# IBM Software Demos
# Rational Performance Tester Overview

<0:00>
IBM Rational Performance Tester is a performance testing tool for identifying the presence and cause of performance bottlenecks in Web, Socket, Citrix or ERP based applications such as SAP, Siebel, PeopleSoft and Oracle. Built for Day 1 Productivity, Performance Tester uses a code free approach that lets testers quickly create, edit and execute large volume performance tests. Using root cause analysis technology, Rational Performance Tester will reveal the presence and cause of system bottlenecks, enabling software teams to deliver applications that scale to meet user performance requirements.

<0:32>
The test creation process begins with a click on Rational Performance Tester's record button. The recorder enables performance tester to capture all traffic between our browser and our web application. The application we're testing today is an online mortgage service called Service Oriented Finance. We'll be testing the function which allows users to request an increase to their principal mortgage amount.

<0:55>
As we navigate the application, Performance Tester pays specific attention to transaction data, both user entered data, such as the requested mortgage increase amount, as well as server generated data, such as the request confirmation number. These elements will require special attention during the test editing process.

<1:13>
Performance Tester's Visual Test Editor represents our test actions in a tree view, where each branch on the tree represents a page we accessed during the test. Snapshots of each page are captured and displayed in the browser view below the test, such as our final page here, showing our request for a $100,000 increase and our request confirmation number 1277.

<1:36>
Performance Tester's visual test editor simplifies and accelerates the test creation process. All common test editing tasks such as adding loops, delays or if-then type decision structures can be done with a wizard, saving hours of manual coding. When the infrequent situation arises that requires a custom coded solution, users have the ability to add in manual HTTP requests or Java code at any point to work around any issue.

<2:04>

To ensure accurate server responses, Performance Tester offers verification points which will ensure the correct page size, code or content was received during a test. Again, no coding effort involved.

<2:16>
To ensure realistic load, it is often necessary to vary user entered data, such as our mortgage request increase amount. To simplify this task, Performance Tester creates variables for all user provided data, and enables user to substitute the data captured during recording with values from a datapool, a spreadsheet of unique values for each individual user.

<2:38>
Although masked from the novice user, Performance Tester has a sophisticated data correlation engine to manage data created by the web application, such as our request confirmation number. Dynamic data such as this is automatically tracked to its source, so that during playback, server generated data is substituted for recorded values to ensure playback continuity. In our example here, each individual user will look up the status of their unique confirmation number generated during test execution, not the confirmation number 1277 that was captured during recording.

<3:12>
Once test editing is complete, tests are typically run in single user mode to validate proper test functionality. In this mode, Performance Tester provides a page-by-page log of each test step, as well as a browser view to ensure proper playback. In our example here, we can see that our test ran correctly, using the first value from our data pool of 25,000, and received a new confirmation number 1278.

<3:37>
With our test validated, the next step is to include the test into a schedule for execution. Performance Tester's schedule editor allows testers to quickly and easily create schedules that will accurately emulate real user load on the system.

<3:50>
In the schedule we see here, we have identified two user groups that will perform actions on our system. The "Customer Viewers" group represents people browsing the site while the "Customer Updaters" group represents people requesting mortgage increases. The groups are proportional, where 75% of the site traffic is from the browsers, and 25% from the updaters. Each group will use a time based loop to execute transactions for 3 minutes.

<4:16>

We'll add our mortgage increase request test into the Customer Updaters loop. Should we choose, at this point we could add in additional schedule elements such as random events, or user synchronization points to ensure a realistic load.

<4:31>
User volumes can be constant or varied in stages. In our example, our test will begin with 5 users for the first minute, increase to 10 users for the second minute, and then finally increase to 15 users for the final minute.

<4:44>
During execution we can optionally choose to monitor our resource utilization. This will enable us to identify hardware performance bottlenecks related to resources such as CPU, disk or network utilization.

<4:57>
To enable us to pinpoint slow performing code, we can also turn on response time breakdown. This will enable us to drill down on our performance results right down to the source code to identify not only the presence, but root cause of performance bottlenecks.

<5:12>
With test creation and scheduling complete, we can now execute our test. Tests can be run locally from one machine, or distributed across multiple machines to create a larger load.

<5:22>
Performance Tester makes test execution results available in real-time, providing instant feedback to testers. Without this data, testers would have wait until run completion to evaluate their results – a process that can cause hours of wasted time when tests fail to run as expected.

<5:38>
Performance Tester monitors all key elements of server activity to provide a complete view of server performance. Response over time, page and server throughput can be used to pinpoint the starting point of a bottleneck. Resource monitoring data can be used to pinpoint hardware related slowdowns.

<5:56>
All graphs are annotated to indicate user load periods. Here, our page response times in the red zone occurred during the 5 user load period, the green zone is

when we had 10 users, and the blue zone represents response times for our 15 user load.

<6:11>
The most critical report is the page performance report which shows the average response time per page during the run. The slower the response time, the taller the bar – and we can see here that our Policy Change Confirmation page took almost 10 seconds, on average, to return.

<6:28>
To investigate the root cause of such bottlenecks, we can drill deeper into the response time breakdown data and view our response times by physical application tier and by class and method. If available, we can even drill right down to the source code to identify and repair the bottleneck.

<6:46>
IBM Rational Performance Tester is an application for novice and professional testers alike. Built for Day 1 Productivity, Performance Tester uses a code free approach that lets testers quickly create, edit and execute large volume performance tests. Using root cause analysis technology, Rational Performance Tester will reveal the presence and cause of system bottlenecks, enabling software teams to deliver applications that scale to meet user performance requirements.

<7:13>
To learn more about Performance Tester please visit our web site - ibm.com/rational - for additional product information and demos, and to download your free 30 day trial evaluation of IBM Rational Performance Tester.

END