U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Unidade curricular: Programação concorrente

Catarina Ferreira Teixeira – L:CC-up201805042

Implementação um servidor correspondente a uma base de dados para requisições de livros numa biblioteca

Trabalho prático

Responsável por este módulo: Mário Florido

Março 2020

O objetivo deste trabalho é implementar um servidor correspondente a uma base de dados para guardar requisições de livros numa biblioteca. Esta base de dados guarda requisições que são pares de identificadores de pessoas e de livros requisitados por essa pessoa. A minha base de dados é constituída por 3 tabelas: pessoa, livro e req. A tabela pessoa tem como parâmetros o CC (**cartaoC**), o Nome (**nome**), a Morada (**morada**) e o Telefone (**telefone**), a tabela livro tem como parâmetros o Código do livro (**codigo**), o Título do livro (nLivro) e o(s) autor(res) (**autores**). A tabela req, como é a minha tabela relacional entre pessoa e livro vai ter como parâmetros o CC (**cC**) relativamente à pessoa e o Código (**cod**) do livro relativamente ao livro. Para testar a base de dados e o servidor foi pedido uma série de mensagens de **lookup** e uma série de mensagens de **update**.

Mensagem de lookup:

- **livros:** dado um número de cartão de cidadão determina a lista de livros requisitada por essa pessoa;
- **empréstimos:** dado o título de um livro determina a lista de pessoas que requisitaram esse livro;

- **requisitado:** dado o código de um livro determina se o livro está requisitado (retorna um booleano);

- **códigos:** dado o título de um livro retorna a lista de códigos de livros com esse título;

- **nRequisicões:** dado um número de cartão de cidadão retorna o número de livros requisitados por essa pessoa;

Mensagem de update:

- **add_req:** dados os dados de uma pessoa e o código de um livro acrescenta o par {pessoa, livro} à base de dados;

- **retorno:** dado um número de cartão de cidadão e o código de um livro retira o par respectivo da base de dados;

Potencialidades do Erlang

O erlang é uma linguagem orientada a processo, ou seja, usa processos para se comunicarem entre si. Podem mandar e receber mensagens de forma simples, sendo uma linguagem de muito usada no estrangeiro para criar aplicações, como por exemplo o WhatsApp. O erlang é uma linguagem funcional como o Haskell e é muito usada para o uso frequente de funções "puras", funções de ordem superior e pattern matching. É uma linguagem muito simples de compreensão e até um bocado divertida de se explorar.

Como iniciar o programa

Para iniciar o programa temos de primeiro inicializar as máquinas virtuais. Precisamos de um terminal para o servidor e pelo menos um terminal para os clientes. Para ter os terminais diferentes, fazemos por exemplo para o servidor:

erl –sname gandalf -> Aí íamos ter um terminal com o nome gandalf para o servidor.

Depois compilamos o nosso ficheiro do servidor com o comando **c(servidor).** Depois fazemos **servidor:starter().** para ligar o nosso servidor.

Para o cliente podemos fazer por exemplo:

erl –sname bilbo -> iríamos ter um terminal com o nome bilbo

Depois compilamos o ficheiro **servidor.erl** no cliente da mesma maneira como fizemos no servidor (**c(servidor)**) e depois temos de fazer as chamadas ao servidor com os seguintes comandos por ordem (o **gandalf@catarina-X550JK** é o nome do meu terminal servidor):

rpc:call('gandalf@catarina-X550JK", servidor, inicia, []). -> Isto vai iniciar a nossa base de dados, criando as tabelas.

rpc:call('gandalf@catarina-X550JK", servidor, start, []). -> Isto vai esperar que todas as nossas tabelas estejam acessíveis.

rpc:call('gandalf@catarina-X550JK", servidor, reset, []). -> Isto vai inicializar o conteúdo das tabelas a partir das tabelas de exemplo que demos em **tabelas()** no ficheiro da base de dados (**baseDados**).

Feito isto agora temos uma vaga de funções que podemos testar no nosso servidor.

Execução de funções e apresentação de resultados

Mensagens de lookup:

livros: vai receber como argumento o CC da pessoa. O que fiz foi fazer uma lista em compreensão em que ia à tabela livro e req e verificava se havia um CC igual na tabela req que correspondia ao que dávamos como argumento e também se o código do livro na tabela livro associado ao CC existia na tabela dos livros. Executando a função correspondente (**book**) o resultado que iremos obter no cliente (**bilbo**) é a seguinte segundo a nossa base de dados:

```
(bilbo@catarina-X550JK)6> rpc:call('gandalf@catarina-X550JK',servidor,book,[3006
]).
[{livro,"32","Os maias","Eça de Queiros"}]
```

empréstimos: recebe como argumento o título do livro. O que fiz foi uma lista em compreensão que verifica se existe um código na tabela livro correspondente a um código na tabela req, também verifica se existe um título na tabela livro igual ao que damos no argumento e depois verificamos se existe um CC na tabela req igual ao CC na tabela pessoa. Executando a função correspondente (**empr**) o resultado que iremos obter no cliente (**bilbo**) é a seguinte segundo a nossa base de dados:

```
(bilbo@catarina-X550JK)15> rpc:call('gandalf@catarina-X550JK',servidor,empr,["Os
 maias"]).
[{pessoa,3006,"Catarina Teixeira","Rua do Atum",9323},
 {pessoa,7345,"Maria João Bacalhau","Rua Ponte de Lime",
         9166}]
```

requisitado: recebe como argumento o código do livro. O que fiz foi uma lista em compreensão que verifica se existe alguma entrada na tabela req em que o código é igual ao Código que damos como argumento. Depois vou buscar o tamanho da lista resultante e faço uma condição em que se a lista for maior que 0 quer dizer que existe uma entrada então retorna true, senão retorna false. Executando a função correspondente (**req**) o resultado que iremos obter no cliente (**bilbo**) é a seguinte segundo a nossa base de dados:

```
(bilbo@catarina-X550JK)5> rpc:call('gandalf@catarina-X550JK',servidor,req,["32"]
).
true
```

```
(bilbo@catarina-X550JK)6> rpc:call('gandalf@catarina-X550JK',servidor,req,["72"]
).
false
```

códigos: recebe como argumento o título do livro. O que fiz foi criar uma lista em compreensão em que verifico se existe um título na tabela livro que corresponde ao título que damos como input e retornamos o código correspondente ao título do livro. Executando a função correspondente (**code**) o resultado que iremos obter no cliente (**bilbo**) é a seguinte segundo a nossa base de dados:

```
(bilbo@catarina-X550JK)7> rpc:call('gandalf@catarina-X550JK',servidor,code,["Os
maias"]).
["33","32"]
```

nRequisicões: recebe como argumento o código do livro. O que fiz foi uma lista em compreensão em que verifica se existe um CC na tabela pessoa igual na tabela req e também se o CC na tabela pessoa é igual ao CC que damos como argumento e devolvemos o código

correspondente na tabela req. Executando a função correspondente (**nReq**) o resultado que iremos obter no cliente (**bilbo**) é a seguinte segundo a nossa base de dados:

```
(bilbo@catarina-X550JK)10> rpc:call('gandalf@catarina-X550JK',servidor,nReq,[300
6]).
1
(bilbo@catarina-X550JK)11> rpc:call('gandalf@catarina-X550JK',servidor,nReq,[344
4]).
2
```

Mensagem de update:

Add_req: recebe como argumento o CC da pessoa e o código do livro. Verifico se existe a pessoa na tabela pessoa e verifico se o código existe na tabela livro e se também se o livro já se encontra requisitado ou não, se não existir imprime uma mensagem de erro no servidor se não existir a pessoa, se não existir o código ou se o livro já se encontra requisitado. Executando a função correspondente (**add**), o caso em que dá fica {atomic, ok} e adiciona à tabela req e depois uso uma função extra que implementei em que vejo o conteúdo da tabela req. No caso em que não dá imprime no servidor as mensagens de erro.

Caso for verdade iremos obter no cliente (**bilbo**):

```
(bilbo@catarina-X550JK)5> rpc:call('gandalf@catarina-X550JK',servidor,add,[3006,
"72"]).
{atomic,ok}
(bilbo@catarina-X550JK)6> rpc:call('gandalf@catarina-X550JK',servidor,treq,[]).
[{req,5656,"42"},
 {req,3444,"12"},
 {req,3444,"53"},
 {req,3006,"32"},
 {req,3006,"72"},
 {req,9999,"11"},
 {req,7345,"33"}]
```

Caso for falso iremos ter uma mensagem de erro no servidor (**gandalf**):

```
(bilbo@catarina-X550JK)7> rpc:call('gandalf@catarina-X550JK',servidor,add,[3006,
"99"]).
{atomic,ok}
(bilbo@catarina-X550JK)8> rpc:call('gandalf@catarina-X550JK',servidor,add,[33232
,"72"]).
{atomic,ok}
(bilbo@catarina-X550JK)9> rpc:call('gandalf@catarina-X550JK',servidor,add,[3006,
"42"]).
{atomic,ok}
```

```
(gandalf@catarina-X550JK)3>
=INFO REPORT==== 25-Mar-2021::19:14:32 ===
    application: mnesia
    exited: stopped
    type: temporary
Não existe o código!
Não existe a pessoa!
Já tá requisitado!
```

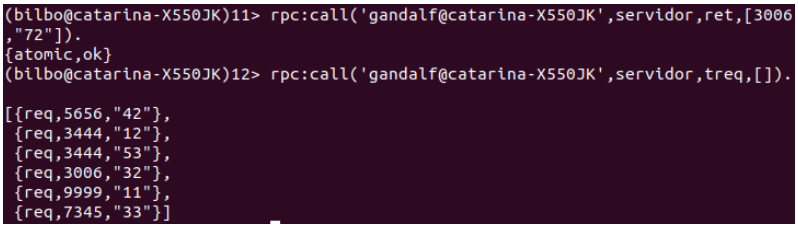
retorno: recebe como argumento o CC da pessoa e o código do livro. Verifico se existe a pessoa na tabela pessoa e verifico se o código existe na tabela livro, se não existir imprime uma mensagem no servidor se não existir a pessoa ou se não existir o código. Executando a função correspondente (**ret)**, o caso em que dá fica {atomic, ok} e remove da tabela req, depois uso uma função extra que implementei em que vejo o conteúdo da tabela req. No caso em que não dá imprime no servidor as mensagens de erro.

Caso for falso, iremos ter uma mensagem de erro no servidor (**gandalf**):

```
(bilbo@catarina-X550JK)16> rpc:call('gandalf@catarina-X550JK',servidor,ret,[3006
,"99"]).
{atomic,ok}
(bilbo@catarina-X550JK)17> rpc:call('gandalf@catarina-X550JK',servidor,ret,[3564
6,"72"]).
{atomic,ok}
```

```
Não existe o código!
Não existe a pessoa!
```

Caso for verdade iremos obter no cliente (**bilbo**):



Funções extras:

- **Add_pessoa:** Adiciona pessoas à tabela pessoa. Os argumentos eram o CC, o Nome, a Morada e o Telefone e para executar no cliente usa-se, por exemplo:

  rpc:call(‘gandalf@catarina-X550JK',servidor,addp,[3232,Amelia,”Rua Aberta”,9877]).

- **Add_livro:** Adiciona livros à tabela livro. Os argumentos eram o Código, Título, Autor e para executar no cliente usa-se, por exemplo:

  rpc:call(‘gandalf@catarina-X550JK',servidor,addl,[“435”,”Memorial”,”Amelio”]).

- **Tab_req:** Vê o conteúdo da tabela req.

  rpc:call('gandalf@catarina-X550JK',servidor,treq,[]).

- **Tab_pessoa:** Vê o conteúdo da tabela pessoa.

  rpc:call('gandalf@catarina-X550JK',servidor,tpess,[]).

- **Tab_livros:** Vê o conteúdo da tabela livro.

  rpc:call('gandalf@catarina-X550JK',servidor,tlivro,[]).

- **Tabela:** Vê o conteúdo de todas as tabelas.

  rpc:call('gandalf@catarina-X550JK',servidor,tabela,[])

- **Stop:** Fecha o servidor e no terminal onde se encontra o servidor imprime uma mensagem a avisar que fechou.

  rpc:call('gandalf@catarina-X550JK',servidor,stop,[])

Conclusão

Neste trabalho aprofundei os meus conhecimentos de Erlang, aprendendo de forma mais aprofundada como funciona a linguagem e que aplicações a mesma tem. De forma geral gostei de fazer o trabalho e diverti-me a fazer o mesmo, aprendendo a fazer um servidor com vários clientes a aceder ao mesmo, aplicado a uma base de dados mnesia feita de raiz com várias mensagens de lookup e update.