Profiling

US-24 Owner sees trainer’s personal information

Diāna Bukša

For profiling I chose US-24, where owner is accessing trainer's personal information. As a profiler I used Glowroot.

After executing the steps for US-24 that have been defined before (as in, for example, positive UI test), I checked the queries that have been executed together with loading an exact page.

I noticed that when the trainer's list page is being opened (it shows the list of all trainers), 16 queries have been executed. This page only shows the first, last name of the trainer, his email and phone number. It definitely should not be 16 queries for the information that has been shown.

Breakdown:	total (ms)	count
http request	58.4	1.0
hibernate query	28.7	1.0
servlet dispatch	17.9	1.0
jdbc query	8.1	16.0
jsp render	4.1	1.0
hibernate commit	1.2	1.0
jdbc commit	0.065	2.0
jdbc get connection	0.043	1.0

switch to tree view

Figure 1 Number of queries for trainer's listing

Then I took a more detailed look inside what those 16 queries were:

`select user0_.username as username1_12_0_, user0_.enabled as enabled2_12_0_, user0_.passw...`	5.9	2	2.9	1.0
`select interventi0_.pet_id as pet_id5_2_0_, interventi0_.id as id1_2_0_, interventi0_.id ...`	0.44	2	0.22	0.5
`select rehabs0_.trainer_id as trainer_6_8_0_, rehabs0_.id as id1_8_0_, rehabs0_.id as id1...`	0.43	2	0.22	1.0
`select adoptions0_.pet_id as pet_id4_0_0_, adoptions0_.id as id1_0_0_, adoptions0_.id as ...`	0.39	3	0.13	0
`select rehabs0_.pet_id as pet_id5_8_0_, rehabs0_.id as id1_8_0_, rehabs0_.id as id1_8_1_,...`	0.30	2	0.15	1.0
`select visits0_.pet_id as pet_id4_15_0_, visits0_.id as id1_15_0_, visits0_.id as id1_15_...`	0.25	2	0.13	1.0
`select interventi0_.vet_id as vet_id6_2_0_, interventi0_.id as id1_2_0_, interventi0_.id ...`	0.16	1	0.16	1.0
`select specialtie0_.vet_id as vet_id1_13_0_, specialtie0_.specialty_id as specialt2_13_0_...`	0.15	1	0.15	2.0
`select trainer0_.id as id1_10_, trainer0_.first_name as first_na2_10_, trainer0_.last_nam...`	0.12	1	0.12	2.0

Figure 2 Queries for trainer's listing

As we can see, many unnecessary queries have been loaded. For example, we can see that queries such those regarding to visits and vets have also been executed, although trainer does not have any relationship with these entities mentioned before. That means that those queries are taking more time to load the page and it means that there is a place for improvement. In Trainer's model I changes the fetch type strategy from "Eager" to "Lazy", so that only necessary queries would be executed.

```
@OneToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
@JoinColumn(name = "username", referencedColumnName = "username")
private User user;

@OneToMany(cascade = CascadeType.ALL, mappedBy = "trainer", fetch = FetchType.LAZY)
private Set<Rehab>  rehabs;
```

Figure 3 Changes made to Trainers model

Since the trainer has a relationship with 2 entities – user and rehab, then I changed fetch type for both of the entities.

After doing so, I executed US-24 again, I checked if there are any changes regarding to queries execution.

Breakdown:	total (ms)	count
http request	36.4	1.0
servlet dispatch	21.6	1.0
jsp render	12.1	1.0
hibernate query	2.2	1.0
hibernate commit	1.0	1.0
jdbc query	0.27	1.0
jdbc commit	0.12	2.0
jdbc get connection	0.059	1.0

switch to tree view

Figure 4 Number of queries after improvement

We can see that the number of queries have been significantly reduced, from 16 to 1. That one query is:

	Total time (ms)	Total count	Avg time (ms)	Avg rows
select trainer0_.id as id1_10_, trainer0_.first_name as first_na2_10_, trainer0_.last_n...	0.27	1	0.27	2.0

Figure 5 Query after improvements

As I open the detailed query description, I can see that this query is responsible for fetching trainer's name, last name, email and phone number, what is exactly what we need for this view and nothing more.

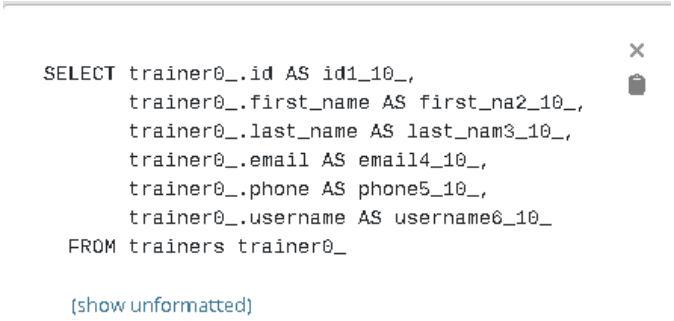

Figure 6 Query description

As the performance has been improved, the page is loading faster. Before (as shown in Figure 1) we can see that the time for this request took 58.4 ms while now it is 36.4 ms (as shown in the Figure 5).

In general, especially when working with multi-functional websites, that have many pages to load, such improvements can significantly improve user experience while using the website. For example, if there are big amount of information included in the page, that consists of many predefined entities, then such optimization steps will have bigger effect, since the http request time will be reduced.