Run JMeter test plan from CMD:

For JMeter-ScrumTaskboard testcases-After Code Improvements, Login\_Add New Developer\_Delete Old Developer\_Logout testcases:

* **100 threads**:

D:\apache-jmeter-2.13\bin>jmeter -n -t Login-Add\_New\_Developer-Delete\_Old\_Developer-Logout-100\_threads.jmx -l Login-Add\_New\_Developer-Delete\_Old\_Developer-Logout-100\_threads.jtl

***Output from CMD*:**

Creating summariser <summary>

Created the tree successfully using Login-Add\_New\_Developer-Delete\_Old\_Developer-Logout-100\_threads.jmx

Starting the test @ Sat Mar 12 17:25:16 EET 2016 (1457796316464)

Waiting for possible shutdown message on port 4445

summary + 119 in 18s = 6.8/s Avg: 6997 Min: 50 Max: 8643 Err: 0 (0.00%) Active: 100 Started: 100 Finished: 0

summary + 333 in 26s = 13.0/s Avg: 7664 Min: 244 Max: 18441 Err: 54 (16.22%) Active: 100 Started: 100 Finished: 0

summary = 452 in 43.3s = 10.4/s Avg: 7488 Min: 50 Max: 18441 Err: 54 (11.95%)

summary + 194 in 30s = 6.5/s Avg: 15300 Min: 83 Max: 37799 Err: 148 (76.29%) Active: 100 Started: 100 Finished: 0

summary = 646 in 73.4s = 8.8/s Avg: 9834 Min: 50 Max: 37799 Err: 202 (31.27%)

summary + 287 in 30s = 9.6/s Avg: 8113 Min: 5 Max: 30047 Err: 121 (42.16%) Active: 89 Started: 100 Finished: 11

summary = 933 in 103s = 9.0/s Avg: 9305 Min: 5 Max: 37799 Err: 323 (34.62%)

summary + 167 in 10s = 17.0/s Avg: 7985 Min: 6 Max: 27722 Err: 77 (46.11%) Active: 0 Started: 100 Finished: 100

summary = 1100 in 113s = 9.7/s Avg: 9104 Min: 5 Max: 37799 Err: 400 (36.36%)

Tidying up ... @ Sat Mar 12 17:27:09 EET 2016 (1457796429948)

... end of run

***The report for this test case was generated in:***

D:\School\Master\dissertation-thesis\experiments\JMeter-ScrumTaskboard testcases-After Code Improvements\Login\_Add New Developer\_Delete Old Developer\_Logout testcases\100 threads\Login-Add\_New\_Developer-Delete\_Old\_Developer-Logout-100\_threads.jtl file

* Same error values as the ones obtained in Summary Report listener (when running from JMeter GUI) => no improvement
* **300 threads:**

D:\apache-jmeter-2.13\bin>jmeter -n -t Login-Add\_New\_Developer-Delete\_Old\_Developer-Logout-300\_threads.jmx -l Login-Add\_New\_Developer-Delete\_Old\_Developer-Logout-300\_threads.jtl

***Output from CMD*:**

Creating summariser <summary>

Created the tree successfully using Login-Add\_New\_Developer-Delete\_Old\_Developer-Logout-300\_threads.jmx

Starting the test @ Sat Mar 12 20:24:01 EET 2016 (1457807041938)

Waiting for possible shutdown message on port 4445

summary + 76 in 28s = 2.7/s Avg: 19843 Min: 2022 Max: 25883 Err: 42 (55.26%) Active: 300 Started: 300 Finished: 0

summary + 592 in 30.1s = 19.6/s Avg: 14754 Min: 4 Max: 57102 Err: 445 (75.17%) Active: 299 Started: 300 Finished: 1

summary = 668 in 58s = 11.5/s Avg: 15333 Min: 4 Max: 57102 Err: 487 (72.90%)

summary + 545 in 30s = 18.2/s Avg: 19637 Min: 4 Max: 77499 Err: 314 (57.61%) Active: 287 Started: 300 Finished: 13

summary = 1213 in 88s = 13.8/s Avg: 17267 Min: 4 Max: 77499 Err: 801 (66.03%)

summary + 302 in 30.1s = 10.0/s Avg: 20241 Min: 500 Max: 48827 Err: 144 (47.68%) Active: 283 Started: 300 Finished: 17

summary = 1515 in 118s = 12.8/s Avg: 17859 Min: 4 Max: 77499 Err: 945 (62.38%)

summary + 272 in 30s = 9.1/s Avg: 27097 Min: 7587 Max: 73874 Err: 135 (49.63%) Active: 270 Started: 300 Finished: 30

summary = 1787 in 148s = 12.1/s Avg: 19266 Min: 4 Max: 77499 Err: 1080 (60.44%)

summary + 290 in 30.1s = 9.6/s Avg: 37102 Min: 7484 Max: 98274 Err: 143 (49.31%) Active: 254 Started: 300 Finished: 46

summary = 2077 in 178s = 11.7/s Avg: 21756 Min: 4 Max: 98274 Err: 1223 (58.88%)

summary + 242 in 30s = 8.1/s Avg: 30657 Min: 9291 Max: 79455 Err: 119 (49.17%) Active: 232 Started: 300 Finished: 68

summary = 2319 in 208s = 11.2/s Avg: 22685 Min: 4 Max: 98274 Err: 1342 (57.87%)

summary + 234 in 30.2s = 7.7/s Avg: 28562 Min: 12082 Max: 70408 Err: 131 (55.98%) Active: 211 Started: 300 Finished: 89

summary = 2553 in 238s = 10.7/s Avg: 23224 Min: 4 Max: 98274 Err: 1473 (57.70%)

summary + 276 in 30s = 9.3/s Avg: 23380 Min: 7 Max: 64572 Err: 153 (55.43%) Active: 171 Started: 300 Finished: 129

summary = 2829 in 268s = 10.6/s Avg: 23239 Min: 4 Max: 98274 Err: 1626 (57.48%)

summary + 351 in 30.1s = 11.7/s Avg: 16112 Min: 14 Max: 51011 Err: 143 (40.74%) Active: 57 Started: 300 Finished: 243

summary = 3180 in 298s = 10.7/s Avg: 22452 Min: 4 Max: 98274 Err: 1769 (55.63%)

summary + 120 in 12s = 10.4/s Avg: 6591 Min: 6 Max: 26216 Err: 61 (50.83%) Active: 0 Started: 300 Finished: 300

summary = 3300 in 309s = 10.7/s Avg: 21875 Min: 4 Max: 98274 Err: 1830 (55.45%)

Tidying up ... @ Sat Mar 12 20:29:11 EET 2016 (1457807351628)

... end of run

***The report for this test case was generated in:***

D:\School\Master\dissertation-thesis\experiments\JMeter-ScrumTaskboard testcases-After Code Improvements\Login\_Add New Developer\_Delete Old Developer\_Logout testcases\300 threads\Login-Add\_New\_Developer-Delete\_Old\_Developer-Logout-300\_threads.jtl file

* Greater error values (55.45%) than the ones obtained in Summary Report listener (54.55%) (when running from JMeter GUI) => no improvement, actually worst results were obtained when running JMeter in Non-GUI mode
* **500 threads:**

D:\apache-jmeter-2.13\bin>jmeter -n -t Login-Add\_New\_Developer-Delete\_Old\_Developer-Logout-500\_threads.jmx -l Login-Add\_New\_Developer-Delete\_Old\_Developer-Logout-500\_threads.jtl

***Output from CMD*:**

Creating summariser <summary>

Created the tree successfully using Login-Add\_New\_Developer-Delete\_Old\_Developer-Logout-500\_threads.jmx

Starting the test @ Sat Mar 12 20:38:47 EET 2016 (1457807927459)

Waiting for possible shutdown message on port 4445

summary + 533 in 12.3s = 43.4/s Avg: 2803 Min: 1994 Max: 10493 Err: 532 (99.81%) Active: 500 Started: 500 Finished: 0

summary + 1432 in 30s = 47.8/s Avg: 3870 Min: 344 Max: 39716 Err: 1421 (99.23%) Active: 423 Started: 500 Finished: 77

summary = 1965 in 42.3s = 46.5/s Avg: 3580 Min: 344 Max: 39716 Err: 1953 (99.39%)

summary + 880 in 31.3s = 28.1/s Avg: 13418 Min: 7 Max: 71453 Err: 809 (91.93%) Active: 380 Started: 500 Finished: 120

summary = 2845 in 74s = 38.7/s Avg: 6623 Min: 7 Max: 71453 Err: 2762 (97.08%)

summary + 466 in 29s = 16.3/s Avg: 29088 Min: 6 Max: 97769 Err: 376 (80.69%) Active: 352 Started: 500 Finished: 148

summary = 3311 in 102s = 32.4/s Avg: 9785 Min: 6 Max: 97769 Err: 3138 (94.77%)

summary + 350 in 30.5s = 11.5/s Avg: 29614 Min: 62 Max: 118223 Err: 234 (66.86%) Active: 320 Started: 500 Finished: 180

summary = 3661 in 133s = 27.6/s Avg: 11681 Min: 6 Max: 118223 Err: 3372 (92.11%)

summary + 354 in 30s = 11.8/s Avg: 23026 Min: 612 Max: 113734 Err: 185 (52.26%) Active: 280 Started: 500 Finished: 220

summary = 4015 in 163s = 24.7/s Avg: 12681 Min: 6 Max: 118223 Err: 3557 (88.59%)

summary + 266 in 30s = 8.9/s Avg: 39949 Min: 9490 Max: 106930 Err: 140 (52.63%) Active: 257 Started: 500 Finished: 243

summary = 4281 in 193s = 22.2/s Avg: 14375 Min: 6 Max: 118223 Err: 3697 (86.36%)

summary + 249 in 30s = 8.3/s Avg: 32390 Min: 260 Max: 113515 Err: 126 (50.60%) Active: 220 Started: 500 Finished: 280

summary = 4530 in 223s = 20.3/s Avg: 15366 Min: 6 Max: 118223 Err: 3823 (84.39%)

summary + 225 in 30.1s = 7.5/s Avg: 29330 Min: 26 Max: 98744 Err: 137 (60.89%) Active: 193 Started: 500 Finished: 307

summary = 4755 in 253s = 18.8/s Avg: 16026 Min: 6 Max: 118223 Err: 3960 (83.28%)

summary + 244 in 32.1s = 7.6/s Avg: 29929 Min: 9386 Max: 66046 Err: 140 (57.38%) Active: 154 Started: 500 Finished: 346

summary = 4999 in 285s = 17.6/s Avg: 16705 Min: 6 Max: 118223 Err: 4100 (82.02%)

summary + 281 in 27.5s = 10.2/s Avg: 16046 Min: 7 Max: 50477 Err: 119 (42.35%) Active: 85 Started: 500 Finished: 415

summary = 5280 in 312s = 16.9/s Avg: 16670 Min: 6 Max: 118223 Err: 4219 (79.91%)

summary + 220 in 23s = 9.6/s Avg: 8202 Min: 3 Max: 33178 Err: 109 (49.55%) Active: 0 Started: 500 Finished: 500

summary = 5500 in 335s = 16.4/s Avg: 16331 Min: 3 Max: 118223 Err: 4328 (78.69%)

Tidying up ... @ Sat Mar 12 20:44:22 EET 2016 (1457808262979)

... end of run

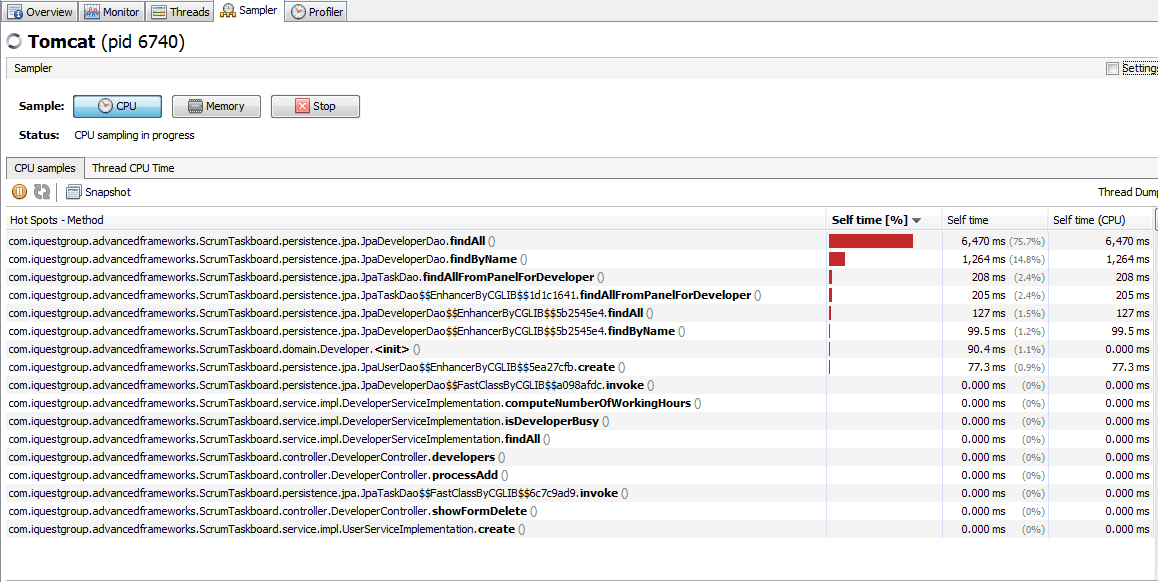
***The report for this test case was generated in:***

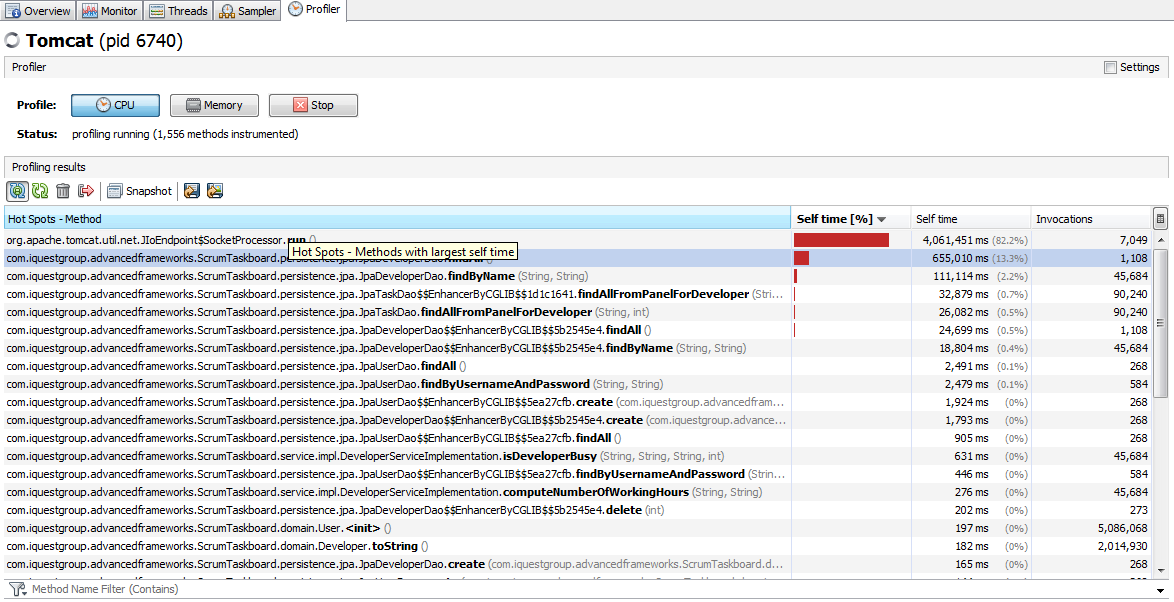
D:\School\Master\dissertation-thesis\experiments\JMeter-ScrumTaskboard testcases-After Code Improvements\Login\_Add New Developer\_Delete Old Developer\_Logout testcases\500 threads\Login-Add\_New\_Developer-Delete\_Old\_Developer-Logout-500\_threads.jtl file

* Greater error values (78.69%) than the ones obtained in Summary Report listener (77.58%) (when running from JMeter GUI) => no improvement, actually worst results were obtained when running JMeter in Non-GUI mode

Conclusions after analyzing VisualVM results:

* When profiling/sampling different application use-cases, the main CPU bottleneck is caused by methods from the persistence layer (when querying the database) => further improvements should be consider on the database level





DB improvements (how to explain/debug SQL queries)

<http://www.ducea.com/2006/11/06/identifying-mysql-slow-queries/>

<https://dev.mysql.com/doc/refman/5.5/en/execution-plan-information.html>

* in C:\Program Files\MySQL\MySQL Server 5.7\bin :

mysql –u root –p

apoi parola : root

* log-urile sunt in : C:\ProgramData\MySQL\MySQL Server 5.7\Data

EXPLAIN SELECT \* FROM Developer;

* Problema e apelul repetat de metode din showDevelopers.jsp

EXPLAIN EXTENDED SELECT \* from Task t where t.status like 'not-taken';

* Am adaugat index pt status pe field in clasa de Spring entity Task:

@Column(name="status")

@Index(name="statusIDX")

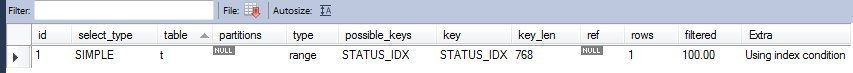
**private** String status;

dar MySql-ul nu ia in considerare acest Index la rularea query-ului

* Am adaugat indexul si din MySql pe tabela task :

**ALTER TABLE `scrum`.`task`**

**ADD INDEX `STATUS\_IDX` (`status` ASC);**



* Indexul pt status este folosit acum
* In fisierul din MySQl (\*-slow.log) apar mai putine aparitii ale query-ului:

SELECT \* from Task t where t.status like ‘…’; dupa adaugarea index-ului

EXPLAIN EXTENDED SELECT \* FROM Developer d WHERE d.firstName = 'miha' and d.lastName = 'man';

* Am adaugat composite index pt field-urile firstName and lastName in clasa de Spring entity Developer:

@Entity

@Table(name="developer", uniqueConstraints={@UniqueConstraint(columnNames={"firstName","lastName"})})

**public** **class** Developer {

@Index(name="firstNameIDX")

@Column(name="firstName")

**private** String firstName;

@Index(name="lastNameIDX")

@Column(name="lastName")

**private** String lastName;

**…**

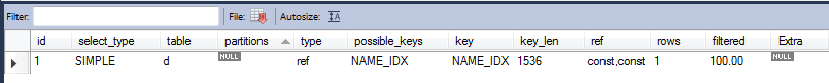
}

dar MySql-ul nu ia in considerare acest Index la rularea query-ului

* Am adaugat indexul si din MySql pe tabela developer :

**ALTER TABLE `scrum`.`developer`**

**ADD INDEX `NAME\_IDX` (`firstName` ASC, `lastName` ASC);**

****

* Indexul compus pt cautarea developerilor dupa firstName si lastName este folosit acum
* In fisierul din MySQl (\*-slow.log) nu mai apare query-ul:

SELECT \* FROM Developer d WHERE d.firstName = ‘…’ and d.lastName = ‘…’; dupa adaugarea index-ului si in JMeter apare error code: 500 – Internal Server Error

* Adaugarea unui index pt un coloana apartinand unei tabele din Java (pe clasa de model) se foloseste la rularea SQL query-urilor doar daca in persistence.xml avem:

**<property name="hibernate.hbm2ddl.auto" value="create"/>**

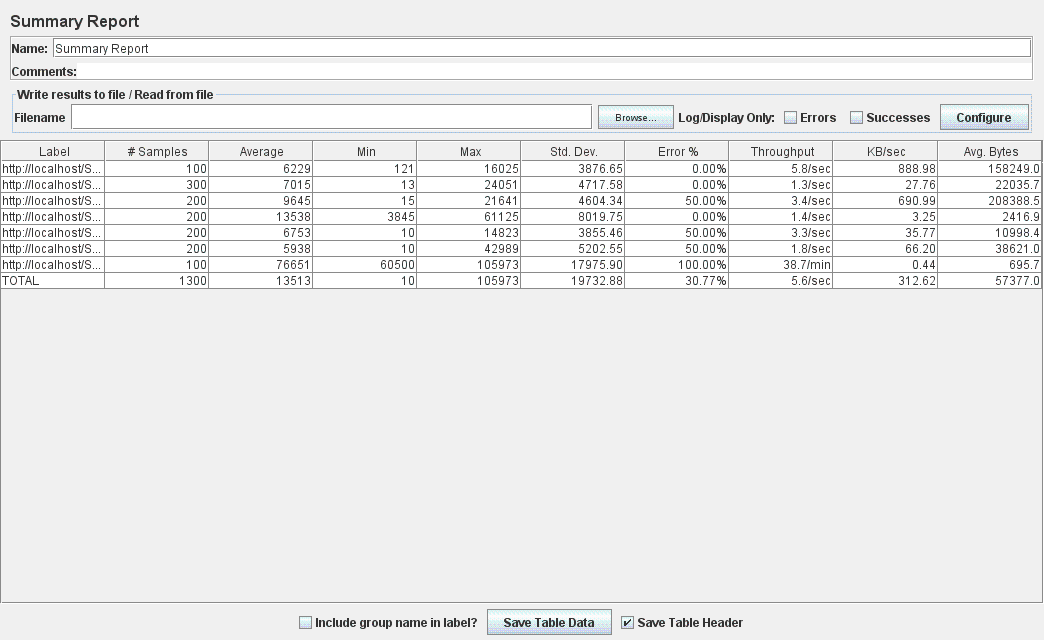
ceea ce presupune re-crearea DB-ului la fiecare restart de server.

Dupa o prima generare a DB-ului, cu indexul adaugat din Java, am schimbat valoarea acestei proprietati in update pt a nu pierde din nou datele din DB:

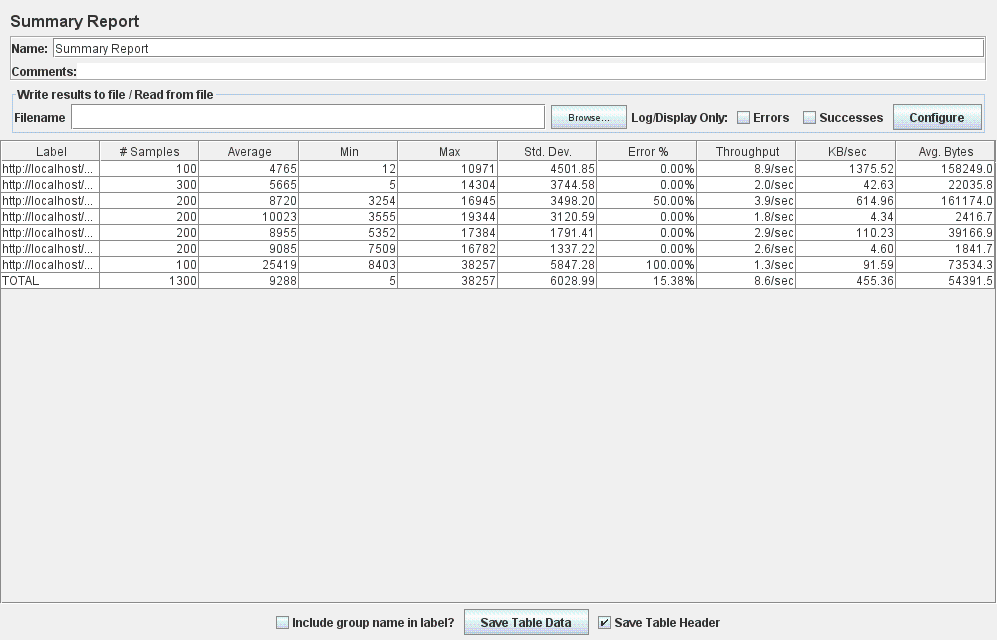
**<property name="hibernate.hbm2ddl.auto" value="update"/>**

Testul improvement-ului adus de adaugarea index-ului pe coloana status din tabela Task a fost realizat pt **Login\_Add New Developer\_Delete Old Developer\_Logout** testcase, cu un numar de 100 thread-uri concurente.

* Astfel inainte de adaugarea index-ului, eroarea era de 30.77 %:



* Dupa adaugarea index-ului, eroarea este de 15.38%:



Aceste rezultate au obtinute din JMeter. De asemenea, la rularea testcase-ului, folosind indexul adaugat, numarul query-urilor de tipul ***SELECT \* from Task t where t.status like ‘…’;*** din fiserul de MySql **\*-slow.log**a fost redus semnificativ.

* Indexul a adus performance improvement de 50% (de la 30… % la 15…%)

Next: de adaugat succesiv indecsi pt field-uri din diverse tabele (entitati) si de comparat performanta adusa (sau nu) de adaugarea acelui index. Aceste teste se vor face folosind JMeter. Daca, intr-adevar, se dovedeste o imbunatatire a performantei aduse de adaugarea indexului, acesta se va pastra, altfel nu.

Pentru test case-ul: **Login\_Skill Upgrades for developer\_Logout**, slow query-urile logate erau pentru a gasi un user dupa username si password. Astfel am adaugat index din cod java pe atributul username al entitatii User:

@NotEmpty

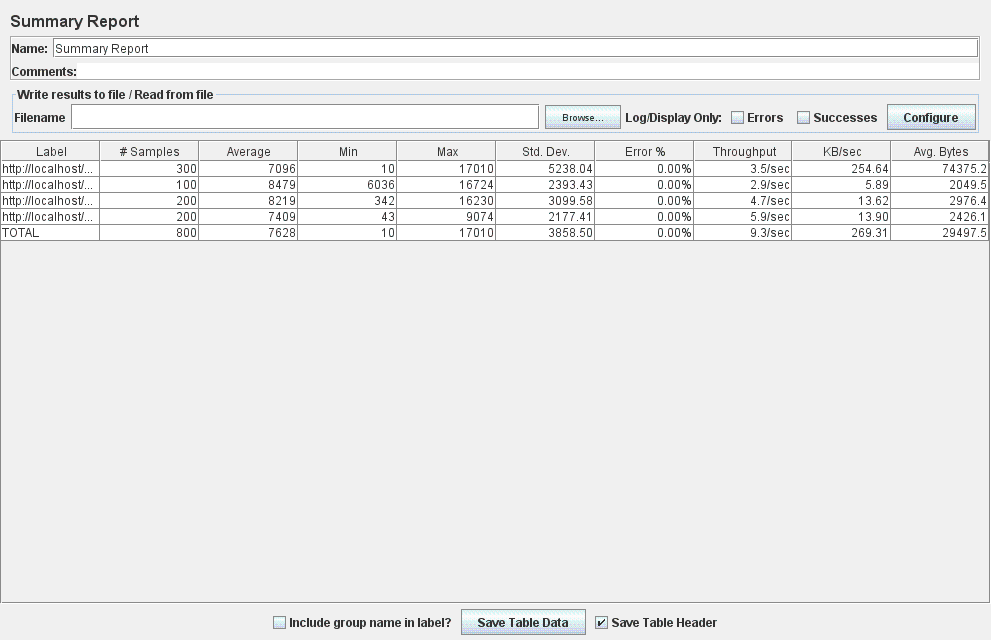
@Column(name="username")

@Index(name = "usernameIDX")

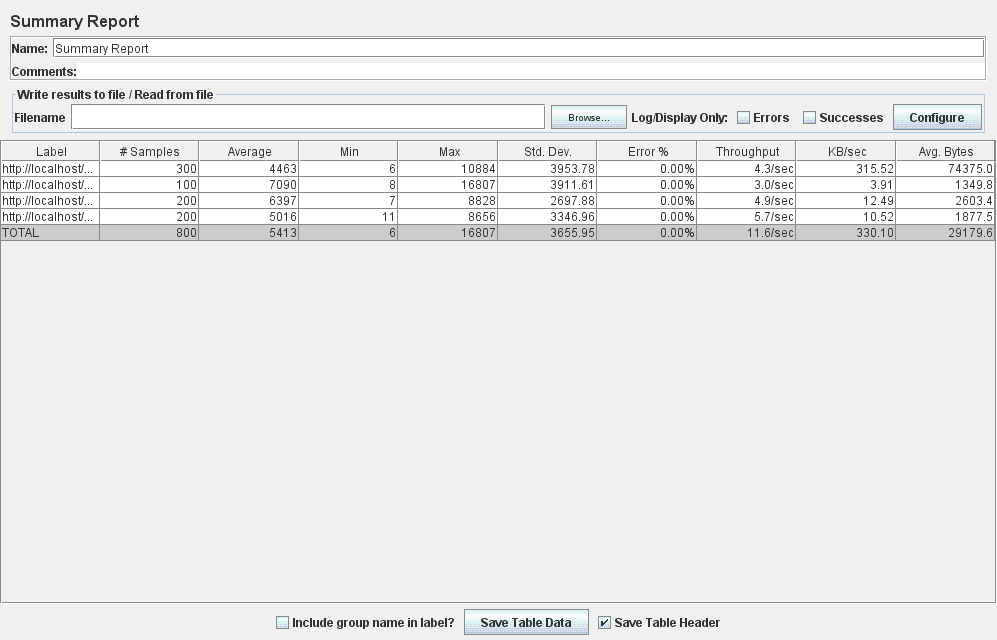
**private** String username;

Dupa adaugarea acestui index, query-ul logat ca fiind slow nu mai apare deloc in loguri. Acest lucru atesta o imbunatatire a performantei adusa de index-ul adaugat pe atributul username al entitatii User. Mai mult, comparand rezultatele oferite de JMeter, inainte si dupa adaugarea index-ului, putem concluziona urmatoarele:

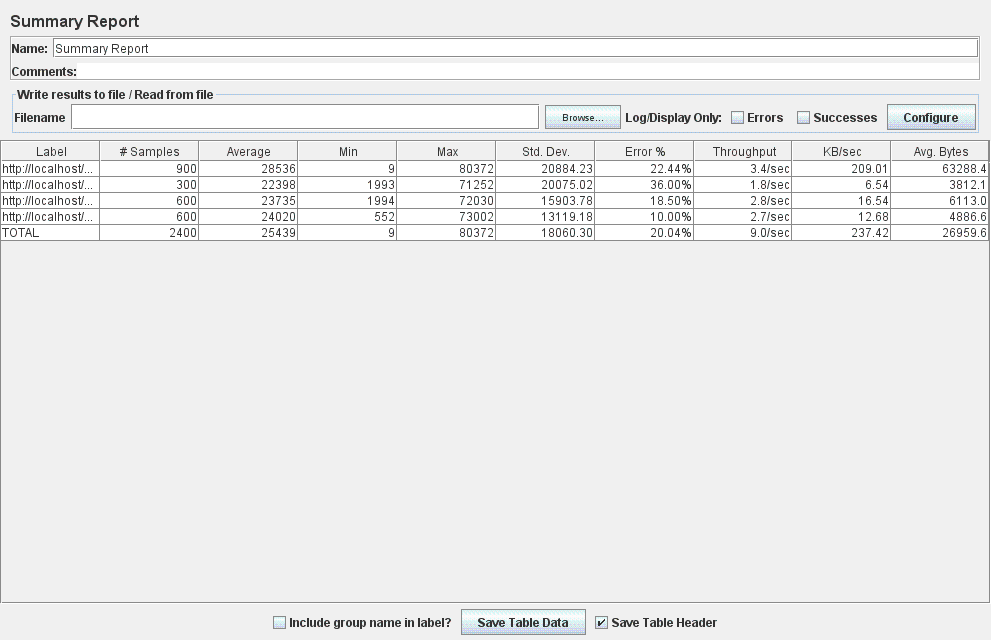
* Inainte de adaugarea index-ului, pentru un numar de 100 thread-uri (utilizatori) concurente, eroarea rezultata din rularea test case-ului corespunzator use case-ului **Login\_Skill Upgrades for developer\_Logout**, era de **0%** :



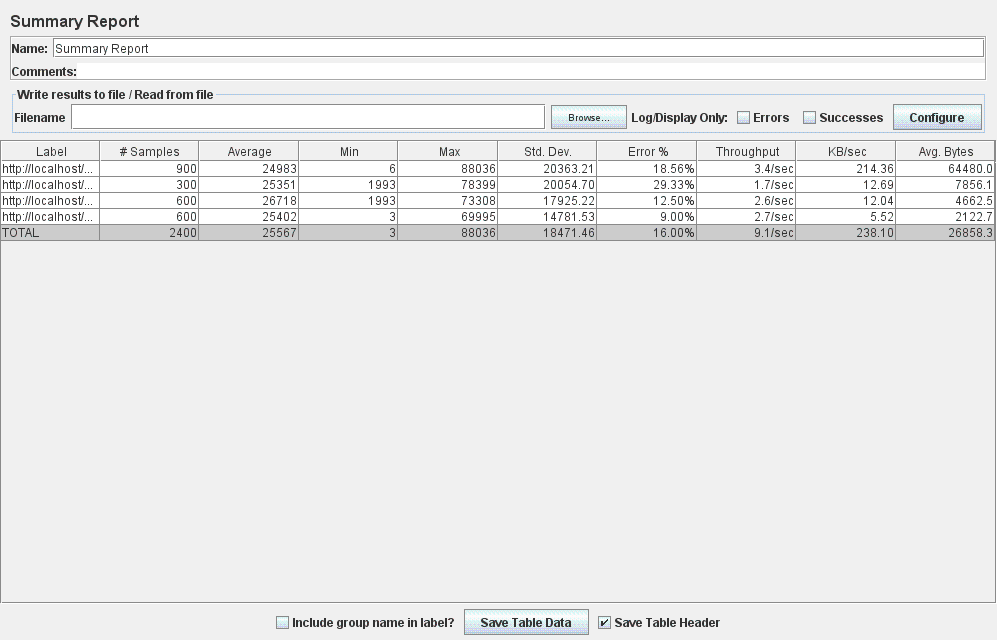
* Dupa adaugarea index-ului, pentru un numar de 100 thread-uri (utilizatori) concurente, eroarea rezultata din rularea test case-ului corespunzator use case-ului **Login\_Skill Upgrades for developer\_Logout**, este tot de **0%** :



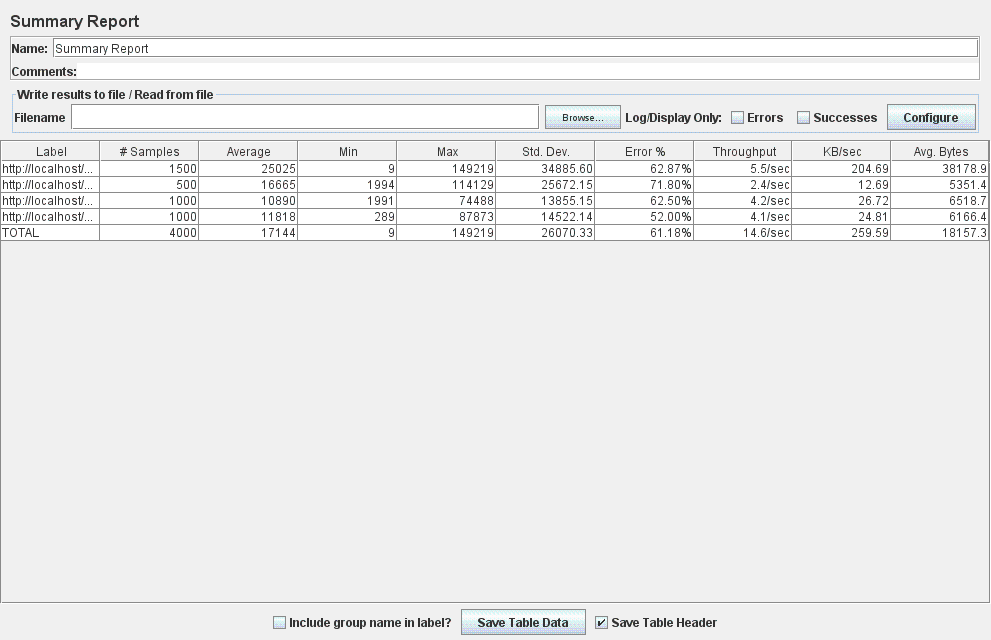
* Inainte de adaugarea index-ului, pentru un numar de 300 thread-uri (utilizatori) concurente, eroarea rezultata din rularea test case-ului corespunzator use case-ului **Login\_Skill Upgrades for developer\_Logout**, era de **20.04**% :



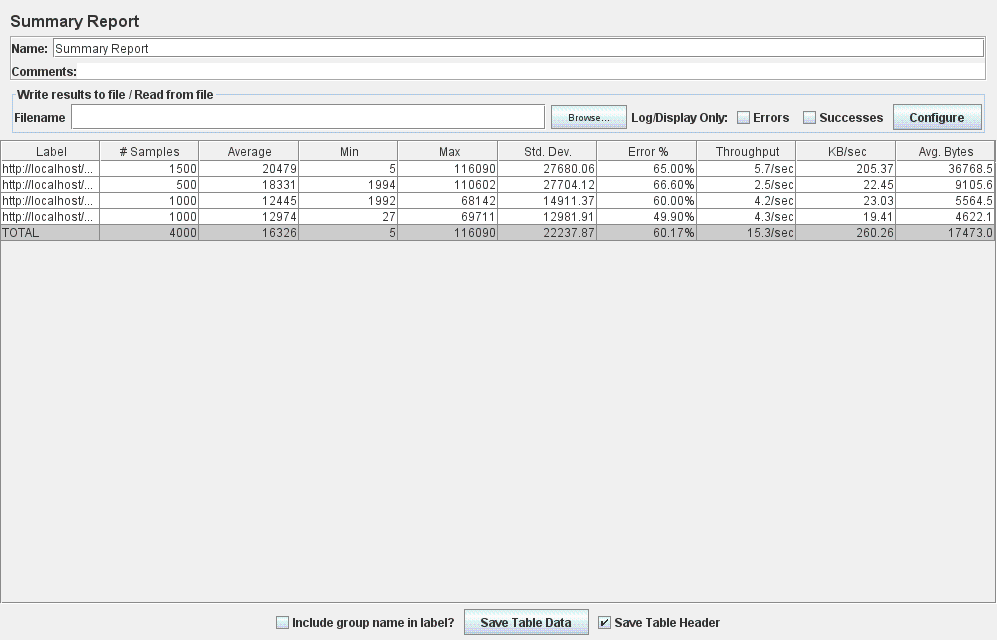
* Dupa adaugarea index-ului, pentru un numar de 300 thread-uri (utilizatori) concurente, eroarea rezultata din rularea test case-ului corespunzator use case-ului **Login\_Skill Upgrades for developer\_Logout**, este de **16.00%** :



* Inainte de adaugarea index-ului, pentru un numar de 500 thread-uri (utilizatori) concurente, eroarea rezultata din rularea test case-ului corespunzator use case-ului **Login\_Skill Upgrades for developer\_Logout**, era de **61.18**% :



* Dupa adaugarea index-ului, pentru un numar de 500 thread-uri (utilizatori) concurente, eroarea rezultata din rularea test case-ului corespunzator use case-ului **Login\_Skill Upgrades for developer\_Logout**, este de **60.17%** :



Concluziile care se pot trage din aceste rezultate:

* Performanta de 0% eroare pentru 100 utilizatori executand simultan use case-ul este pastrata si dupa adaugarea index-ului pentru username.
* In cazul a 300 utilizatori excutand simultan use case-ul, eroarea este redusa de la 20.04% la 16.00% dupa adaugara index-ului pe atributul username al entitatii User.
* Considerand 500 utilizatori excutand simultan use case-ul, eroarea este redusa de la 61.18% la 60.17% dupa adaugara index-ului pe atributul username al entitatii User.

In concluzie, in toate cele 3 cazuri, performanta a fost pastrata sau imbunatatita dupa adaugarea acestui index.

Cum query-ul initial care aparea in \*-slow.log , cautarea user-ului se facea filtrandu-se dupa atributele username si password, vom considera stergerea index-ului pt username si vom efectua aceleasi masurari dupa adaugarea unui index pe atributul password.

Astfel am sters index-ul adaugat pentru atributul username si am creat unul (tot din cod java) pe atributul password al entitatii User:

@NotEmpty

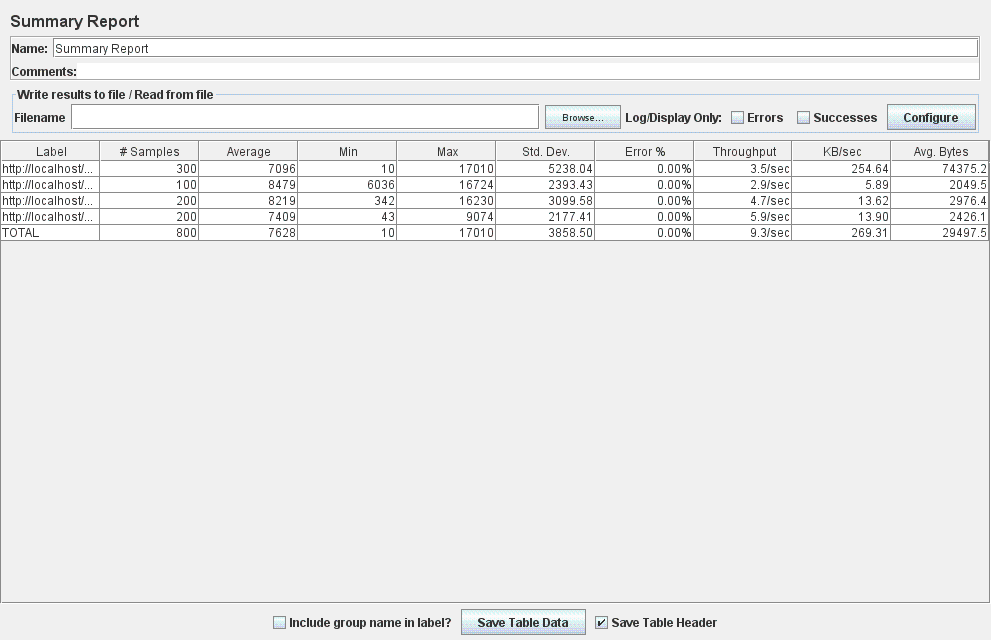
@Column(name="password")

@Index(name = "passwordIDX")

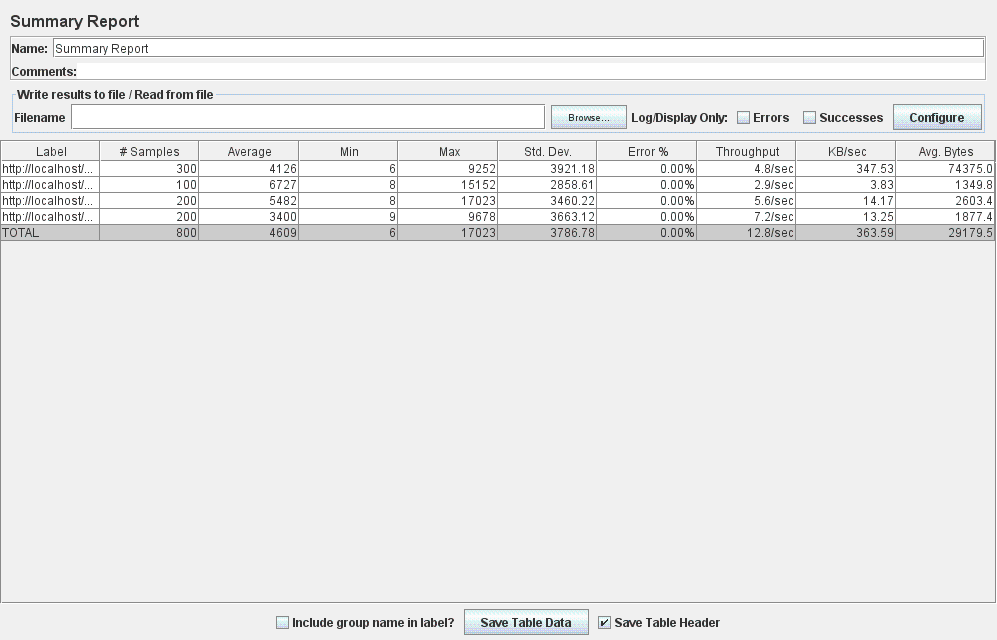
**private** String password;

Dupa adaugarea acestui index, query-ul logat ca fiind slow nu mai apare deloc in loguri. Acest lucru atesta o imbunatatire a performantei adusa de index-ul adaugat pe atributul password al entitatii User. Acest behavior s-a intalnit si la adaugarea index-ului pe atributul username. Mai mult, comparand rezultatele oferite de JMeter, inainte si dupa adaugarea index-ului, putem concluziona urmatoarele:

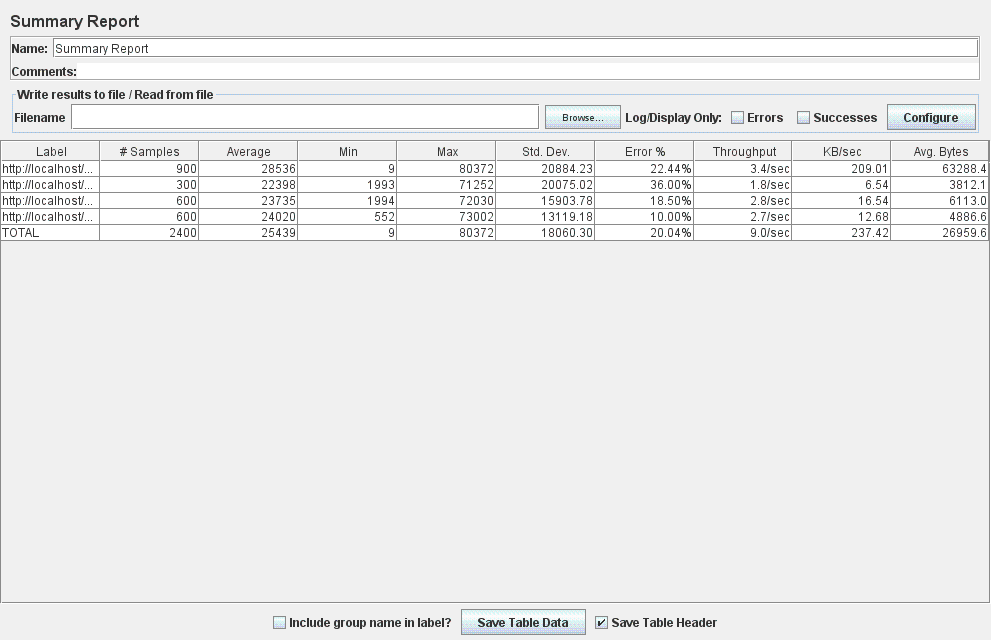
* Inainte de adaugarea index-ului, pentru un numar de 100 thread-uri (utilizatori) concurente, eroarea rezultata din rularea test case-ului corespunzator use case-ului **Login\_Skill Upgrades for developer\_Logout**, era de **0%** :



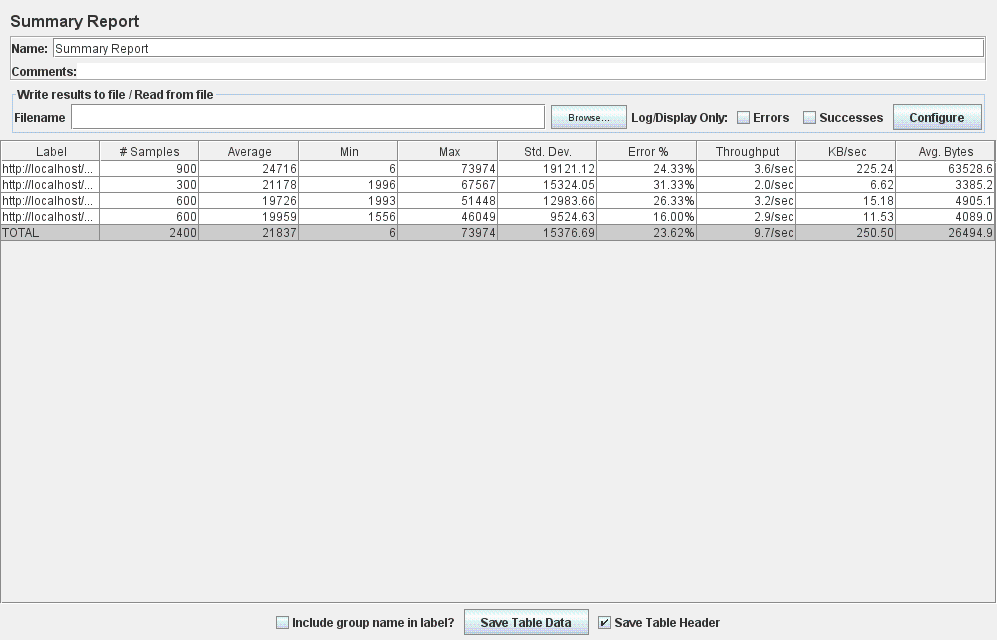
* Dupa adaugarea index-ului, pentru un numar de 100 thread-uri (utilizatori) concurente, eroarea rezultata din rularea test case-ului corespunzator use case-ului **Login\_Skill Upgrades for developer\_Logout**, este tot de **0%** :



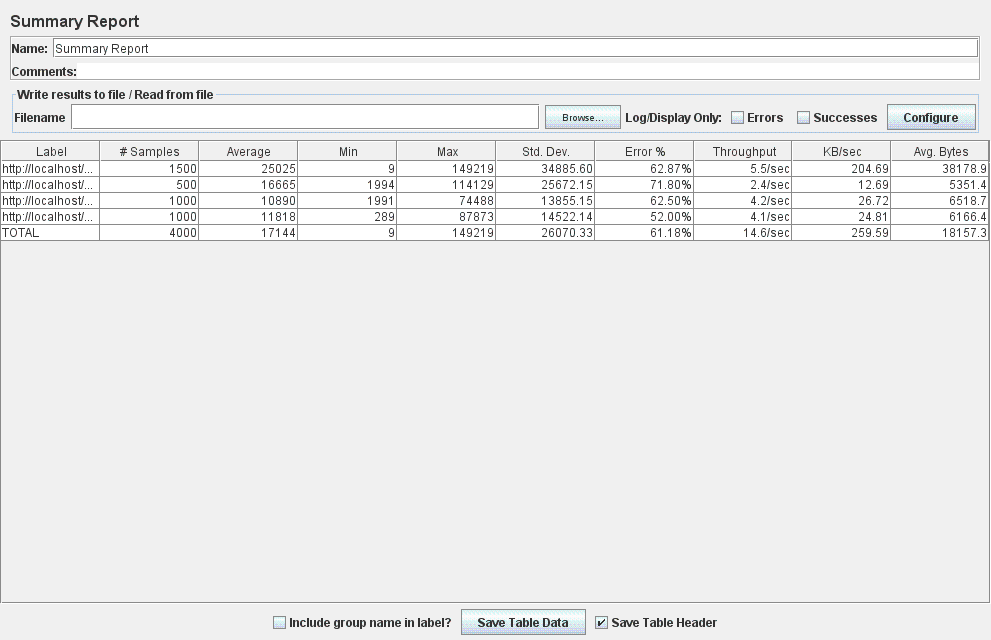
* Inainte de adaugarea index-ului, pentru un numar de 300 thread-uri (utilizatori) concurente, eroarea rezultata din rularea test case-ului corespunzator use case-ului **Login\_Skill Upgrades for developer\_Logout**, era de **20.04**% :



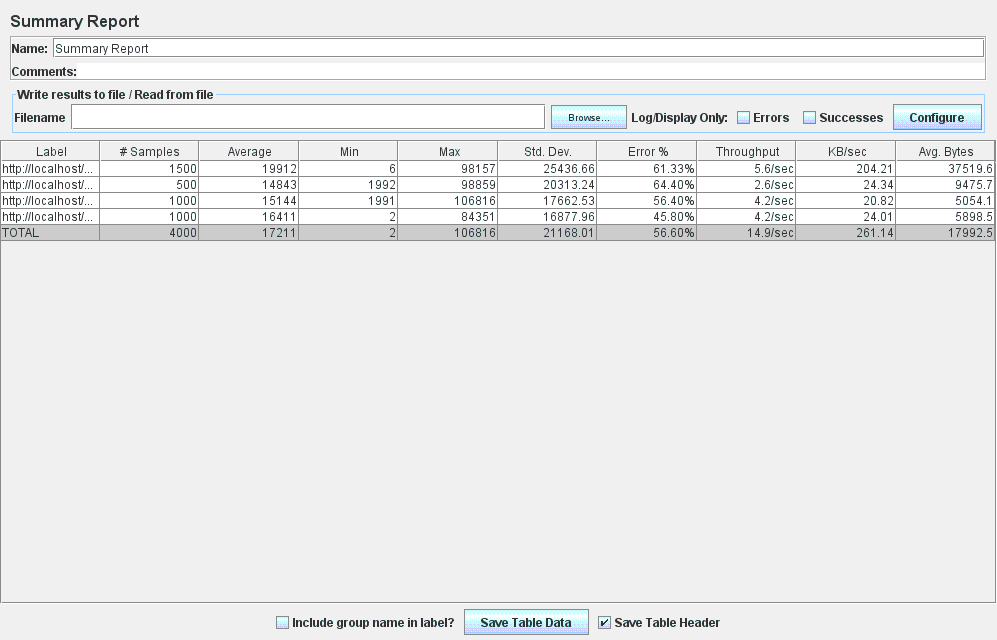
* Dupa adaugarea index-ului, pentru un numar de 300 thread-uri (utilizatori) concurente, eroarea rezultata din rularea test case-ului corespunzator use case-ului **Login\_Skill Upgrades for developer\_Logout**, este de **23.62%** :



* Inainte de adaugarea index-ului, pentru un numar de 500 thread-uri (utilizatori) concurente, eroarea rezultata din rularea test case-ului corespunzator use case-ului **Login\_Skill Upgrades for developer\_Logout**, era de **61.18**% :



* Dupa adaugarea index-ului, pentru un numar de 500 thread-uri (utilizatori) concurente, eroarea rezultata din rularea test case-ului corespunzator use case-ului **Login\_Skill Upgrades for developer\_Logout**, este de **56.60%** :



Concluziile care se pot trage din aceste rezultate:

* Performanta de 0% eroare pentru 100 utilizatori executand simultan use case-ul este pastrata si dupa adaugarea index-ului pentru password.
* In cazul a 300 utilizatori excutand simultan use case-ul, eroarea este marita de la 20.04% la 23.62% dupa adaugara index-ului pe atributul password al entitatii User.
* Considerand 500 utilizatori excutand simultan use case-ul, eroarea este redusa de la 61.18% la 56.60% dupa adaugara index-ului pe atributul password al entitatii User.

In concluzie, index-ul adaugat pe atributul password manifesta comportamente diferite pentru un numar diferit de ultizatori simultani al aceluiasi use case. Mai precis, putem deduce o imbunatatire a performantei pentru un numar de 100 si 500 thread-uri concurente, insa o degradare a acesteia la executia simultana a use case-ului de catre 300 utilizatori.

Cum query-ul initial care aparea in \*-slow.log , cautarea user-ului se facea filtrandu-se dupa atributele username si password, vom considera pastrarea index-ului pentru atributul password si vom adauga cel creat initial pentru atributul username, efectuand apoi aceleasi masurari in contextul utilizarii a doi indecsi separati pentru atributele folosite in clauza WHERE al query-ului SQL de cautare a user-ului in baza de date.

Astfel am pastrat index-ul pentru atributul password si l-am adaugat pe cel corespunzator atributului username password al entitatii User:

@NotEmpty

@Column(name="username")

@Index(name = "usernameIDX")

**private** String username;

@NotEmpty

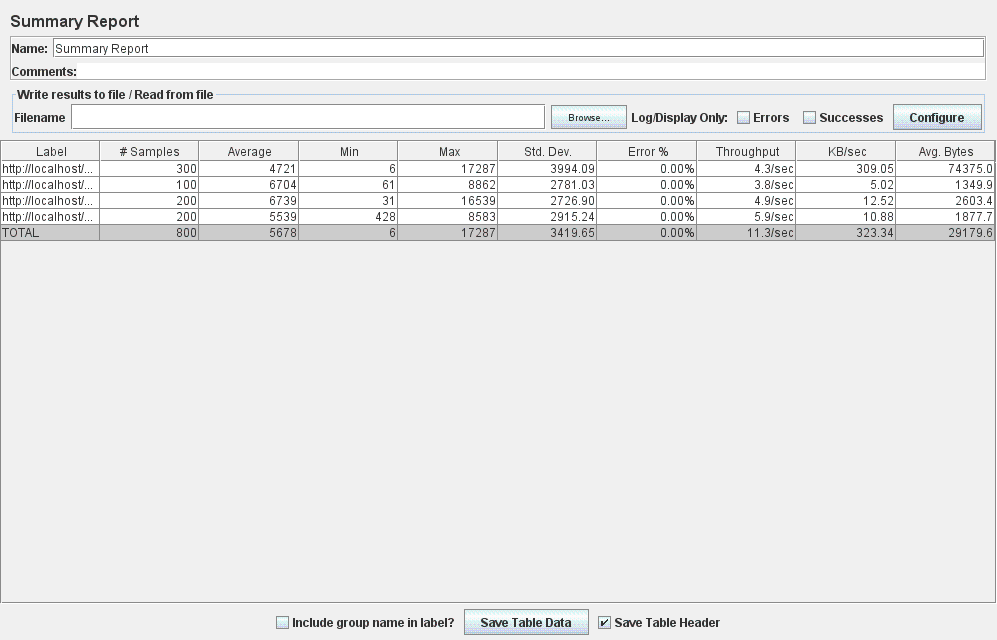
@Column(name="password")

@Index(name = "passwordIDX")

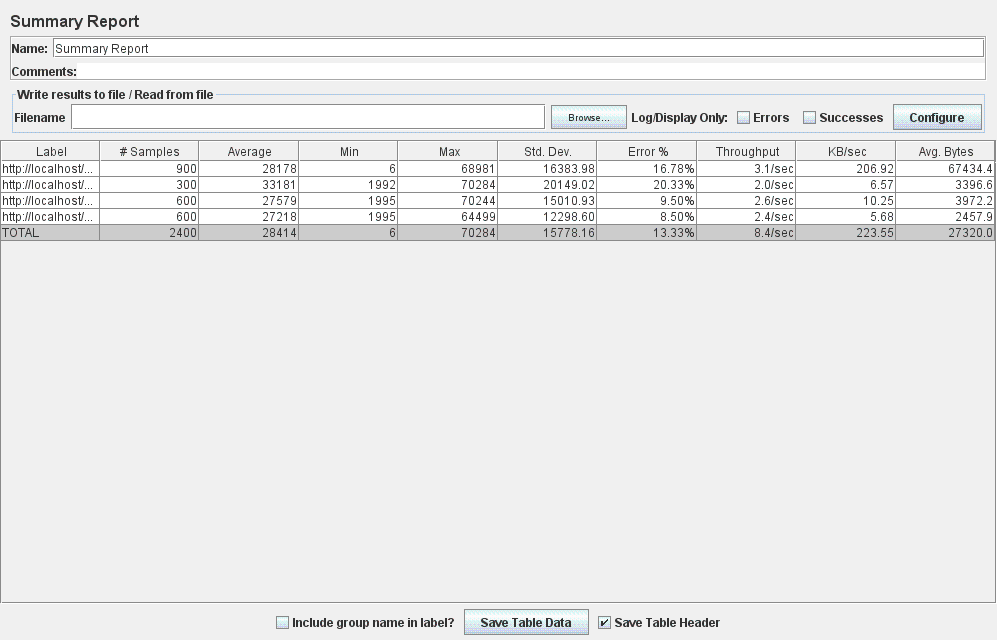
**private** String password;

La executia use case-ului folosind acesti doi indecsi, query-ul logat ca fiind slow nu mai apare deloc in loguri. Acest lucru atesta o imbunatatire a performantei adusa de indecsi adaugati pe atributele username si password al entitatii User. Acest behavior s-a intalnit si la utilizarea separata a indecsilor (initial pentru atributul username si apoi pentru password). In continuare, vom prezenta rezultatele obtinute din JMeter la folosirea a doi indecsi pentru use case-ul **Login\_Skill Upgrades for developer\_Logout**, facand apoi o comparatie a metricilor obtinute in folosirea a cate unuia dintre indecsi si respectiv al ambilor:

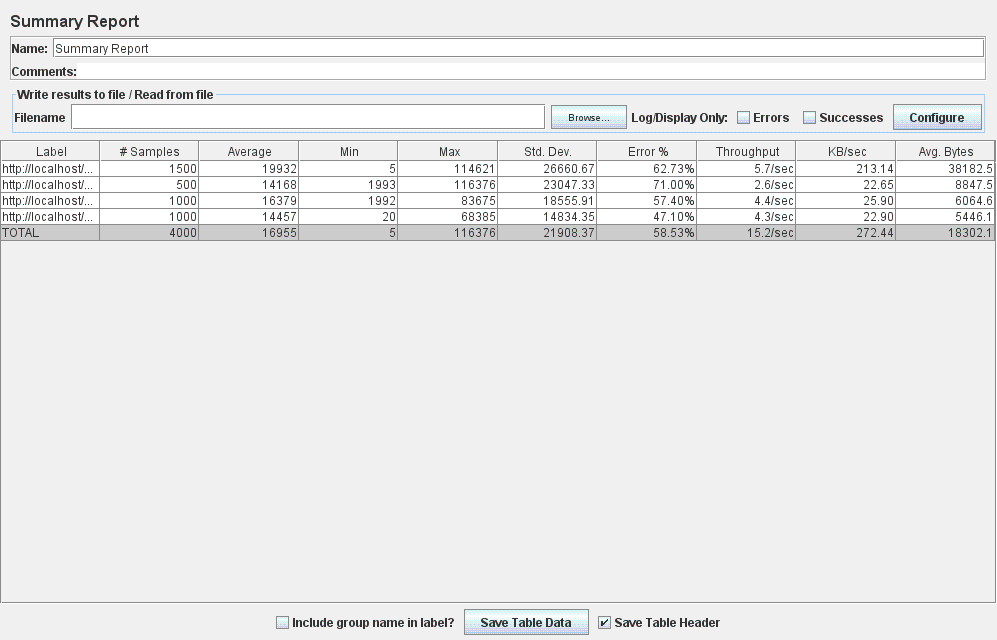
* Dupa adaugarea celor doi indecsi, pentru un numar de 100 thread-uri (utilizatori) concurente, eroarea rezultata din rularea test case-ului corespunzator use case-ului **Login\_Skill Upgrades for developer\_Logout**, este tot de **0%,** asemenea ca si in celelalte cazuri (fara index sau cu folosirea a cate unuia dintre indecsi) :



* Dupa adaugarea celor doi indecsi, pentru un numar de 300 thread-uri (utilizatori) concurente, eroarea rezultata din rularea test case-ului corespunzator use case-ului **Login\_Skill Upgrades for developer\_Logout**, este de **13.33%** :



* Dupa adaugarea celor doi indecsi, pentru un numar de 500 thread-uri (utilizatori) concurente, eroarea rezultata din rularea test case-ului corespunzator use case-ului **Login\_Skill Upgrades for developer\_Logout**, este de **58.53%** :



Concluziile care se pot trage din aceste rezultate:

* Performanta de 0% eroare pentru 100 utilizatori executand simultan use case-ul este pastrata si dupa adaugarea celor doi indecsi (pentru username si password).
* In cazul a 300 utilizatori excutand simultan use case-ul, eroare este de 13.33% dupa adaugarea celor doi indecsi (pentru username si password).
* Considerand 500 utilizatori excutand simultan use case-ul, eroarea este de 58.53% dupa adaugarea celor doi indecsi (pentru username si password).

In concluzie, index-ul adaugat pe cele doua atribute aduce o imbunatatire a performantei in toate cele trei cazuri (atat fata de neutilizarea indecsilor, cat si fata de utilizarea separata a cate unuia dintre acestia).

Next: index compus din username si password

In continuare vom sterge cei doi indecsi separati (pt username si password) si vom crea unul compus din cele doua atribute ale entitaii User (username si password), efectuand apoi aceleasi masurari in contextul utilizarii a unui index compus din atributele folosite in clauza WHERE al query-ului SQL de cautare a user-ului in baza de date.

Indexul compus din atributele username si password se specifica pe entitatea de care apartin (User) in cod java:

@Entity

@Table(name="userList", uniqueConstraints=@UniqueConstraint(columnNames={"username", "password"}))

**public** **class** User {

…

@NotEmpty

@Column(name="username")

**private** String username;

@NotEmpty

@Column(name="password")

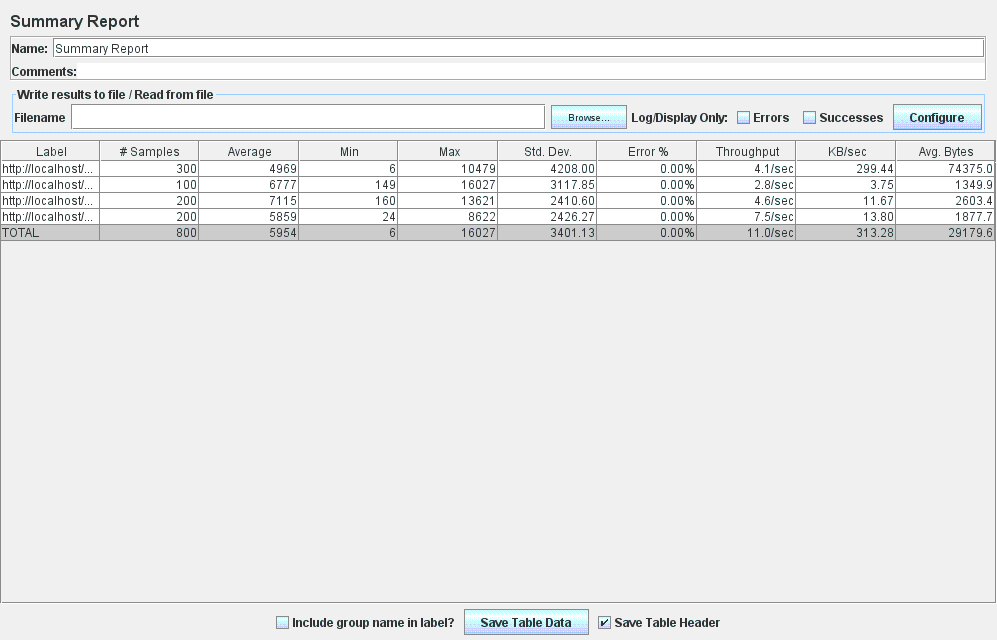
**private** String password;

…

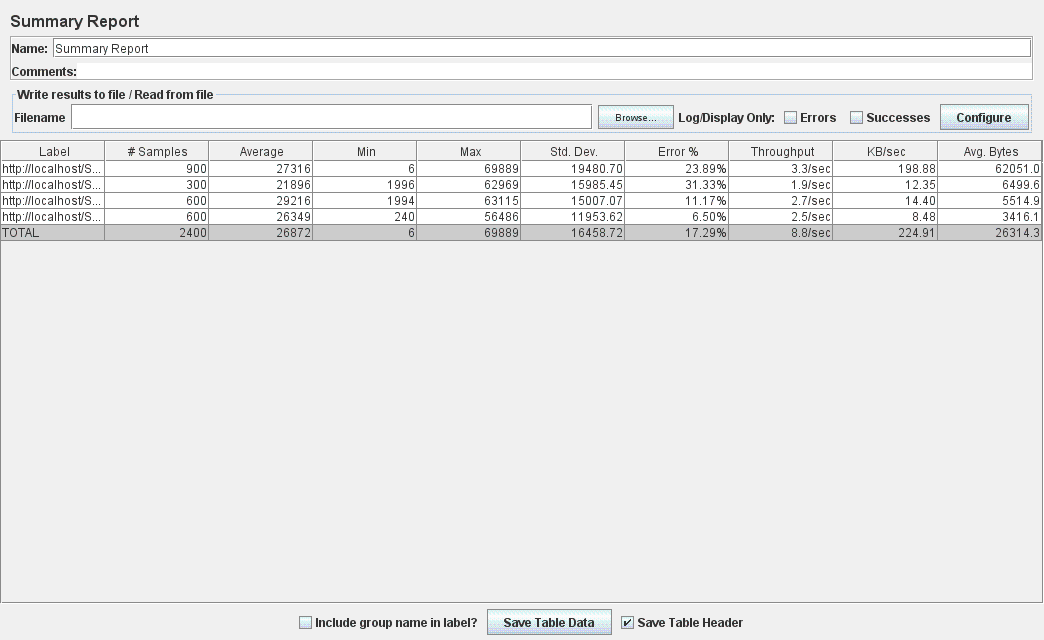
}

La executia use case-ului folosind acest index compus, query-ul logat ca fiind slow nu mai apare deloc in loguri. Acest lucru atesta o imbunatatire a performantei adusa de indexul compus din atributele username si password al entitatii User. Acest behavior s-a intalnit si in cazurile anterior preszentate: la utilizarea separata a indecsilor (initial pentru atributul username si apoi pentru password) si la utilizarea cate unui index separat pentru fiecare atribut (username si password). In continuare, vom prezenta rezultatele obtinute din JMeter la folosirea indexului compus pentru use case-ul **Login\_Skill Upgrades for developer\_Logout**, facand apoi o comparatie intre rezultatele obtinute in cazurile precedente si cel current:

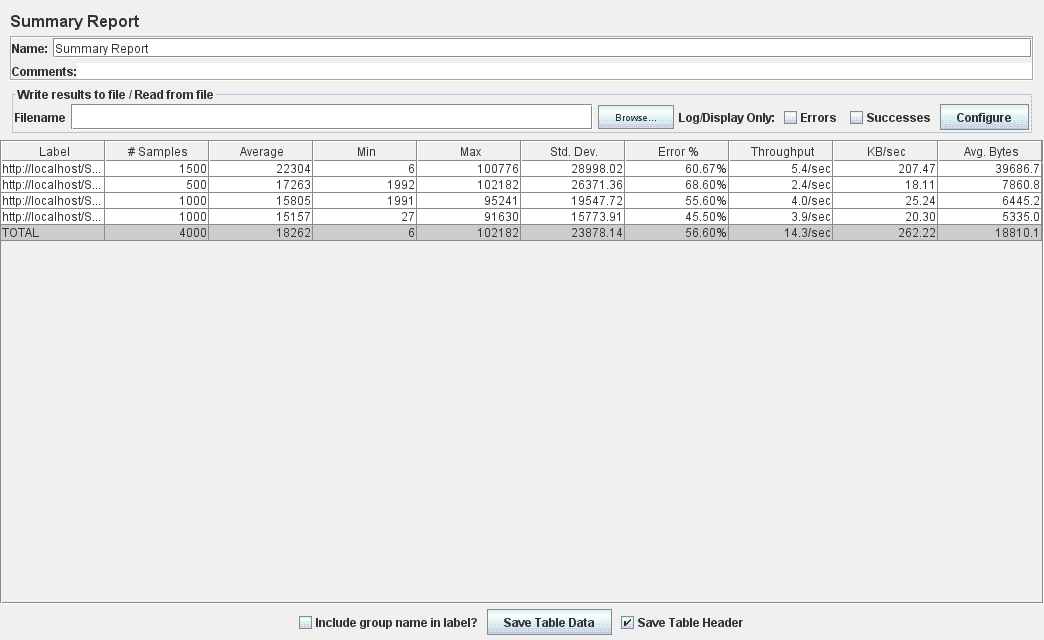
* Dupa adaugarea indexului compus, pentru un numar de 100 thread-uri (utilizatori) concurente, eroarea rezultata din rularea test case-ului corespunzator use case-ului **Login\_Skill Upgrades for developer\_Logout**, este tot de **0%,** asemenea ca si in celelalte cazuri (fara index, utilizand un singur index si respectiv, a cate doi indecsi separati pentru atributele username si password) :



* Dupa adaugarea indexului compus, pentru un numar de 300 thread-uri (utilizatori) concurente, eroarea rezultata din rularea test case-ului corespunzator use case-ului **Login\_Skill Upgrades for developer\_Logout**, este de **17.29%** :



* Dupa adaugarea indexului compus, pentru un numar de 500 thread-uri (utilizatori) concurente, eroarea rezultata din rularea test case-ului corespunzator use case-ului **Login\_Skill Upgrades for developer\_Logout**, este de **56.60%** :



Concluziile care se pot trage din aceste rezultate:

* Performanta de 0% eroare pentru 100 utilizatori executand simultan use case-ul este pastrata si dupa adaugarea indexului compus (din username si password).
* In cazul a 300 utilizatori excutand simultan use case-ul, eroarea este de 17.29% dupa adaugarea indexului compus (din username si password).
* Considerand 500 utilizatori excutand simultan use case-ul, eroarea este de 56.60% dupa adaugarea indexului compus (din username si password).

Analizand rezultatele obtinute in studiul experimental efectuat, putem concluziona faptul ca index-ul compus nu ofera o imbunatatire a performantei in toate cele trei cazuri (100, 300 si respectiv 500 de utilizatori simultani). Mai precis, pentru un numar de 300 utilizatori, se observa metrici mai bune obtinute la utilizarea unui singur index corespunzator atributului username, precum si a cate unui index separat pentru username si, respectiv, password. Pentru celelalte cazuri analizate, performanta aplicatiei este pastrata (prin mentinerea unei erori de 0% la un numar de 100 utilizatori) sau imbunatatita (prin reducerea erorii la un numar de 500 utilizatori).

Tabelul de mai jos ofera o sumarizare a erorii raportate in raportul generat de JMeter pentru toate cazurile analizate:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 100 utilizatori | 300 utilizatori | 500 utilizatori |
| Fara Index | 0% | 20.04% | 61.18% |
| Index pt username | 0% | 16.00% | 60.17% |
| Index pt password | 0% | 23.62% | 56.60% |
| Index pt username + Index pt password | 0% | 13.33% | 58.53% |
| Index compus din username si password | 0% | 17.29% | 56.60% |

Din studiul experimental legat de metodologia de alegere potrivita a indexarii pentru entitatile stocate in baza de date, putem concluziona faptul ca in majoritatea cazurilor, indexarea ofera o imbunatatire a performantei aplicatiei. Mai mult, pornind de la query-urile SQL logate ca fiind “slow”, se recomanda adaugarea indexarii pentru atributul/atributele care apar in clauza WHERE. In cazul a mai multor atribute folosite in filtrarea rezultatelor (in clauza WHERE a SELECT query-ului), performanta este maximizata in crearea unui index separat pentru fiecare atribut, sau a unuia singur (alegerea acestuia determinande-se prin rezultate experimentale). Surprinzator, crearea unui index compus din aceste atribute poate aduce “overhead”, oferind o performanta putin imbunatatita decat in cazul utilizarii a cate unui index separat pentru fiecare atribut, sau a unuia singur. Desigur, testele experimentale vor oferi rezultate particulare fiecarui use case in parte, ajutand la alegerea adecvata a indexarii entititatilor stocate in baza de date.

Pentru use case-ul studiat (**Login\_Skill Upgrades for developer\_Logout)**, in urma rezultatelor dobandite, se for folosi doi indecsi (unul pentru username si unul pentru password), ducand astfel la maximizarea performantei.

Next: index pentru alte slow query-uri logate (pt alte use case-uri)

Pentru use case-ul **Login\_Add New Developer\_Delete Old Developer\_Logout**, SQL query-ul logat ca fiind slow este :

***SELECT \* FROM Developer d WHERE d.firstName = ‘… ‘ and d.lastName =’…’;***

Astfel, tinand cont de rezultatele obtinute la use case-ul anterior(**Login\_Skill Upgrades for developer\_Logout)**, unde filtrarea rezultatelor in urma SELECT-ului se facea tot dupa doua atribute (username si password), vom alege adaugarea a doi indecsi separati pentru atributele utilizate in clauza WHERE (firstName si lastName) corespunzatoare entitatii Developer.

@Column(name="firstName")

@Index(name = "firstNameIDX")

**private** String firstName;

@Column(name="lastName")

@Index(name = "lastNameIDX")

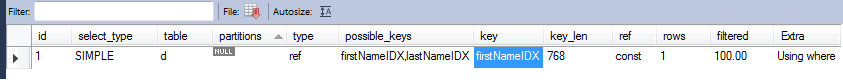
**private** String lastName;

Performanta adusa de acesti doi indecsi adaugati s-a observat din logurile de MySQL pentru slow query-uri. Astfel, inainte de adaugarea indecsilor, clauza SELECT (SELECT \* FROM Developer d WHERE d.firstName = ‘… ‘ and d.lastName =’…’;) avea un numar de 240 de aparitii in fisierul de loguri slow. Dupa crearea acestor doi indecsi, executand use case-ul in aceleasi conditii, query-ul nu mai aparea deloc in logul de slow queries.

Mai mult, pentru validarea aplicarii corecte a celor doi indecsi in momentul executiei query-ului SQL, am executat comanda:

***EXPLAIN EXTENDED SELECT \* FROM Developer d WHERE d.firstName = ‘…’ and d.lastName = ‘…’;***

Rezultatul obtinut sub forma tabelara atesta recunoasterea acestor doi indecsi si utilizarea lor in clause SELECT:



---------------------------------------------------------------------------------------------------------------------

Next step in the dissertation thesis research is adding **Caching** for the SCRUM web application and then evaluate the performance improvement obtained.

First: **study** different caching methodologies and approaches, **compare** them and establish the one(s) appropriate for the current application.

Then **implement** the caching strategy established and **assess** the performance enhancements gathered.

Consider another suitable caching methodology, implement it and test the results obtained.

Finally, consider also developing a combination of cache strategies (if appropriate) and evaluate the performance metrics (using JMeter and VisualVM application performance monitoring tools).

Caching:

* **Spring eh-cache** + **Hibernate eh-cache**
* study and document comparison between Spring & Hibernate cache
* implement caching using **Spring eh-cache for a service method** (from Service layer) and test the performance obtained
* implement caching using **Hibernate eh-cache for a Java entity class** (from Model layer) and test the performance obtained
* implement caching using **both Spring eh-cache and Hibernate eh-cache** and test the performance obtained