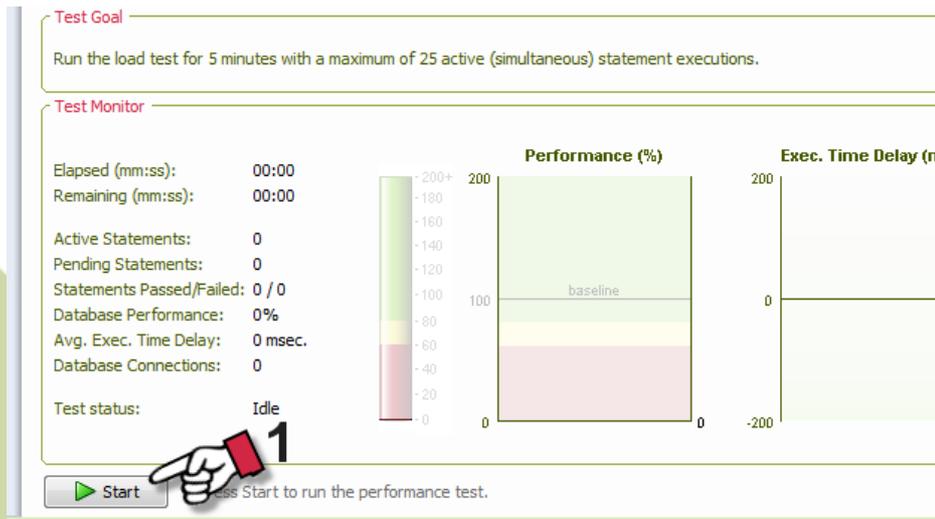
**Status** – current status of the load test: running, stopping, aborting and stopped.

Now you are ready to actually run the performance test.

Database2test has two run modes. Interactive (GUI) and batch (silent) mode. Batch or silent mode is required to schedule a performance test as a background job. You can run a performance test in batch mode using the operating system script console.bat. This script is located in the bin folder of the Database2test software.

Before running the test in batch mode you should first define you test scenario using the graphical interface.

For running the performance test in interactive (GUI) mode you have to press the Start button on the Test page (see 1).



Note that you can stop and abort the test at any time. If you stop the test no more new database statements are executed but the system will wait for the active database statements to complete gracefully.

However, when you abort the test no more new database statements are executed and the active database statements are terminated forcefully.

## ➤ Analyzing The Test Results

At completion of the performance test, Database2test will automatically generate comprehensive graphical reports (MS Excel) using the captured performance statistics and show you a Summary Report dialog.

The screenshot displays the Database2test 3 application window. A 'Performance Test Summary' dialog box is open in the center, providing a summary of the test results. The dialog box includes a 'Report' button, which is highlighted by a red arrow and the number '1'. Another red arrow and the number '2' point to the 'Test' button in the top right corner of the main application window. The background shows the main interface with a sidebar on the left and a main area with various charts and data.

The performance reports are saved to the output directory (see Settings Page). The performance reports have the following naming convention:

Whole scenario report: database2test\_<yyyymmdd>-<hhmiss>.xls

Per script reports: <script name>\_<yyyymmdd>-<hhmiss>.xls

During the performance test, Database2test captures performance statistics and error and log information to the log directory and the output directory. The output directory can be set in the Settings section.

**Database2test.err (log directory)** – this file contains a log of errors that might be useful for problem determination. This log is saved as a text file.

An error log entry has the following format:

[timestamp] [script name]([session id]): [log message]

**Database2test.log (log directory)** – this file contains runtime information about events during the test scenario. It shows among others the runtime parameter values, etc.

**Database2test.csv (output directory)** – this file contains whole scenario performance statistics captured by Database2test every five seconds during the test.

This file is in a comma-separated values (CSV) format and can be imported into a spreadsheet directly.

**<script name>.csv (output directory)** – Besides the whole scenario performance statistics Database2test captures also the performance statistics for each script. This file is also in a comma-separated values (CSV) format and contains the same metrics as the whole scenario performance metrics.

The graphical Excel reports are generated from the csv files above.

---

## > What It Means

**Elapsed** – Elapsed time since the start of the performance test.

**Active Statements** – Total number of active (running) database statements.

**Pending Statements** - Delayed statement executions due to active statement execution limit.

**Passed Statements** – Total number of passed database statements.

**Failed Statements** – Total number of failed database statements.

**Execution time** – The time to execute the database statement on the database server.

**Actual Performance** – The statement execution performance of the database for the last 5 seconds. A performance more than 100% means that the test execution time decreased compared to the baseline execution time. A performance less than 100% means that the test execution time increased compared to the baseline execution time.

**Overall performance** – The overall performance

**Target load** – Scheduled target load in statement executions per minute.

**Actual load** – Established load during the test in statement executions per minute.

**Generated Load** – Generated database statements per minute through Database2test.

**Connection Setups** – total number of database connections during the test.

**Cumulative execution time (sec)** – Sum of the execution times of all database statement executions during the test.

**Cumulative baseline execution time (sec)** - Sum of the baseline execution times for the database statement executions.

**Cumulative Gross Execution Time (sec)** – Sum of only the positive delays for the baseline and actual execution times of the database statement executions during the test.

**Execution time delay per statement execution (msec)** – Average difference between the baseline execution times and the actual execution times during the test over the last five seconds.

**Average execution time delay per statement execution (msec)** – Average difference between the baseline execution times and the actual execution times during the test for all database statement executions (in milliseconds).