Laboratorios de Informática 3
MiEI — 2017/18

Grupo nr.	48
a82053	Nelson Sousa
a80207	Rui Ribeiro
a81922	Tiago Sousa

1 Tipo de dados Concreto

```
struct TCD_community {
    GHashTable* posts;
    GHashTable* users;
    GHashTable* tags;
    GList* posts_list;
    GList* users_list;
    GList* questions_list;
    GList* users_list_rep;
    long total_questions;
    long total_answers;
};
```

2 Estrutura de Dados

- MyPost

  Estrutura responsavel por armezenar a informação dos posts.

  Contem:

  - Id do post;
  - Titulo do post;
  - Id do usuário que publicou o post;
  - Data de criação;
  - Tipo do post, 1 se for pergunta 2 se for resposta;
  - Lista com as tags que o post possui;
  - Lista com as respostas ao post caso este seja uma pergunta;
  - Tabela de hash com as respostas ao post caso este seja uma pergunta;
  - Score do post, ou seja numero de votos;
  - Numero de comentarios;
  - Pontuação do post;
  - Numero de respostas;
  - Id do post "pai", ou seja caso o post seja uma resposta, contem o id da respetiva pergunta;

- MyUser

  Estrutura responsavel por armezenar a informação dos users.

  Contem:

  - Id do user;
  - Nome do user;
  - Short bio do user;
  - Número de perguntas feitas pelo user;
  - Número de respostas dadas pelo user;
  - Número total de posts feitos pelo user;
  - Reputação do user;
  - Lista com os ultimos 10 posts;

- Tabelas de Hash

  Na nossa estrutura possuimos 3 tabelas de hash, para conseguirmos tempo de procura constante. Uma tabela para os Users que contem os dados armazenados segundo a estrutura **MyUser**, onde a key é representada pelo id do user. Uma para os Posts que contem os dados armazenados segundo a estrutura **MyPost**, onde a key é representada pelo id do post. E finalmente uma tabela de hash para as tags, onde a key é o nome da tag e o id da tag é o value(terá utilidade na query 11.

- Listas

  Para melhorar os tempos decidimos produzir varias listas ligadas, ordenadas de diferentes formas para acelarar as querys, assim possuimos 4 listas ligadas. Duas contem todos os users, estando apenas ordenadas de maneira diferente. Um ordenada por numero total de posts que cada usuário tem, e outra ordenada pela reputação de cada usuário.

  As outras duas listas contem os posts ordenados por data. A diferença é que uma contem tanto perguntas como respostas e outra só contem as perguntas.

  Nota: estas listas são "shallow copy's" das tabelas de hash, ou seja o conteudo dentro de cada nodo da lista é o mesmo que está contido numa das tabelas de hash.Assim evitamos um gasto de memória muito grande pois teriamos de copiar cada post e cada usuário tantas vezes quanto o numero de listas que possuiamos.O encapsulamento continua garantido apesar disso pois, não é possivel para o utilizador do programa alterar nenhuma destas listas.

- Contadores

  Para finalizar a nossa estrutura armazena o número total de perguntas e respostas contidas no ficheiro xml.

3 Modularização Funcional

Neste projeto Modularizamos o nosso codigo em varias partes, um modulo "interface" que contem a chamada de todas as querys do load da estrutura de dados e do clean da estrutura, um modulo para cada querry contendo a sua respetiva implementação, um modulo "loader" que contem a implementação do load, carregando a estrutura de dados e funes de acesso aos campos da estrutura como a get_user() ou a get_post(), um modulo "parser" que contem 2 funoes auxiliares, que abrem ficheiros xml e que encontra atributos especificados pelo utilizador, possuimos tambem 2 modulos "myuser" e "myparser" que implementam as estrutras **MyUser** e **MyPost** e as respetivas funes necesseąarias que operam sobre essas estruras, por fim temos um modulo "date_to_int" que é responsavel por transformar uma data em dias passando do formato Date para um int;

4 Abstração de Dados

Garantimos abstração de dados ao definir as nossas estruturas nos ficheiros .c respetivos, passando apenas uma API ao utilizador que contem as funções que eles podem usar para ter acesso a informaao contida nestas mesmas estruturas, essas funções estao feitas de modo a garantir encapsulamento ou seja tudo o que passado ao utilizador são apenas copias da informaçao contida na estrutura para que não seja possivel esta ser alterada por utilizadores externos.

5 Estrategias seguidas nas query's

- **Query 1**

  Nesta query tiramos proveito das tabelas de hash de usuários e de posts que possuimos na estrutura tendo acesso em tempo constante a qualquer post.Caso o post passado como parametro seja uma resposta fazemos o get do respetivo post na tabela de hash, e verificamos o campo da estrutura que contem o id do pai.A seguir é só consultar o campo da estrutura que tem o titlo e o campo que tem o id do dono do post.Por fim fazemos get do dono do post e consultarmos o campo que contem o seu nome.

- **Query 2**

  Atrves da lista ligada users_list que possuimos na estrutura, que está por ordem decrescente número total de post que um usuário fez.A unica coisa necessária a fazer para esta query é obter os N primeiros id's desta lista.

- **Query 3**

  A estrategia adotada nesta query foi percorrer a lista ligada posts_list que está ordenada da data da mais recente para a mais antiga.Parando quando se encontra uma data maior que a data que é passada como parametro (Date end).A medida que percorre-mos a lista vamos incrementando as variaveis que contam o número de perguntas e respostas dependo do tipo de post que obsevamos em cada iteração.

- **Query 4**

  Nesta query tiramos proveito de possuirmos uma lista ligada só com as perguntas na nossa estrutura.Assim o tempo de percorrer a lista é consideravelmente menor.Tal como na query anterior percorremos a lista ate que seja encontrada uma data maior que a data que é passada como parametro (Date end).Em cada iteração vemos se o post possui ou no a tag passada como parametro, visto que na nossa estrutura os posts armazenam as tags que possuem.Em caso afirmativo inserimos o id desse post numa lista que no fim é passada ao utilizador.

- **Query 5**

  A partir da tabela de hash tudo o que precisamos de fazer nesta query é o get do user atravéz do id passado como parametro e depois consultar os campos shorbio e lastposts que a estrutra MyUser possui.

- **Query 6**

  A estrategia seguida nesta query foi percorrer a lista ligada posts_list como na query 2 e inserindo todos as respostas numa lista auxiliar.

  No fim de percorrida a lista, ordena-se lista auxiliar que, assim contem as perguntas por ordem decrescente de votos. Assim é só preciso devolver os N primeiros id's dessa lista ao utilizador.

- **Query 7**

  Para resover esta query basta percorrer a lista de perguntas (questions_list), e inserir todas as perguntas que estao dentro numa lista auxiliar, ordenando-a no fim, e depois retirar os N primeiros id's da lista auxiliar.

- **Query 8**

  Para fazer esta query é apenas necessário percorrer a lista ligada questions_list que possui apenas as perguntas e está ordenada por cronologia inversa.À medida que encontramos posts cujo o titulo contenha a palavra passada como parametro inserimos o id dessa post numa lista que é no fim passada ao utilizador.

- **Query 9**

  É de relembrar que cada post, caso seja uma pergunta, armazena uma lista com todas as suas respostas.Assim, percorrendo mais uma vez a lista ligada questions_list e para cada pergunta percorrendo a sua respetiva lista de respostas ate que se detete que ambos os utilizadores participaram nessa pergunta ou ate ao fim da lista de respostas, obtemos a lista com os id's das perguntas em que ambos os utilizadores participaram, a ordem cronologica inversa est garantida pois a lista ligada est também em ordem cronologica inversa.

- **Query 10**

  Estando a pontuação de cada resposta calculada previamente pelo load, para esta query só precisamos de fazer o get da pergunta atravéz do seu id, a partir da tabela de hash. E percorrer a lista das respostas que essa pergunta tem vendo qual tem a pontuação mais alta.

- **Query 11**

  A estrategia seguida nesta query foi percorrer a lista de todos os posts e, caso o proprietário esteja nos N primeiros lugares da lista users_list_rep, que está na nossa estrutura de dados por ordem decrescente de reputação, todas os ids das tags desse post são adicionados por ordem decrescente de vezes que a tag já foi usada, a uma lista auxiliar. Esta lista tem em cada nó um par com o id da tag e o número de vezes que esta ja foi usada. No fim basta retornar os N primeiros ids da lista auxiliar.

  Esta query foi melhorada posteriormente! Ver secçao "Estrategias para melhorar o desempenho"

  Em caso de empate de respostas em algumas querys, o criterio escolhido para desempatar em todas elas foi a ordem decrescente de ids.

6 Estrategias para melhorar o desempenho

A principal estrategia seguida para melhorar o desempenho foi passar a maior quantidade possivel de trabalho para a parte de loading da estrutura. Isto foi feito com o pressoposto que o load e feito apenas uma vez e cada query pode ser executada varias vezes. Assim é preferivel ter um load mais pesado a nivel de complexidade e ter querys mais eficientes. Coisas como criar diferentes listas ordenadas de diferentes maneiras e filtradas de diferentes maneiras, o facto de cada pergunta armazenar a lista das suas respostas e cada usuário armazenar os seus ultimos 10 posts foram implementadas neste ambito. O que faz melhorar o desempenho de grande parte das querys .

Na query 11 de modo melhorar o desempenho, criamos uma tabela de hash com os N primeiros users da lista user_list_rep, para que a parte de verificar se o dono de um post esta ou nao no top N users seja feita em tempo constante.