Apache JMeter™ workshop

Ideas and examples

Piotr Sobieraj, July 2016

Date	Owner	Description
July 2016	Piotr	Review – to do is to adjust to jmeter 3.
June 2015	Piotr	Update to jmeter 2.13. Added the Performance Testing chapter.
November 2014	Piotr	Initial release against jmeter 2.11.

Table of Content

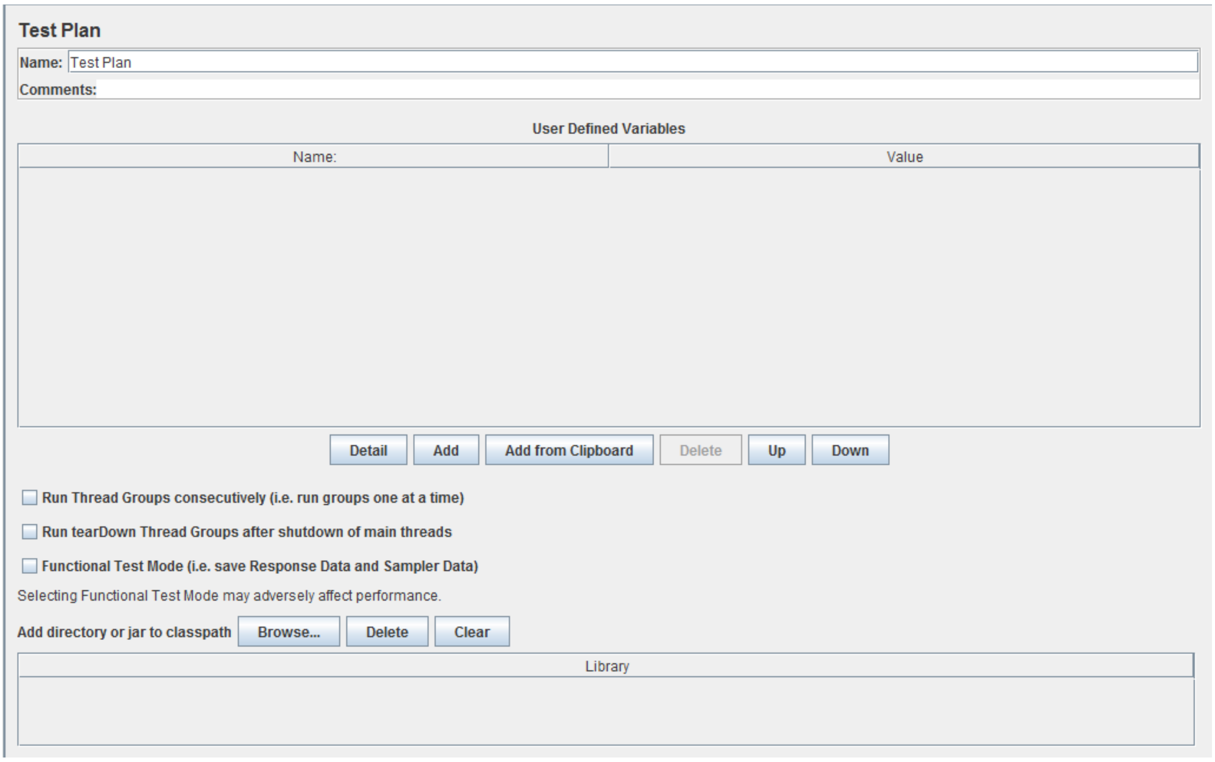
About Jmeter

Initially meant for load tests, developed for years became a functional tests tool as well. First release was issued in December 1998.

Test plan

A Test Plan is a series of steps that Jmeter executes during its runtime, defines and provides a layout of how and what to test. It can be perceived as a container for running tests.

A complete Test Plan will consist of one or more elements such as Thread Groups, logic controllers, sample-generating controllers, listeners, timers, assertions, and configuration elements. A test plan must have at least one Thread Group.



Test Plan

Name: Test Plan

Comments:

User Defined Variables

Name:	Value

□ Run Thread Groups consecutively (i.e. run groups one at a time)

□ Run tearDown Thread Groups after shutdown of main threads

□ Functional Test Mode (i.e. save Response Data and Sampler Data)

Selecting Functional Test Mode may adversely affect performance.

Add directory or jar to classpath

Library

Attributes

1. Run Thread Consecutively: by default all the threads coming under test plan run concurrently. If you check this option threads will run one after another, as defined in the Test Plan.

2. Run tearDown Thread Groups after shutdown of main threads: if selected, the tearDown groups (if any) will be run after graceful shutdown of the main threads. The tearDown threads won't be run if the test is forcibly stopped.

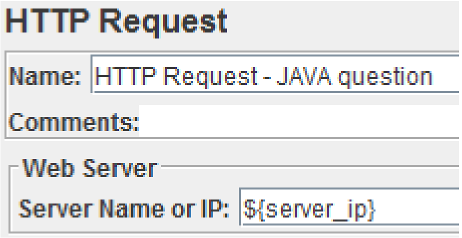
3. Functional Test Mode. If this option is checked Jmeter will save additional sample information – Response Data and Sampler Data to all result files affecting the performance of system.

User-Defined Variables:

Jmeter enables users to create their own variables. It is a name-value pair.
To use your variable anywhere inside a Test Plan go for notation: **${variable_name}**
To create variable that will be used for server ip address:

1. Click Add button.
2. Change empty Name to: **server_ip**
3. Change empty Value to: **127.0.0.1**

Now you could use **${server_ip}** as, for example, value of Server Name or IP field in HTTP Request (what will be explained wider later on):



HTTP Request

Name: HTTP Request - JAVA question

Comments:

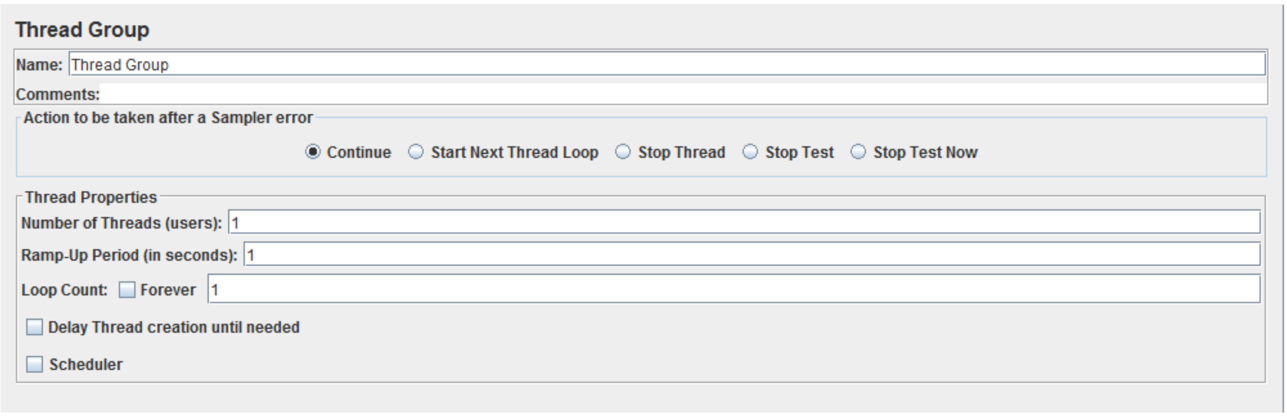
Web Server

Server Name or IP: ${server_ip}

Thread Group

Thread Group defines a pool of users that will execute a particular test case.

Each sample, assertion and any other relevant item can be grouped in Threads. More – a Thread Group acts like a loop. The below shows Thread Group control panel.



Thread Group

Name: Thread Group

Comments:

Action to be taken after a Sampler error

● Continue ○ Start Next Thread Loop ○ Stop Thread ○ Stop Test ○ Stop Test Now

Thread Properties

Number of Threads (users): 1

Ramp-Up Period (in seconds): 1

Loop Count: □ Forever 1

□ Delay Thread creation until needed

□ Scheduler

Action to be taken after a Sampler error	
Continue	Ignores the error and continue with the test.
Start Next Thread Loop	Ignores the error, start next loop and continue with the test.
Stop Thread	Exits the current thread.
Stop Test	Stops the entire test at the end of current samples. Useful especially when checking against some particular test conditions.
Stop Test Now	Stops the entire tests unconditionally.

Number of Threads (users)	Determines number of concurrent threads trying to sample.
Ramp-Up Period (in seconds)	Says how long (in seconds) Jmeter will keep reaching the given Number of Threads. Let's assume, there are 50 Threads and the Ramp-Up Period is set to 10. It means that after 10 s Jmeter will reach all the 50 Threads; in other words there will be 5 new Threads each second. The higher Ramp-Up the more gently warm-up to tested environment.
Loop Count	
Forever	Jmeter will run the test infinitely or until other controller says different.
Given number	The Thread will be repeated the given number times.
Delay Thread creation Until Needed	If not selected, Jmeter will create the Thread at the start of the Test Plan.
Scheduler	Enables scheduling options.

How to determine maximal Number of Threads?

It's not easy to set down this number. It depends on the resources of the machine which a test is run from. The golden rule says the maximum is reached when processor utilization is around 80%. Going above 80% might make the test even slower, since the machine is running out of its resources. On a common $1000 expensive machine it would be somewhere around 100 Threads.

Special setUp and tearDown types of Threads

Jmeter comes with two special Thread Groups types. The former precedes any other regular Thread's execution, while the latter is executed once any other regular Thread has finished.

HTTP Request

HTTP Request is a sampler that sends a requests to a server and waits for a response. Response from a sampler can be, for instance, passed for further calculation or validated against assertion.

This kind of sampler communicates over HTTP(S) protocol.

It can retrieve following resources: images, applets, style sheets, external scripts, frames, iframes, background images (body, table, td, tr), background sound.

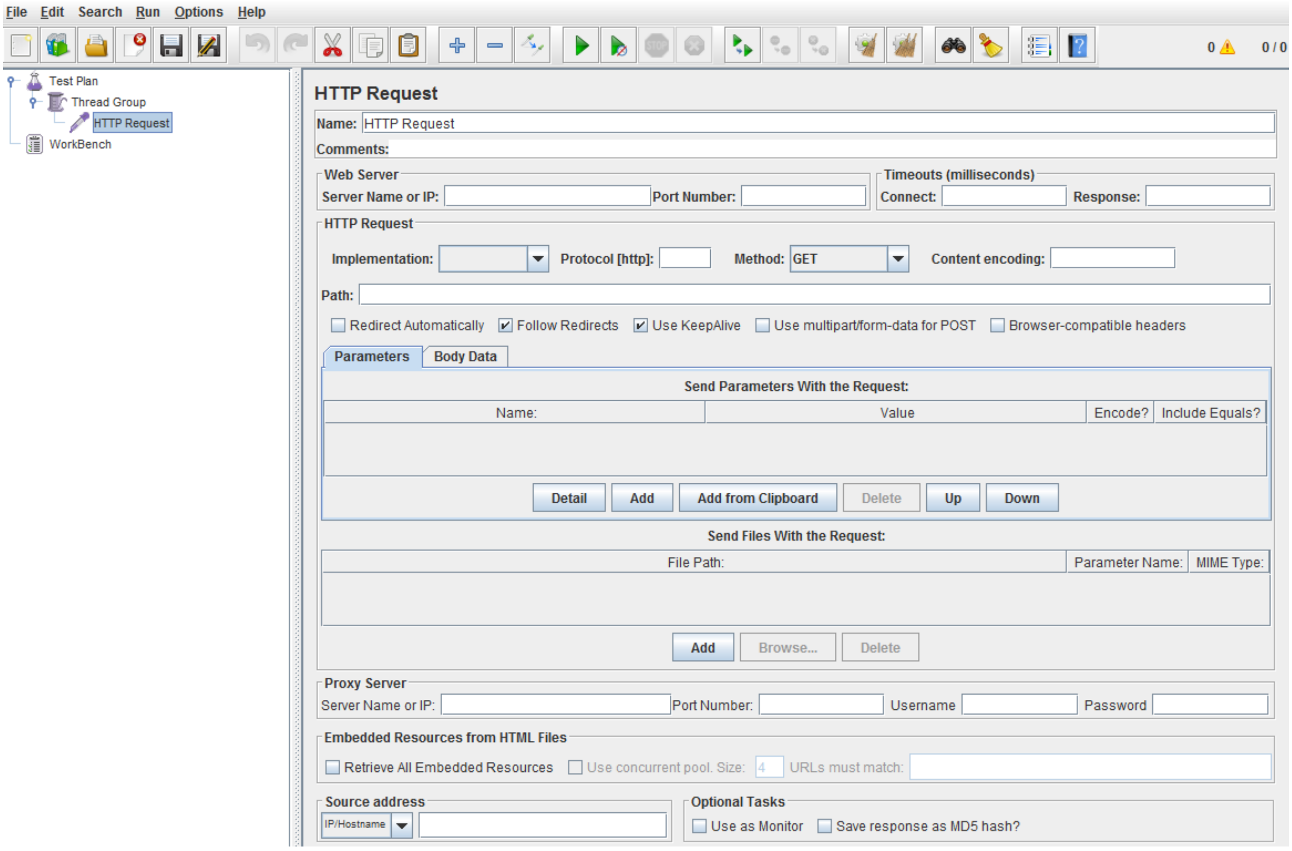
Aim of this lesson is to get to know how to use HTTP Request sampler in communication with an application.

Before sending any packet to the application, there are some basics to be explained:

1. Add Thread Group.
2. Right click on the Thread node.

3. Select Add > Sampler > HTTP Request.

You should have a picture similar to this one below:



Name: HTTP Request
Comments:

Web Server
Server Name or IP:
Port Number:

Timeouts (milliseconds)
Connect:
Response:

HTTP Request
Implementation:
Protocol [http]:
Method: GET
Content encoding:
Path:

□ Redirect Automatically ☑ Follow Redirects ☑ Use KeepAlive □ Use multipart/form-data for POST □ Browser-compatible headers

Proxy Server
Server Name or IP:
Port Number:
Username
Password

Embedded Resources from HTML Files
□ Retrieve All Embedded Resources □ Use concurrent pool. Size: 4 URLs must match:

Source address
IP/Hostname

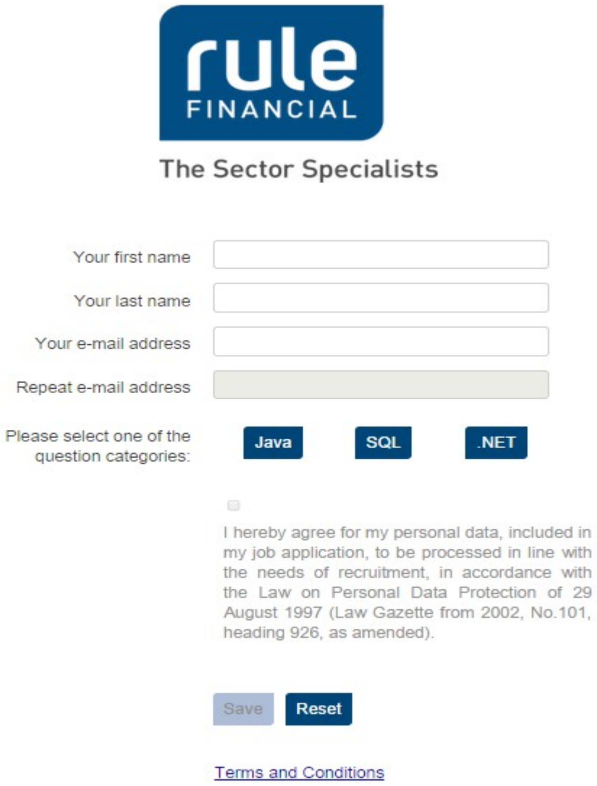
Optional Tasks
□ Use as Monitor □ Save response as MD5 hash?

We will set the following parameters and leave the rest with their defaults:

Attribute	Description
Name	Descriptive name for this sampler that is shown in the tree.
Server	Domain name or IP address of the web server. e.g. www.example.com. Do not include the http:// prefix.
Port	Port the web server is listening to. Default: 80.
Path	The path to resource (for example, /servlets/myServlet).

We will be sending request to our Contest application that is used during some external events. On your Virtual machines, it works at this URL: http://localhost:18080/registrationform

When you open this link in your browser, you will see something similar to this:



Your first name

Your last name

Your e-mail address

Repeat e-mail address

Please select one of the question categories:

□ I hereby agree for my personal data, included in my job application, to be processed in line with the needs of recruitment, in accordance with the Law on Personal Data Protection of 29 August 1997 (Law Gazette from 2002, No.101, heading 926, as amended).

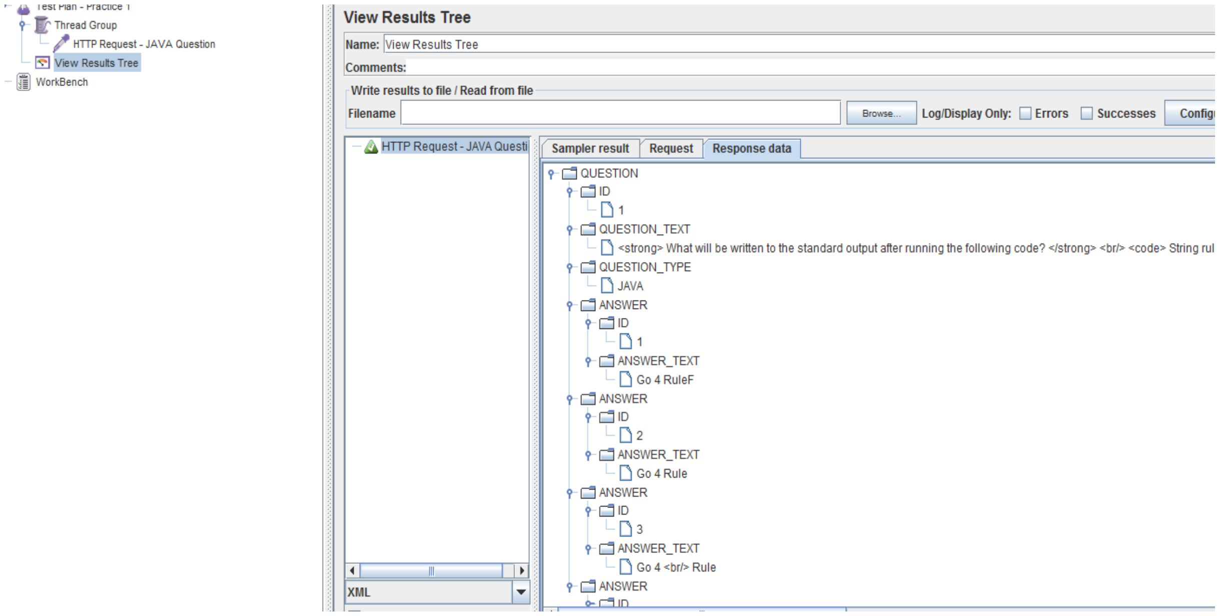
Practice 1

Using HTTP Request, retrieve a JAVA question. (Please note the question will be chosen randomly by the Contest application.)

Solution 1

1. Create an empty *Test Plan* and change its name to **Test Plan – Practice 1**.
2. Save the test plan by the name of Practice_01.jmx.
3. Go for Add > Threads (Users) > Thread Group.
4. Add HTTP Request (Add > Sampler > HTTP Request) to the Thread Group and change its name to **HTTP Request – JAVA question**. Change the relevant settings as below:
    a. Server Name or IP: 127.0.0.1
    b. Port Number: 18080
    c. Protocol [http]: HTTP
    d. Method: GET
    e. Content encoding: UTF-8
    f. Path: /registrationform/registrations/JAVA
5. Add View Result Tree to the Test Plan (Add > Listener > View Results Tree) to observe flow of the test.
6. Run test from menu Run > Start or use shortcut Ctrl+R.
7. Click on View Result Tree to see Response Data.

You should see something similar to this (please note to change formatting to XML):



Name: View Results Tree

Comments:

```
QUESTION
  ID
    1
  QUESTION_TEXT
    <strong> What will be written to the standard output after running the following code? </strong> <br/> <code> String rul
  QUESTION_TYPE
    JAVA
  ANSWER
    ID
      1
    ANSWER_TEXT
      Go 4 RuleF
  ANSWER
    ID
      2
    ANSWER_TEXT
      Go 4 Rule
  ANSWER
    ID
      3
    ANSWER_TEXT
      Go 4 <br/> Rule
  ANSWER
    ID
```

Assignment 1 (very easy)

Knowing that the Contest application comes with questions from JAVA, SQL and DOTNET domains, the assignment is to retrieve one question per domain.

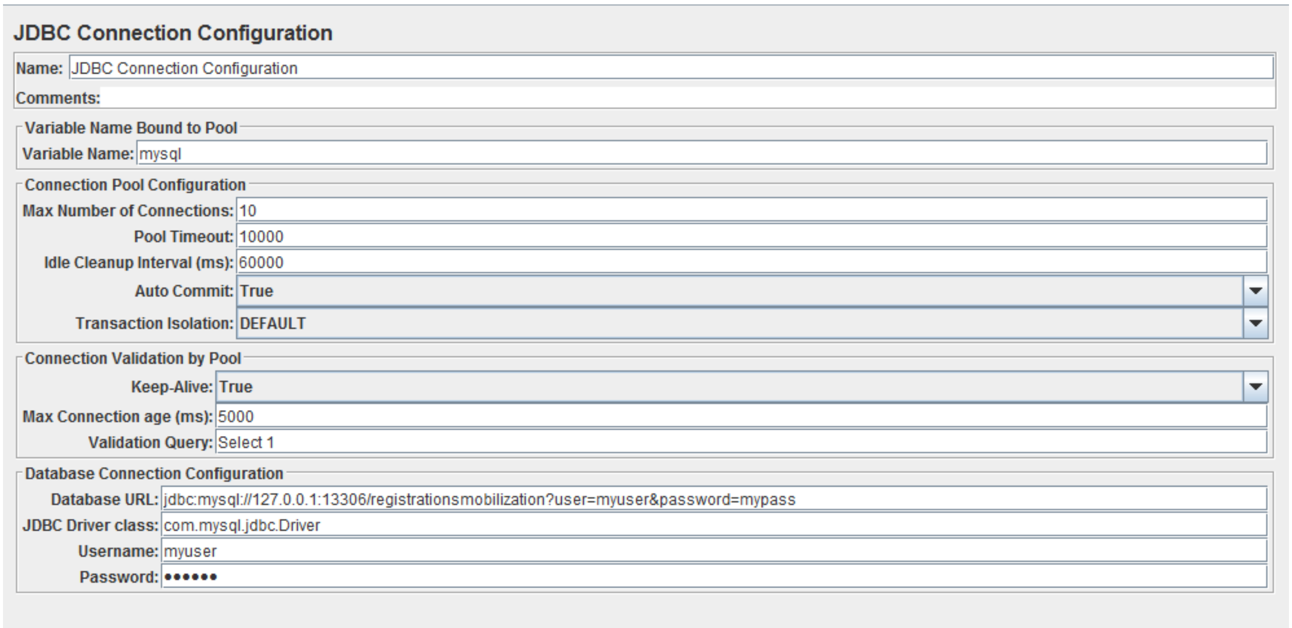
JDBC Connection Configuration

Jmeter itself comes with no database driver. For the purposes of this workshop we will use MySQL connectivity provided by Oracle MySQL itself. To install it just put the jar file (here mysql-connector-java-5.1.31-bin.jar) to the /lib folder and start Jmeter over. The whole procedure was done on the installation package you were given prior to the workshop.

JDBC Request

1. Create a new Test Plan.
2. Add to it and configure Config Elements > JDBC Connection Configuration as shown below.

Variable Name	A Name to use by JDBC samplers. Jmeter allows to use more than one database connectivity, so it must be said which connection should be chosen. Here mysql.
Database URL	A URL in format expected by MySQL driver, for MySQL jdbc:mysql://<host>:<port>/<database>?user=<user>&password=<password> Here jdbc:mysql://127.0.0.1:13306/registrationsmobilization?user=myuser&password=mypass
JDBC Driver Class	Here com.mysql.jdbc.Driver.
Username	Database user name. Here 'myuser'.
Password	Database password bound with the given user. Here 'mypass'.



JDBC Connection Configuration

Name: JDBC Connection Configuration

Comments:

Variable Name Bound to Pool

Variable Name: mysql

Connection Pool Configuration

Max Number of Connections: 10

Pool Timeout: 10000

Idle Cleanup Interval (ms): 60000

Auto Commit: True

Transaction Isolation: DEFAULT

Connection Validation by Pool

Keep-Alive: True

Max Connection age (ms): 5000

Validation Query: Select 1

Database Connection Configuration

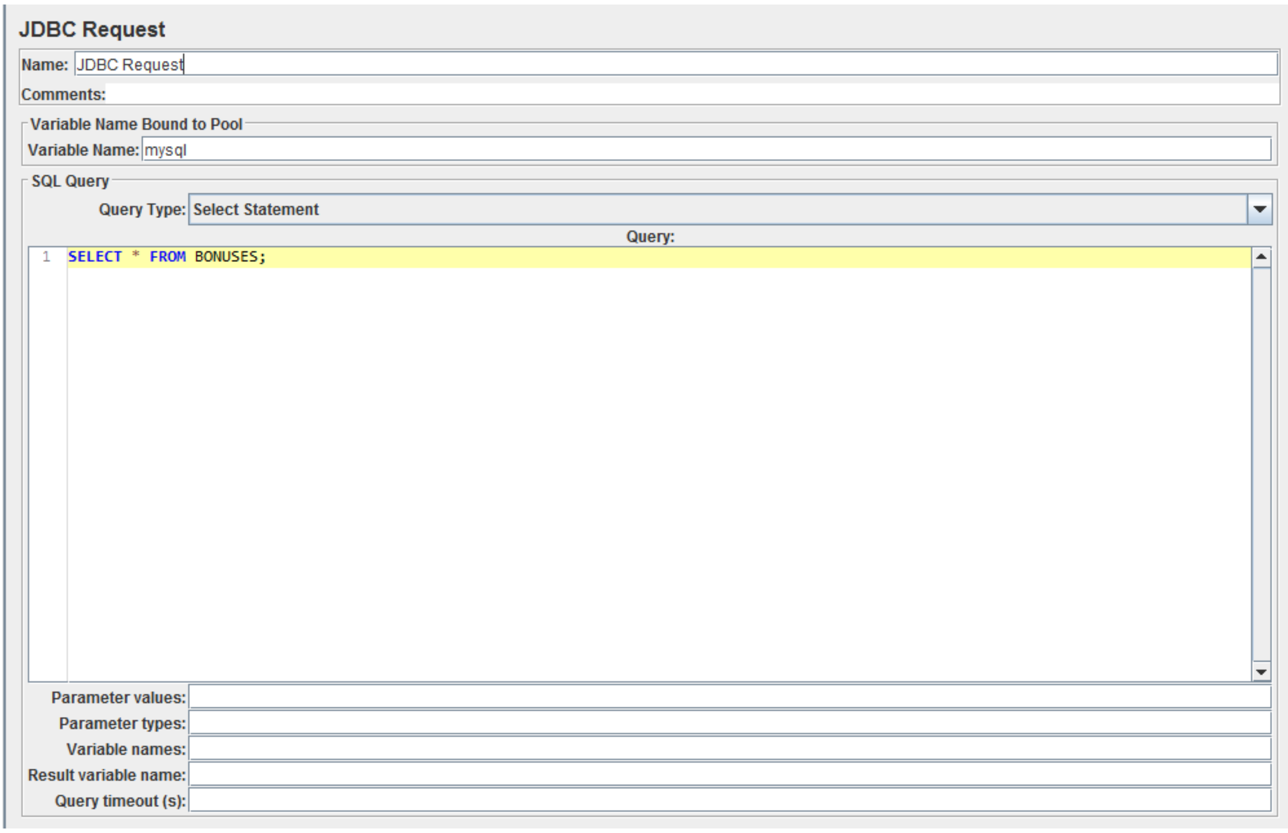
Database URL: jdbc:mysql://127.0.0.1:13306/registrationsmobilization?user=myuser&password=mypass

JDBC Driver class: com.mysql.jdbc.Driver

Username: myuser

Password: ●●●●●●

3. Add Thread Group to the Test Plan.
4. Add to it and set the Sampler > JDBC Request as follows:



JDBC Request

Name: JDBC Request

Comments:

Variable Name Bound to Pool

Variable Name: mysql

SQL Query

Query Type: Select Statement

Query:

```
SELECT * FROM BONUSES;
```

Parameter values:

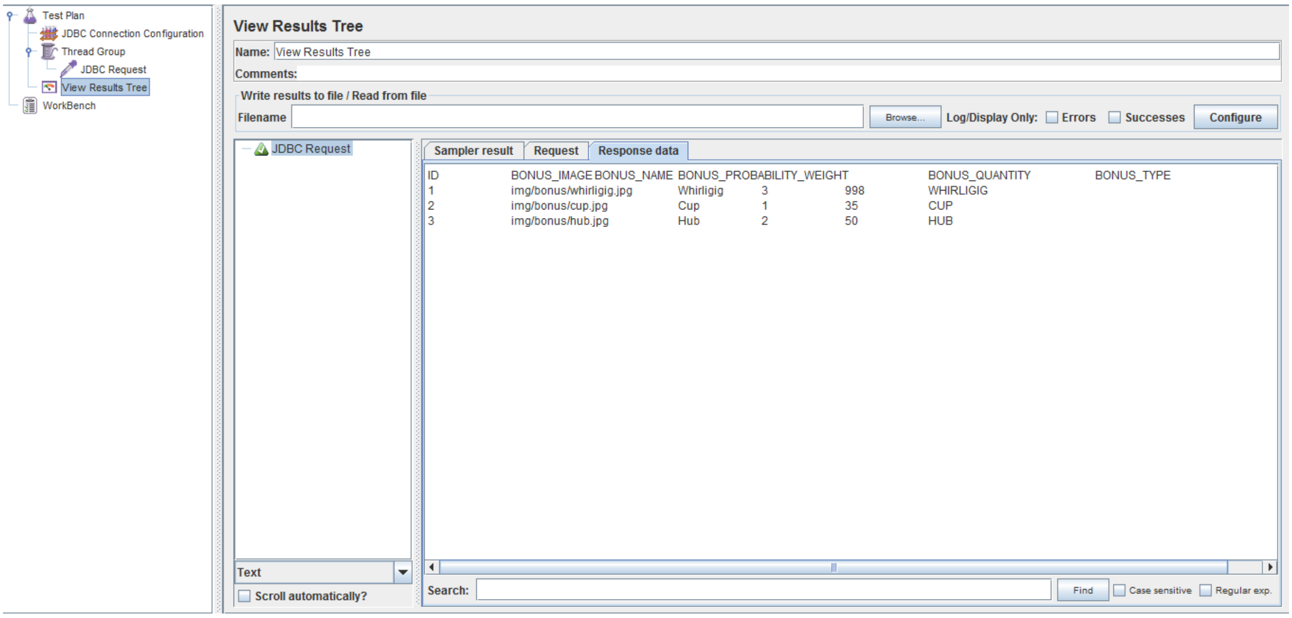
Parameter types:

Variable names:

Result variable name:

Query timeout (s):

Add a View Result Tree component to your Test Plan and put the other components as shown below. Run the test and observe View Result Tree behavior.



```
ID          BONUS_IMAGE BONUS_NAME  BONUS_PROBABILITY_WEIGHT          BONUS_QUANTITY          BONUS_TYPE
1           img/bonus/whirligig.jpg     Whirligig     3          998          WHIRLIGIG
2           img/bonus/cup.jpg           Cup           1          35           CUP
3           img/bonus/hub.jpg           Hub           2          50           HUB
```

Practice 2

The Practice is to test the following flow:

1. Add a user to the database using HTTP Request.
2. Check if it is actually added using JDBC Request.
3. More - the database is cleaned up before the test is run.

All will be done by Jmeter, though there was some investigation needed to get to know how to put a proper HTTP request.

Solution 2

1. Create an empty Test Plan and change its name to Test Plan - Practice 2.
2. Save the test plan by name Practice_02.jmx.
3. Add Config Element > JDBC Connection Configuration to the Test Plan and set its vital parameters as follows:
    1. Variable Name: mysql
    2. Database URL: jdbc:mysql://127.0.0.1:13306/registrationsmobilization?user=myuser&password=mypass
    3. JDBC Driver Class: com.mysql.jdbc.Driver
    4. Username: myuser
    5. Password: mypass
4. Add Config Element > HTTP Header Manager to the Test Plan.
    1. Add one item to the Headers Stored in the Header Manager:

Content-Type	text/xml

This is to inform HTTP Request that the content of request will be sent as XML.

5. Add Thread (Users) > Thread Group to the Test Plan.
6. Add Sampler > JDBC Request component to the Thread Group and change its name to JDBC Request - clean the database up.

   Remark: the REGISTRATIONS table comes with unique constrain put on the email column. If we didn't clean the database prior to every test run, we would be kept giving error (HTTP 400) coming from the application.

   1. Set Variable Name: mysql
   2. Change Query Type to Callable Statement.
   3. Put

      ```
      DELETE FROM REGISTRATIONS;
      ```

      in the Query box.
7. Add Sampler > HTTP Request to the Thread Group and change its name to “HTTP Request - add user to the database”. Change the relevant settings as below:
   1. Server Name or IP: 127.0.0.1
   2. Port Number: 18080
   3. Protocol [http]: HTTP
   4. Method: **POST**
   5. Content encoding: UTF-8
   6. Path: /registrationform/registrations
   7. Switch onto Body Data tab and put there (please feel free to come up with first, last name and email of your choice):

```
<?xml version="1.0" encoding="UTF-8"?>
<REGISTRATION>
   <FIRST_NAME>Piotr</FIRST_NAME>
   <LAST_NAME>Sobieraj</LAST_NAME>
   <EMAIL>piotr.sobieraj@gft.com</EMAIL>
   <QUESTION>Which one of the following keywords allows to sort the query
result?
         GROUP BY
         ORDER FROM
         ASC BY
         ORDER BY</QUESTION>
   <CHALLENGE>null</CHALLENGE>
   <ANSWER>null</ANSWER>
   <QUESTION_ID>7</QUESTION_ID>
   <ANSWER_INDEX>28</ANSWER_INDEX>
</REGISTRATION>
```

8. Add Assertions > Response Assertion to the HTTP Request and change the following settings:

1. Change its name to Response Assertion – check if 200.
2. Response Field to Test: Response code.
3. Pattern Matching Rules: Equals
4. Patterns to Test: 200.

9. Add Sampler > JDBC Request to the Thread Group and change its name to “JDBC Request - select newly added user from the database”.
    1. Set variable name: mysql
    2. Leave Query Type as it was by default (Select Statement).
    3. In Query box type in:

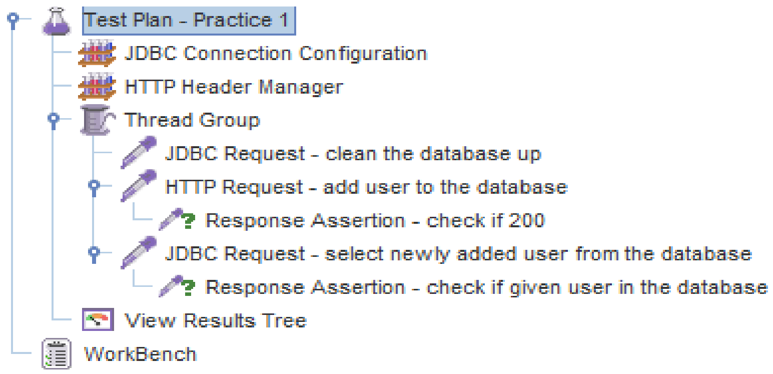
```
SELECT COUNT(*) FROM REGISTRATIONS WHERE EMAIL = 'piotr.sobieraj@gft.com';
```

    4. Type in how_many_rows in the Variable Names text field.

    Remark: comma separated values associated with Variable Names are objects to store a result of a query, column by column. These objects are iterable. In our case how_many_rows_# would equal the number of rows returned, and how_many_rows_1 would be the actual value of the first row. To see these values directly please add Sampler > Debug Sampler to the Thread Group, set Jmeter variables to true, and observe the output in the View Result Tree.

10. Add Assertions > Response Assertion to the newly added JDBC Request, change its name to “Response Assertion - check if given user in the database” and set:
    1. Response Field to Test: Jmeter variable.
        1. Type in how_many_rows_1 in the text field next to the radio button.
    2. Pattern Matching Rules: Equals.
    3. Pattern to test: 1.
11. Add Listener > View Result Tree to the Test Plan to observe test's flow.

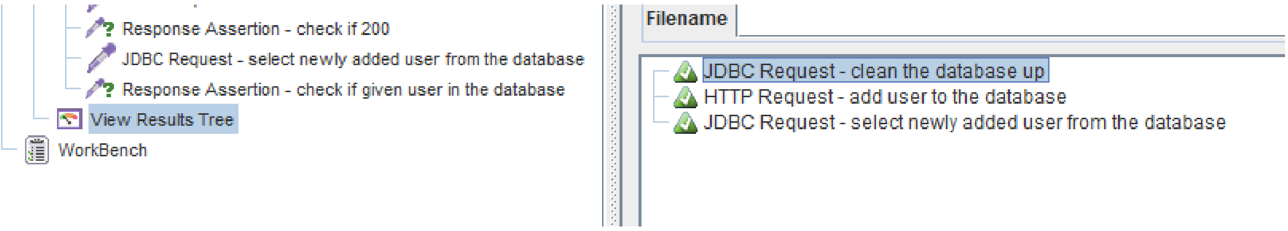
Your test plan should look similarly to the below picture:



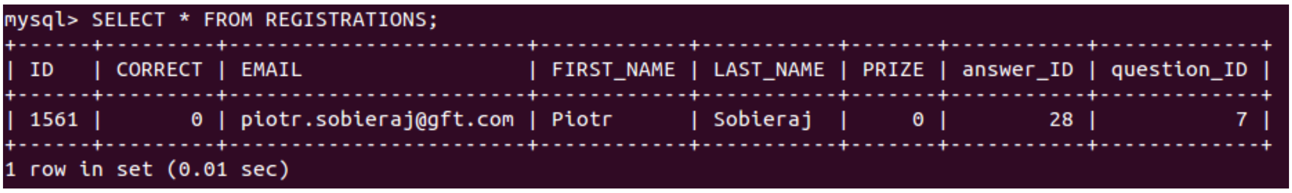
- Test Plan - Practice 1
    - JDBC Connection Configuration
    - HTTP Header Manager
    - Thread Group
        - JDBC Request - clean the database up
        - HTTP Request - add user to the database
            - Response Assertion - check if 200
        - JDBC Request - select newly added user from the database
            - Response Assertion - check if given user in the database
    - View Results Tree
- WorkBench

Now run the test.

In the View Result Tree component you should see the three proper responses:


Response Assertion - check if 200
JDBC Request - select newly added user from the database
Response Assertion - check if given user in the database
View Results Tree
WorkBench
Filename
JDBC Request - clean the database up
HTTP Request - add user to the database
JDBC Request - select newly added user from the database

If you went to the virtual machine and executed query against the REGISTRATION table, you would see something similar to the picture:



```
mysql> SELECT * FROM REGISTRATIONS;
+------+---------+------------------------+------------+-----------+-------+-----------+-------------+
| ID   | CORRECT | EMAIL                  | FIRST_NAME | LAST_NAME | PRIZE | answer_ID | question_ID |
+------+---------+------------------------+------------+-----------+-------+-----------+-------------+
| 1561 |       0 | piotr.sobieraj@gft.com | Piotr      | Sobieraj  |     0 |        28 |           7 |
+------+---------+------------------------+------------+-----------+-------+-----------+-------------+
1 row in set (0.01 sec)
```

Assignment 2 (harder, but still nice)

During building and debugging the previous plan you could probably notice that the tested application does not allow entering the same email twice. Now your assignment is to build a “regression test” which ensures that uniqueness of email addresses' is kept. In other words the job to do is to check if the application wouldn't allow entering the same email more than one.

Fetching the test data from an external file

It's a common need to execute multiple test cases against a particular test scenario. Jmeter provides a way to read such a data from a file, evaluates the fetched data into its variables and executes test cases.

We will practice it in a test scenario.

Practice 3

Register multiple users (3 of them) sequentially into our application. Use an external test data file.

Solution 3

1. Open your Practice_02.jmx script and save it by the name of Test Plan - Practice 3 (file Practice_03.jmx).
2. Add Config Element > HTTP Request Defaults to the Test Plan.
    1. Type 127.0.0.1 in the Server name or IP edit.
    2. Type 18080 in the Port box.
    3. Type HTTP in the Protocol (http) edit.
    4. Type UTF-8 in the Content Encoding box.

    5. Put /registrationform/registrations as Path.
    6. Clean Server name, Port HTTP, Content Encoding, Path field in the following componets:
        1. HTTP Request - add user to the database.

3. Add Threads (Users) > setUp Thread Group to the Test Plan and change its name to “setUp Thread Group - clean the database up”.
    1. Drag “JDBC Request - clean the database up” component to the newly added setUp Thread Group.
4. Tick the check button Forever in the Thread Group component. This feature, combined with CSV file configuration, will tell Jmeter to proceed the whole file from its beginning to end.
5. Create a file Practice_03.csv next to Practice_03.jmx.
6. Put there registration data of users to be added to our application.

```
fnamea|lnamea|email.1@gft.com
fnameb|lnameb|email.2@gft.com
fnamec|lnamec|email.3@gft.com
```

7. Add Config Element > CSV Data Set Config component to your main Thread Group and change its name to “CSV Data Set Config – users to register”.
    1. Type Practice_03.csv as Filename. Jmeter evaluates the path starting with the .jmx file.
    2. Determine UTF-8 as File encoding.
    3. Type *first_name,last_name,email* in the Variable Names field.
    4. Choose | (pipe) as Delimiter.
    5. Set *Allow Quoted Data?* and *Recycle on EOF?* to false.
    6. Set *Stop thread on EOF?* to true.
    7. Leave Sharing Mode as given by default (All threads).
8. Change name of “HTTP Request - add user to the database” to “HTTP Request - add user to the database - ${email}”.
9. Type in the following content into the “HTTP Request - add user to the database - ${email}” component. Please note the presence of the ${email} variable.

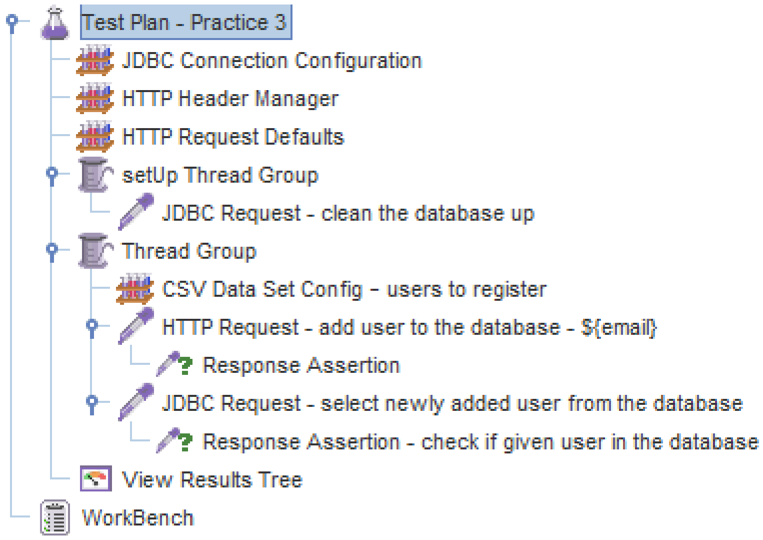
```
<?xml version="1.0" encoding="UTF-8"?>
<REGISTRATION>
   <FIRST_NAME>First Name</FIRST_NAME>
   <LAST_NAME>Last Name</LAST_NAME>
   <EMAIL>${email}</EMAIL>
   <QUESTION>Which one of the following keywords allows to sort the query result?
        GROUP BY
        ORDER FROM
        ASC BY
```

```
          ORDER BY</QUESTION>
    <CHALLENGE>null</CHALLENGE>
    <ANSWER>null</ANSWER>
    <QUESTION_ID>7</QUESTION_ID>
    <ANSWER_INDEX>28</ANSWER_INDEX>
</REGISTRATION>
```

10. Change JDBC Request - select newly added user from the database query box to

```
SELECT COUNT(*) FROM REGISTRATIONS WHERE EMAIL = '${email}';
```

The final state of the test plan:



```
Test Plan - Practice 3
    JDBC Connection Configuration
    HTTP Header Manager
    HTTP Request Defaults
    setUp Thread Group
        JDBC Request - clean the database up
    Thread Group
        CSV Data Set Config - users to register
        HTTP Request - add user to the database - ${email}
            Response Assertion
        JDBC Request - select newly added user from the database
            Response Assertion - check if given user in the database
    View Results Tree
WorkBench
```

Your test is now ready to be started off.

Assignment 3 (research required)

1. Make FirstName and LastName values to be fetched from the external CSV file. They should be passed to the query to add user to the database.
2. Add tearDown Thread Group component to clean the database up once the test is done.

Performance Testing

To proceed we will need to enrich Jmeter with more listeners. To do that go to the http://jmeter-plugins.org/ and pick Extras with Libs Set. To install the plug-ins just copy the files to the /lib folder of Jmeter. The whole procedure was done prior the workshop and was applied to the jmeter installation you are working with.

Standalone Performance Testing

Practice 4

Simulate a load of 100 different users being registered sequentially. Measure

- min and max response time
- median
- standard deviation
- Response Codes per Second
- Response Times Over Time
- Response Times Percentiles.

Solution 4

1. Open the Practice_03.jmx and save if by the name of Practice_04.jmx. Change its name to Test Plan - Practice 4.

2. Create a Practice_04.csv file and make its content equal 100 different emails, like

```
fname|lname|email.1@gft.com
…
fname|lname|email.100@gft.com
```

3. Make the component "CSV Data Set Config – users to register" point out to the Practice_04.csv file.

    1. Type *email* only in the Variable Names box.

4. Remove or disable "JDBC Request - select newly added user from the database" controller.

5. To the Test Plan add:

    Listener > Summary Report

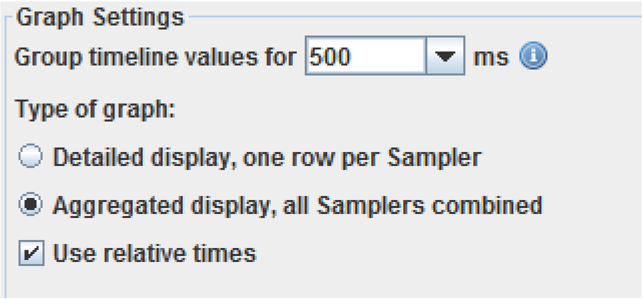
    Listener > Aggregate Report

    Listener > jp@gc - Response Codes per Second

    Listener > jp@gc - Response Times Over Time

    Listener > jp@gc - Response Times Percentiles

6. In the two last controllers go to the Settings tab and make *Aggregated display* selected.



Graph Settings

Group timeline values for 500 ms

Type of graph:

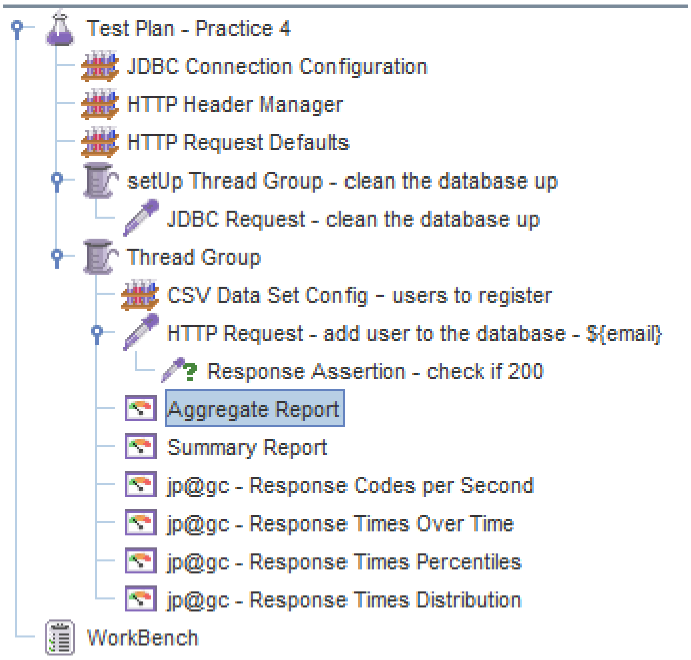
○ Detailed display, one row per Sampler

● Aggregated display, all Samplers combined

☑ Use relative times

7. To avoid counting in database operations make sure all the listeners are children of the main

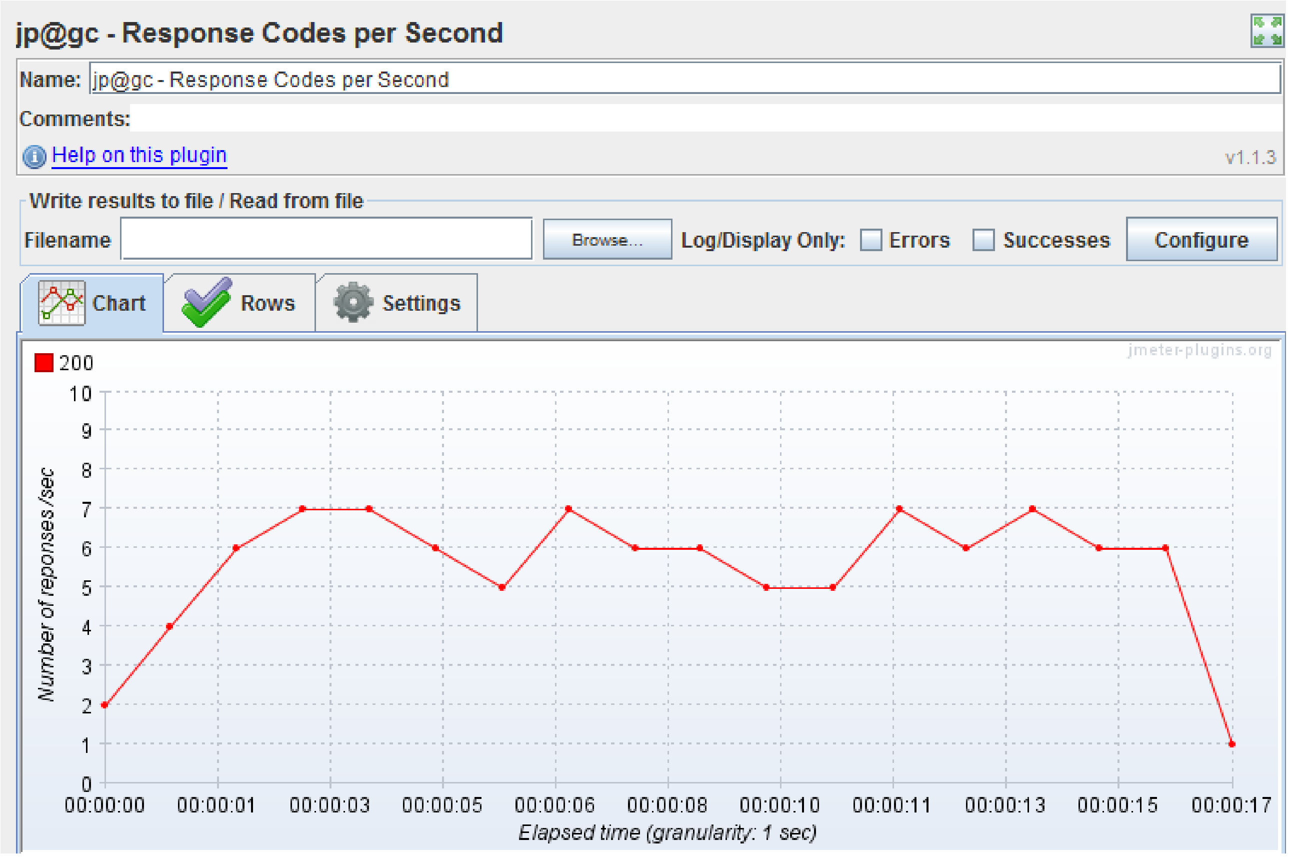
Thread Group.



Test Plan - Practice 4
JDBC Connection Configuration
HTTP Header Manager
HTTP Request Defaults
setUp Thread Group - clean the database up
JDBC Request - clean the database up
Thread Group
CSV Data Set Config - users to register
HTTP Request - add user to the database - ${email}
Response Assertion - check if 200
Aggregate Report
Summary Report
jp@gc - Response Codes per Second
jp@gc - Response Times Over Time
jp@gc - Response Times Percentiles
jp@gc - Response Times Distribution
WorkBench

8. Run the test.

Example outcome is shown below.



jp@gc - Response Codes per Second

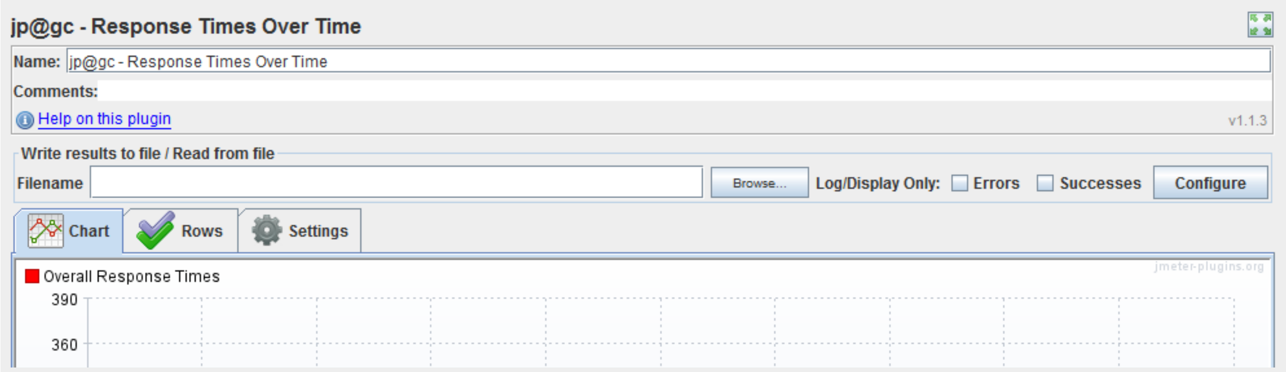
Name: jp@gc - Response Codes per Second

Comments:

Write results to file / Read from file

Filename

Log/Display Only: □ Errors □ Successes



jp@gc - Response Times Over Time

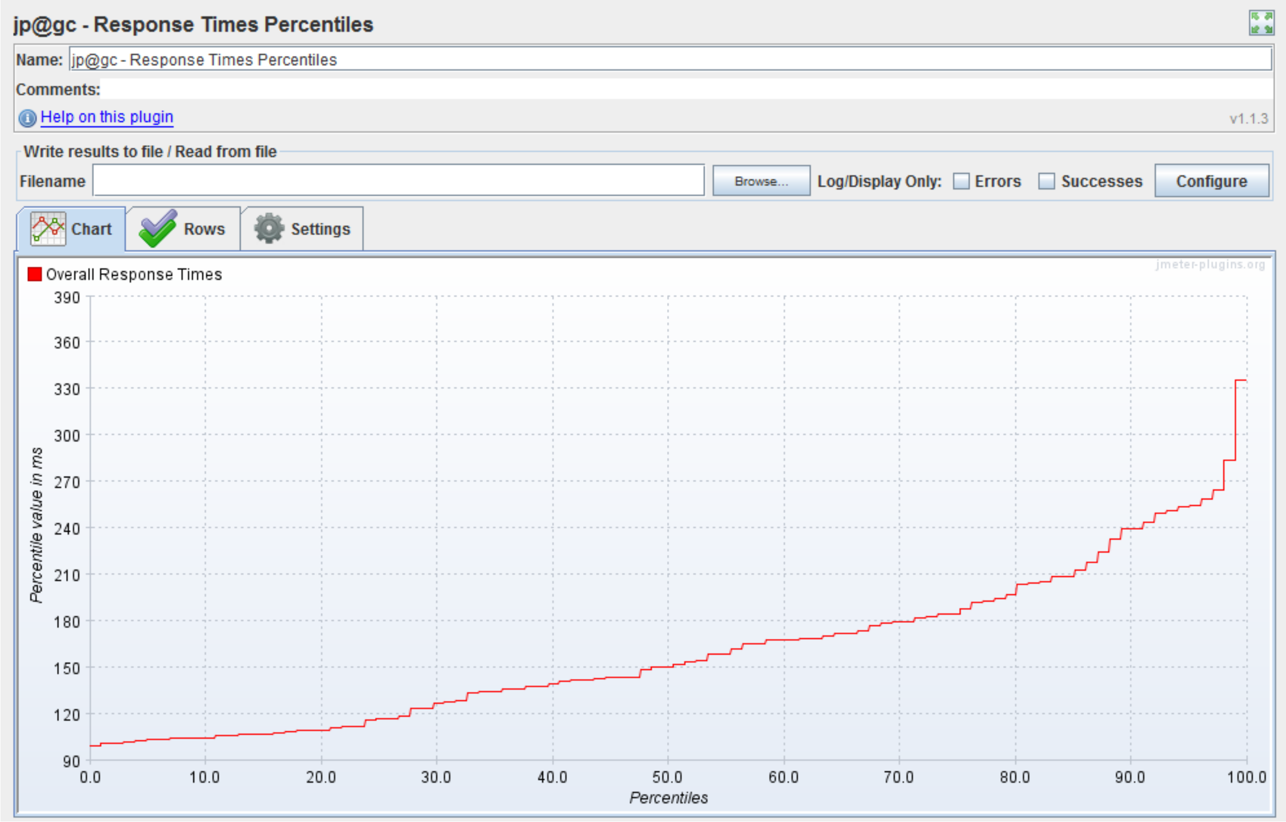
Name: jp@gc - Response Times Over Time

Comments:

Write results to file / Read from file

Filename

Log/Display Only: □ Errors □ Successes



jp@gc - Response Times Percentiles

Name: jp@gc - Response Times Percentiles

Comments:

Write results to file / Read from file

Filename

Log/Display Only: □ Errors □ Successes

Tips

Turn off listeners you don't need. Especially the View Result Tree listener is a resource-consuming one.
Turn off all the unnecessary operations, like checking against the database.
Turn off all the unnecessary assertions.
Turn off Functional Test mode in the Test Plan.

Distributed Performance Testing

Every local machine gets saturated at some point. To mitigate that effect Jmeter offers a way to distribute load testing. It works in client node – server nodes architecture.

```
client node
                |---server node 1 (JmeterEngine)
                ...
                |---server node n (JmeterEngine)
```

The client sends samples to its servers, servers send back the results. The results are collected by the client.

Practice 5

Simulate a load of 100 different users being registered sequentially. Measure Response Codes per Second, Response Times Over Time and Response Times Percentiles. Assume there are 10 users being registered "at the same time".

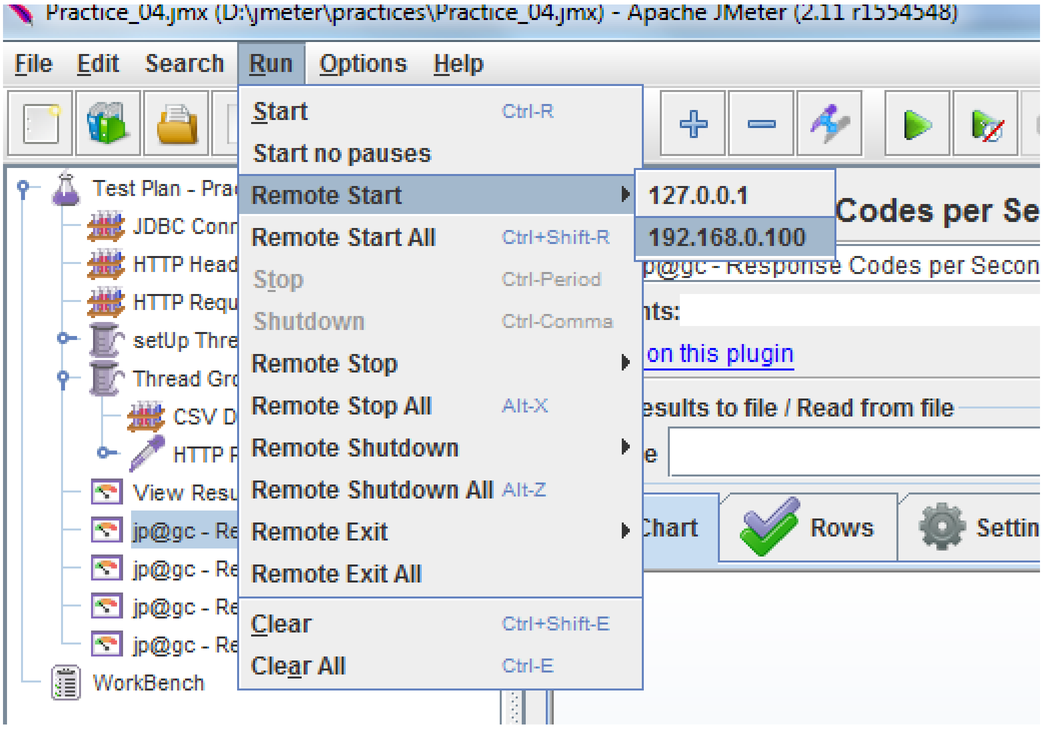
Remark: to avoid network related issues it can be done locally. It means the client and the server (JmeterEngine) can be run on the same physical machine.

Solution 5

1. Obtain your IP.
2. Close Jmeter. Edit the /bin/jmeter.properties file and add the given IP to the remote_hosts parameter. These values are comma-separated.

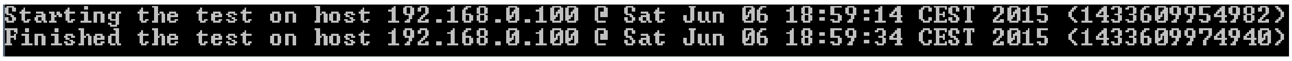
```
# Remote Hosts - comma delimited
remote_hosts=127.0.0.1,192.168.0.100
```

3. Launch /bin/jmeter-server.bat batch file.
4. Launch /bin/ApacheJMeter.jar.
5. Load the Practice_04.jmx test plan and save it by the name of Practice_05.jmx. Make the same for the .csv file.
6. Put number 10 in the Thread Group.Number of Threads (users) field.
7. From menu Run choose *Remote Start* and then pick the newly added machine.



If there were more servers you could use the *Remote Start All* entry.

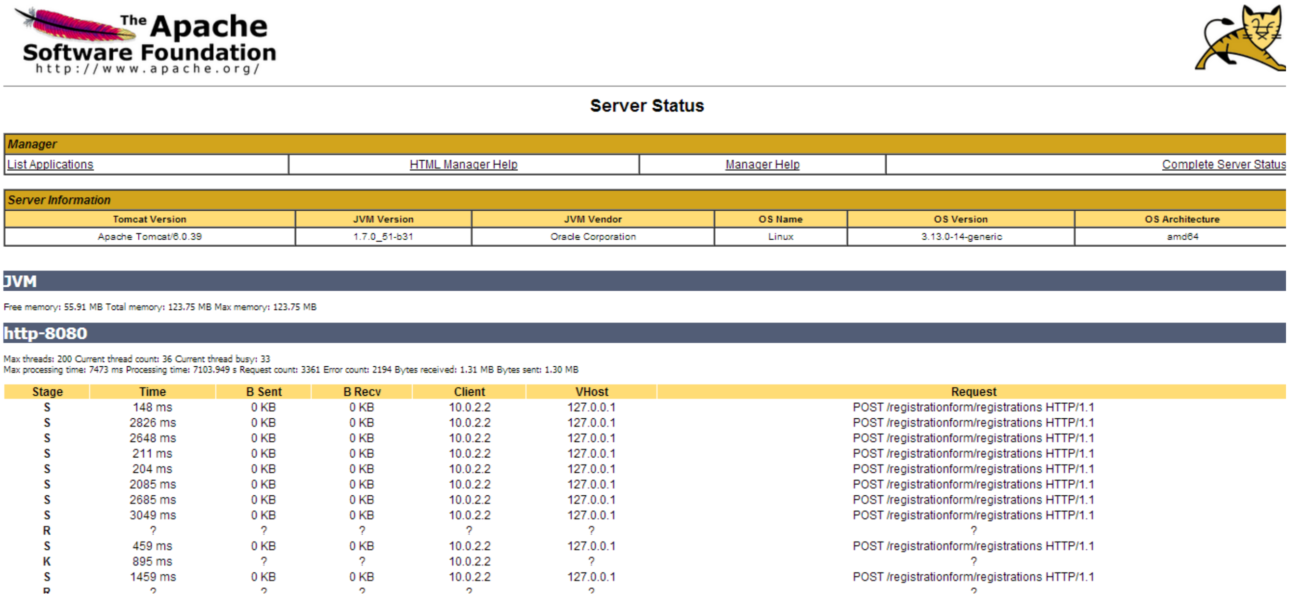
8. In the server console there will be appropriate information about the Test Plan starting and finishing. The results are collected on the client's side.



```
Starting the test on host 192.168.0.100 @ Sat Jun 06 18:59:14 CEST 2015 (1433609954982)
Finished the test on host 192.168.0.100 @ Sat Jun 06 18:59:34 CEST 2015 (1433609974940)
```

Monitor Results

Jmeter offers monitoring a server. We will take advantage of of the status page exposed at http://127.0.0.1:18080/manager/status (user and password are 'user').



Server Status

Manager			
List Applications	HTML Manager Help	Manager Help	Complete Server Status

Server Information					
Tomcat Version	JVM Version	JVM Vendor	OS Name	OS Version	OS Architecture
Apache Tomcat/6.0.39	1.7.0_51-b31	Oracle Corporation	Linux	3.13.0-14-generic	amd64

JVM

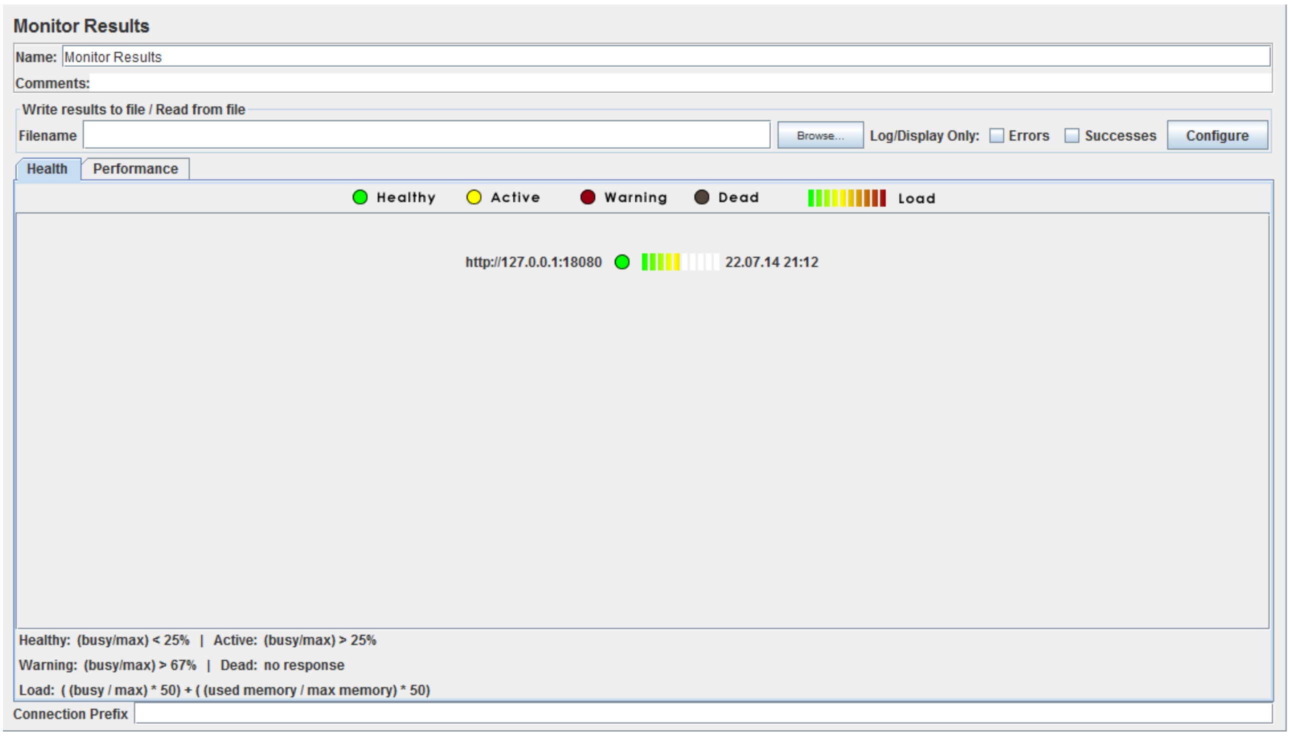
Free memory: 55.91 MB Total memory: 123.75 MB Max memory: 123.75 MB

http-8080

Max threads: 200 Current thread count: 36 Current thread busy: 33
Max processing time: 7473 ms Processing time: 7103.949 s Request count: 3361 Error count: 2194 Bytes received: 1.31 MB Bytes sent: 1.30 MB

Stage	Time	B Sent	B Recv	Client	VHost	Request
S	148 ms	0 KB	0 KB	10.0.2.2	127.0.0.1	POST /registrationform/registrations HTTP/1.1
S	2826 ms	0 KB	0 KB	10.0.2.2	127.0.0.1	POST /registrationform/registrations HTTP/1.1
S	2648 ms	0 KB	0 KB	10.0.2.2	127.0.0.1	POST /registrationform/registrations HTTP/1.1
S	211 ms	0 KB	0 KB	10.0.2.2	127.0.0.1	POST /registrationform/registrations HTTP/1.1
S	204 ms	0 KB	0 KB	10.0.2.2	127.0.0.1	POST /registrationform/registrations HTTP/1.1
S	2085 ms	0 KB	0 KB	10.0.2.2	127.0.0.1	POST /registrationform/registrations HTTP/1.1
S	2685 ms	0 KB	0 KB	10.0.2.2	127.0.0.1	POST /registrationform/registrations HTTP/1.1
S	3049 ms	0 KB	0 KB	10.0.2.2	127.0.0.1	POST /registrationform/registrations HTTP/1.1
R	?	?	?	?	?	?
S	459 ms	0 KB	0 KB	10.0.2.2	127.0.0.1	POST /registrationform/registrations HTTP/1.1
K	895 ms	?	?	10.0.2.2	?	?
S	1459 ms	0 KB	0 KB	10.0.2.2	127.0.0.1	POST /registrationform/registrations HTTP/1.1
R	?	?	?	?	?	?

Component Monitor Results asks this page for a status, gathers the responses and produces graphs similar to these ones below. There is an option to monitor a number of servers.



Monitor Results

Name: Monitor Results

Comments:

Write results to file / Read from file

Filename

Log/Display Only: □ Errors □ Successes

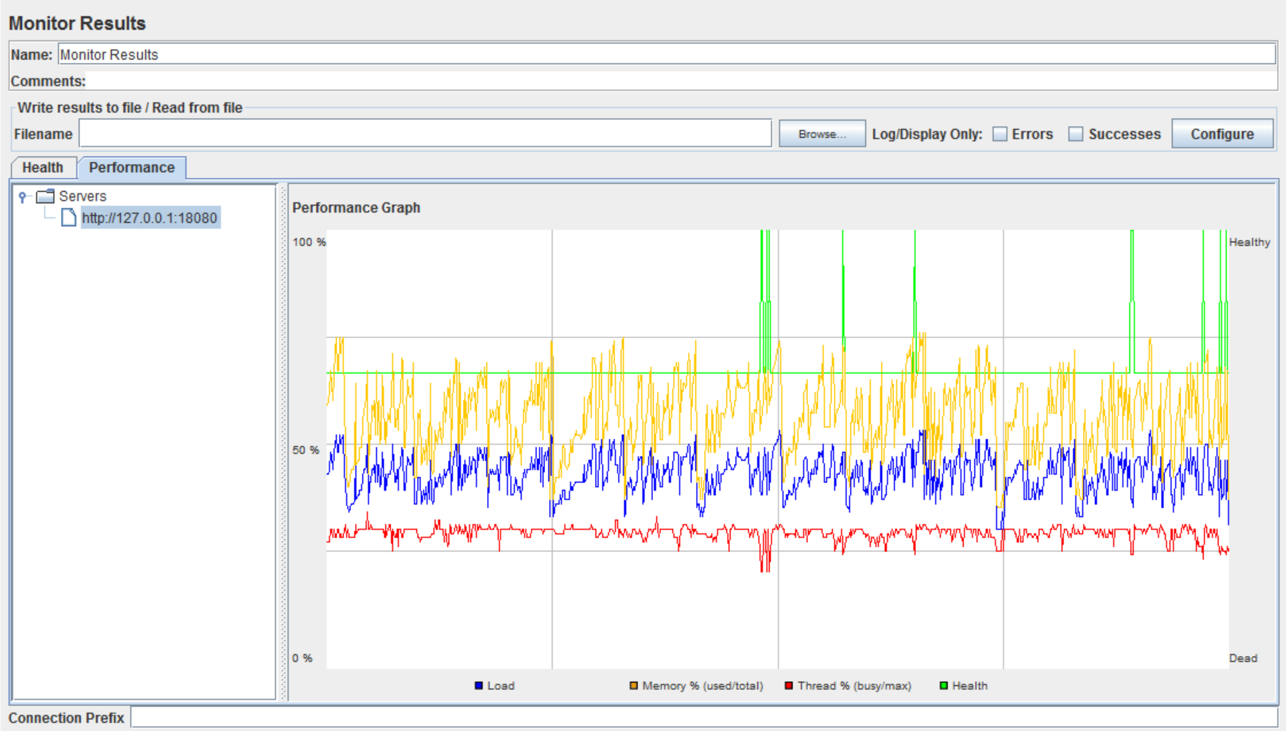
http://127.0.0.1:18080 ● 22.07.14 21:12

Healthy: (busy/max) < 25% | Active: (busy/max) > 25%

Warning: (busy/max) > 67% | Dead: no response

Load: ((busy / max) * 50) + ((used memory / max memory) * 50)

Connection Prefix



Monitor Results

Name: Monitor Results

Comments:

Write results to file / Read from file

Filename

Log/Display Only: □ Errors □ Successes

Connection Prefix

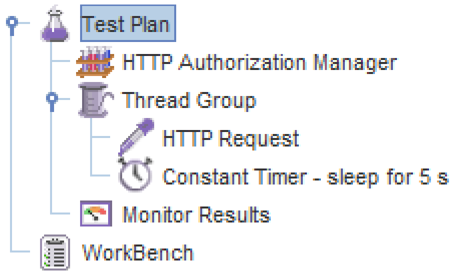
Practice 6

Build the Monitor Results test plan.

Solution 6

1. Create a new Test Plan, name it Test Plan – Practice 6 and save it as Practice_06.jmx.
2. Add Config Element > HTTP Authorization Manager to the Test Plan.
3. Add Authorization to the list and set:
    1. Both Username and Password to 'user' (with no quotes).
    2. Mechanism to BASIC_DIGEST.
4. Add Thread Group to the Test Plan, leave all the options except for Loop Count as they came by default (Action to be taken after a Sampler Error = Continue, Number of Threads = 1, Ramp-Up Period= 1).
    1. Set Loop Count to Forever.
5. Add Sampler > HTTP Request to the Thread Group.
    1. Set Server Name or IP to 127.0.0.1.
    2. Set Port Number to 18080.
    3. Set Path to /manager/status
    4. In the Parameters tab add an entry with the following settings:
        1. Name = XML (it's case sensitive)
        2. Value = true.
    5. In the Optional Tasks group box check Use as Monitor.
6. Add Timer > Constant Timer to the Thread Group and change its name to Constant Timer - sleep for 5 s.
    1. Type 5000 in the Thread Delay (in milliseconds) edit box.
7. Add Listener > Monitor Results to the Test Plan.

Your Solution should look like below picture



Test Plan
HTTP Authorization Manager
Thread Group
HTTP Request
Constant Timer - sleep for 5 s
Monitor Results
WorkBench

Now you can run the test plan and observe server status.

Remark: Monitor Results does not store any data by itself. To do such a thing you can use Simple

Data Writer component, store the results in a CSV file and till them as you like.

BeanShell Console

BeanShell is a Java-like scripting language allowing to create custom outputs, perform logical operations, manipulate Jmeter variables etc. It gives the user almost full control on Jmeter while a test is run.

The following BeanShell variables are exposed to users.

Var	Description	Example
log	Can be used to print something to jmeter.log file or embedded console.	log.info("This came from a BeanShell postprocessor"); log.error("This is the ERROR level message");
ctx	Represents org.apache.jmeter.threads.JMeterContext class which is pretty much JMeter itself, it provides read/write access to underlying JMeter Engine, Samplers and their Results as well as to variables/properties. Further reading at JmeterContext.	int threadNum = ctx.getThreadNum();

vars	Stands for Jmeter variables. Can be used to get or set a variable.	String myvar = vars.get("myvar");
		vars.put("myvar", 123);

props	Stands for JMeter Properties. Basically the same as variables, but variables visibility is limited to current thread group only and properties are "global".	Like above.

prev	Represents previous SampleResult.	String code = prev.getResponseCode();
		String msg = prev.getResponseMessage();

data	Byte array to contain parent sampler response data.	String samplerData = new String(data);
		System.out.println(samplerData);

For further reading please visit BeanShell manual.

Practice 7

Let's consider Practice 3 scenario (register multiple users – fetch data from a file). The test plan will be altered in a way, that:

1. We will check on email uniqueness and handle the exception with a BeanShell script.
2. We will display the currently processed email in Jmeter console.
3. We will log some data to a file.

Solution 7

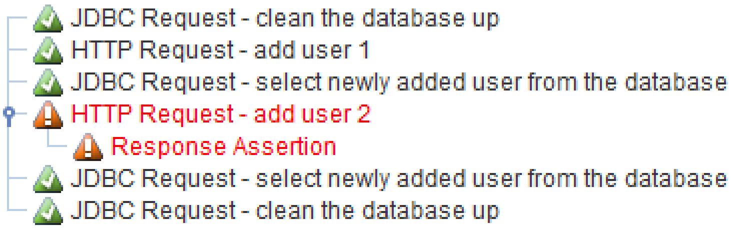
1. Open your Practice_03.jmx test plan, change its name to "Test Plan - Practice 7" and save it by the name of Practice_07.jmx.
2. Open your Practice_03.csv file and save it by the name of Practice_07.csv.
   1. Make it come with two the same entries, like

```
fnamea|lnamea|email.1@gft.com
fnamea|lnamea|email.1@gft.com
```

3. Go to the "CSV Data Set Config – users to register" component and change Filename entry

to make it indicate Practice_07.csv file.

4. Add Config Element > Counter to the Thread Group.
    1. Set Start =1
    2. Increment = 1
    3. Reference name set to 'counter'.
    4. Leave other fields empty.
5. Change “HTTP Request - add user to the database - ${email}” to “HTTP Request - add user ${counter}”.
6. If you run the test now, it will fail along with the second request.



JDBC Request - clean the database up
HTTP Request - add user 1
JDBC Request - select newly added user from the database
HTTP Request - add user 2
Response Assertion
JDBC Request - select newly added user from the database
JDBC Request - clean the database up

7. Remove Response Assertion (that one which fails).
8. Add Post Processors > BeanShell PostProcessor to the “HTTP Request - add user ${counter}” controller.
    1. Change its name to “BeanShell PostProcessor - switch 400 to 200”.
    2. Put in the script box:

```
if(prev.getResponseCode().equals("400")== true){
   prev.setResponseOK();
}
```

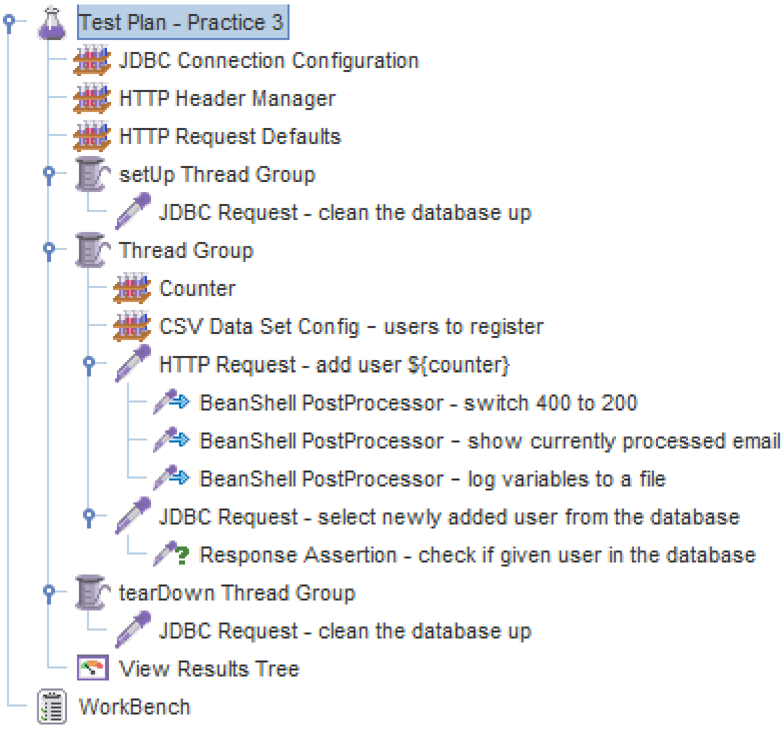
9. Add another BeanShell PostProcessor to the HTTP Request and change its name to “BeanShell PostProcessor – show currently processed email”.
    1. Type in the script box:

```
String email = vars.get("email");
log.info("The currently processed email is " + email);
```

10. Add yet another BeanShell PostProcessor to the HTTP Request and change its name to “BeanShell PostProcessor – log variables to a file”.
    1. Put in the script box:

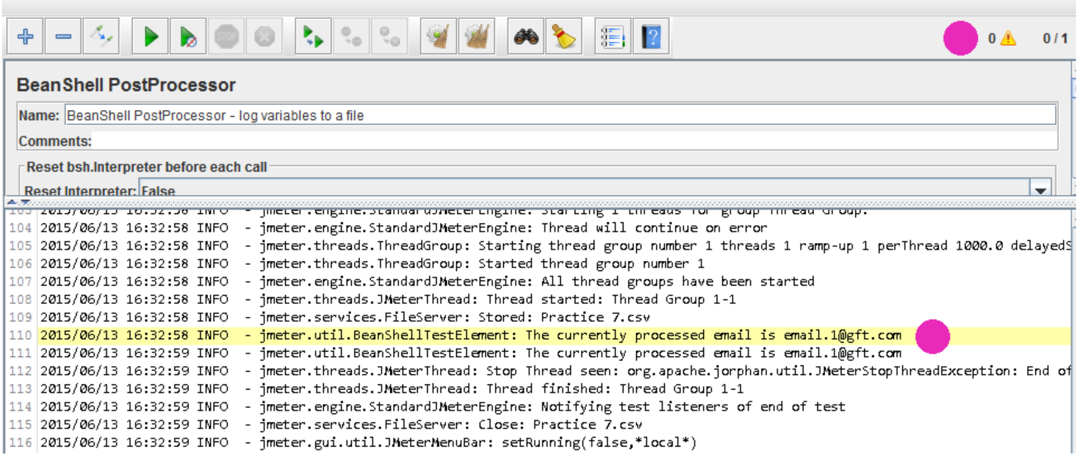
```
PrintStream out = new PrintStream(new FileOutputStream("filename.txt",
true));
String counter = vars.get("counter");
String email = vars.get("email");

out.println("Email "+counter+" is "+email+".");
out.close();
```

The resultant test plan is:



```
Test Plan - Practice 3
    JDBC Connection Configuration
    HTTP Header Manager
    HTTP Request Defaults
    setUp Thread Group
        JDBC Request - clean the database up
    Thread Group
        Counter
        CSV Data Set Config - users to register
        HTTP Request - add user ${counter}
            BeanShell PostProcessor - switch 400 to 200
            BeanShell PostProcessor - show currently processed email
            BeanShell PostProcessor - log variables to a file
        JDBC Request - select newly added user from the database
            Response Assertion - check if given user in the database
    tearDown Thread Group
        JDBC Request - clean the database up
    View Results Tree
WorkBench
```

11. Run the test and check observe that:
    1. The HTTP 400 error is not reported anymore
    2. In the jmeter console there is information about currently processed email



BeanShell PostProcessor

Name: BeanShell PostProcessor - log variables to a file

Comments:

Reset bsh.Interpreter before each call

Reset Interpreter: False

```
2015/06/13 16:32:58 INFO  - jmeter.engine.StandardJMeterEngine: Starting 1 threads for group Thread Group.
2015/06/13 16:32:58 INFO  - jmeter.engine.StandardJMeterEngine: Thread will continue on error
2015/06/13 16:32:58 INFO  - jmeter.threads.ThreadGroup: Starting thread group number 1 threads 1 ramp-up 1 perThread 1000.0 delayedS
2015/06/13 16:32:58 INFO  - jmeter.threads.ThreadGroup: Started thread group number 1
2015/06/13 16:32:58 INFO  - jmeter.engine.StandardJMeterEngine: All thread groups have been started
2015/06/13 16:32:58 INFO  - jmeter.threads.JMeterThread: Thread started: Thread Group 1-1
2015/06/13 16:32:58 INFO  - jmeter.services.FileServer: Stored: Practice 7.csv
2015/06/13 16:32:58 INFO  - jmeter.util.BeanShellTestElement: The currently processed email is email.1@gft.com
2015/06/13 16:32:59 INFO  - jmeter.util.BeanShellTestElement: The currently processed email is email.1@gft.com
2015/06/13 16:32:59 INFO  - jmeter.threads.JMeterThread: Stop Thread seen: org.apache.jorphan.util.JMeterStopThreadException: End of
2015/06/13 16:32:59 INFO  - jmeter.threads.JMeterThread: Thread finished: Thread Group 1-1
2015/06/13 16:32:59 INFO  - jmeter.engine.StandardJMeterEngine: Notifying test listeners of end of test
2015/06/13 16:32:59 INFO  - jmeter.services.FileServer: Close: Practice 7.csv
2015/06/13 16:32:59 INFO  - jmeter.gui.util.JMeterMenuBar: setRunning(false,*local*)
```

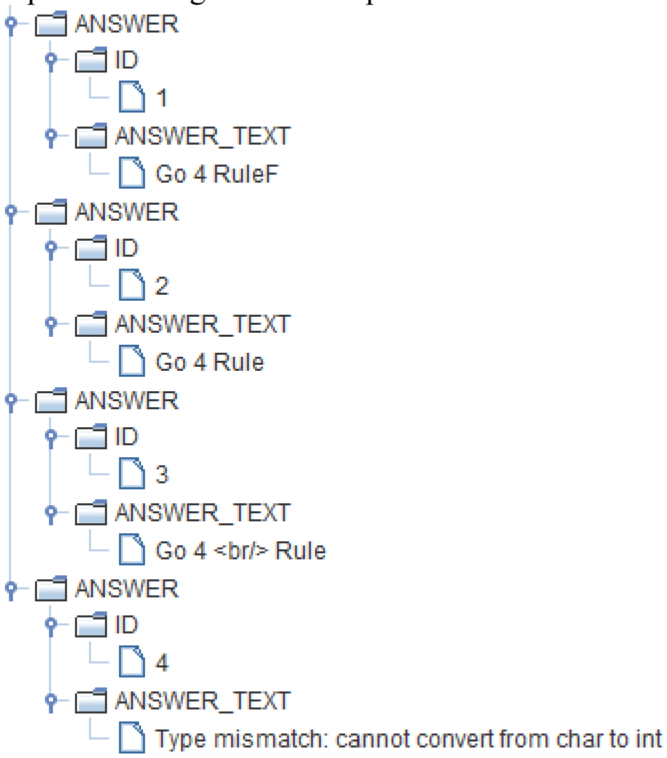
    3. There is a log file with current counter and email. The path is relative by the jmeter executable.

Parsing documents

Jmeter offers BeanShell console, which is almost as powerful as Java is. Though there are especially dedicated components to parse XML documents.

Practice 8

If a request to retrieve a question from a particular domain (Java, .Net or SQL) is put, our application returns that question along with all the possible answers.



```
ANSWER
    ID
        1
    ANSWER_TEXT
        Go 4 RuleF
ANSWER
    ID
        2
    ANSWER_TEXT
        Go 4 Rule
ANSWER
    ID
        3
    ANSWER_TEXT
        Go 4 <br/> Rule
ANSWER
    ID
        4
    ANSWER_TEXT
        Type mismatch: cannot convert from char to int
```

We will loop through these answers making a JDBC query against each of them.

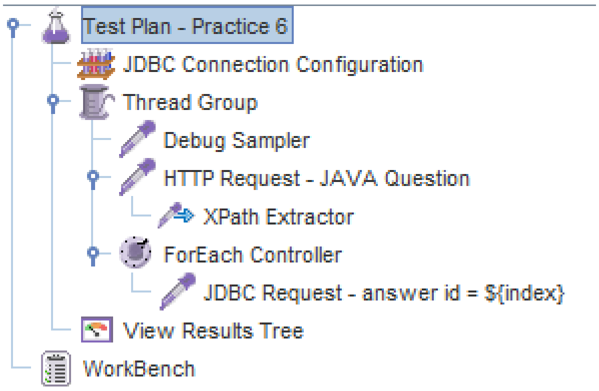
Solution 8

1. Create a new test plan, change its name to "Test Plan – Practice 8" and save it by the name of Practice_08.jmx.
2. Add (or copy from Test Plan – Practice 2) Config Element > JDBC Configuration to the Test Plan and configure it as follows:
    1. Variable Name: mysql
    2. Database URL: jdbc:mysql://127.0.0.1:13306/registrationsmobilization?user=myuser&password=mypass
    3. JDBC Driver Class: com.mysql.jdbc.Driver
    4. Username: myuser
    5. Password: mypass
3. Add Threads (Users) > Thread Group to the Test Plan.
4. Add (or copy from Test Plan – Practice 1) Sampler > HTTP Request.
    1. Server Name or IP: ${__machineIP()}
    2. Port Number: 18080
    3. Protocol [http]: HTTP
    4. Method: GET
    5. Content encoding: UTF-8
    6. Path: /registrationform/registrations/JAVA
5. Add Post Processors > XPath Extractor to the newly added HTTP Request.
    1. Set Reference Name to answerID.
    2. XPAth Query = /QUESTION/ANSWER/ID

    3. Default value set to NOT FOUND.
6. Add Logic Controller > ForEach Controller to the Thread Group.
   Set:
   1. Input variable prefix to answerID.
   2. Start index for loop (exclusive) to 0.
   3. End index for loop (inclusive) to ${answerID_matchNr}.
   4. Output variable name to *index*.
   5. Make sure “Add “_” before number?” is active.
7. Add Sampler > JDBC Request to the ForEach Controller.
   1. Change its name to JDBC Request - answer id = ${index}.
   2. Set Variable name to mysql.
   3. Put in the query box:

SELECT * **FROM** ANSWERS **WHERE** ID = ${index};

8. Add View Result Tree to the Test Plan.
9. Optionally Add Sampler > Debug Sampler to the Thread Group.



The test plan should look similarly to the one above.

Assignment 4 (combines BeanShell Console and parsing documents)

Retrieve the actual text of one of the Java questions and store it to a file.
The actual text means value of the node QUESTION_TEXT. So in the file there should be for example text:

<strong> Which one of the following keywords allows to sort the query result? </strong> <br/> <ol> <li>GROUP BY</li> <li>ORDER FROM</li> <li>ASC BY</li> <li>ORDER BY</li> </ol>

Important is to use BeanShell Console to parse the XML document and store the text to a file.

Attachment 1

Practice 4 data

```
fname|lname|email.1@gft.com
fname|lname|email.2@gft.com
fname|lname|email.3@gft.com
fname|lname|email.4@gft.com
fname|lname|email.5@gft.com
fname|lname|email.6@gft.com
fname|lname|email.7@gft.com
fname|lname|email.8@gft.com
fname|lname|email.9@gft.com
fname|lname|email.10@gft.com
fname|lname|email.11@gft.com
fname|lname|email.12@gft.com
fname|lname|email.13@gft.com
fname|lname|email.14@gft.com
fname|lname|email.15@gft.com
fname|lname|email.16@gft.com
fname|lname|email.17@gft.com
fname|lname|email.18@gft.com
fname|lname|email.19@gft.com
fname|lname|email.20@gft.com
fname|lname|email.21@gft.com
fname|lname|email.22@gft.com
fname|lname|email.23@gft.com
fname|lname|email.24@gft.com
fname|lname|email.25@gft.com
fname|lname|email.26@gft.com
fname|lname|email.27@gft.com
fname|lname|email.28@gft.com
fname|lname|email.29@gft.com
fname|lname|email.30@gft.com
fname|lname|email.31@gft.com
fname|lname|email.32@gft.com
fname|lname|email.33@gft.com
fname|lname|email.34@gft.com
fname|lname|email.35@gft.com
fname|lname|email.36@gft.com
fname|lname|email.37@gft.com
fname|lname|email.38@gft.com
fname|lname|email.39@gft.com
fname|lname|email.40@gft.com
fname|lname|email.41@gft.com
fname|lname|email.42@gft.com
fname|lname|email.43@gft.com
fname|lname|email.44@gft.com
fname|lname|email.45@gft.com
fname|lname|email.46@gft.com
fname|lname|email.47@gft.com
fname|lname|email.48@gft.com
fname|lname|email.49@gft.com
fname|lname|email.50@gft.com
fname|lname|email.51@gft.com
fname|lname|email.52@gft.com
fname|lname|email.53@gft.com
fname|lname|email.54@gft.com
fname|lname|email.55@gft.com
fname|lname|email.56@gft.com
fname|lname|email.57@gft.com
fname|lname|email.58@gft.com
fname|lname|email.59@gft.com
fname|lname|email.60@gft.com
fname|lname|email.61@gft.com
fname|lname|email.62@gft.com
fname|lname|email.63@gft.com
fname|lname|email.64@gft.com
fname|lname|email.65@gft.com
fname|lname|email.66@gft.com
fname|lname|email.67@gft.com
fname|lname|email.68@gft.com
fname|lname|email.69@gft.com
fname|lname|email.70@gft.com
fname|lname|email.71@gft.com
fname|lname|email.72@gft.com
fname|lname|email.73@gft.com
fname|lname|email.74@gft.com
fname|lname|email.75@gft.com
fname|lname|email.76@gft.com
fname|lname|email.77@gft.com
fname|lname|email.78@gft.com
fname|lname|email.79@gft.com
fname|lname|email.80@gft.com
fname|lname|email.81@gft.com
fname|lname|email.82@gft.com
fname|lname|email.83@gft.com
fname|lname|email.84@gft.com
fname|lname|email.85@gft.com
fname|lname|email.86@gft.com
fname|lname|email.87@gft.com
fname|lname|email.88@gft.com
fname|lname|email.89@gft.com
fname|lname|email.90@gft.com
fname|lname|email.91@gft.com
fname|lname|email.92@gft.com
fname|lname|email.93@gft.com
fname|lname|email.94@gft.com
fname|lname|email.95@gft.com
fname|lname|email.96@gft.com
fname|lname|email.97@gft.com
fname|lname|email.98@gft.com
fname|lname|email.99@gft.com
fname|lname|email.100@gft.com
```