UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
http://www.icmc.usp.br/

Professor: Rodrigo Fernandes de Mello (mello@icmc.usp.br)
Alunos PAE: Lucas Pagliosa (lucas.pagliosa@usp.br)
Felipe Duarte (fgduarte@icmc.usp.br)
Monitor: Victor Forbes (victor.forbes@usp.br)

Trabalho 01: SGBD

1 Prazos e Especificações

O trabalho descrito a seguir é individual e não será tolerado qualquer tipo de plágio ou cópia em partes ou totalidade do código. Caso seja detectado alguma irregularidade, os envolvidos serão chamados para conversar com o professor responsável pela disciplina.

A entrega deverá ser feita única e exclusivamente por meio do Sistema Run.codes no endereço eletrônico `https://run.codes` até o dia **15 de agosto de 2016 às 23 horas e 59 minutos**. Sejam responsáveis com o prazo final para entrega, o Run.codes está programado para não aceitar submissões após este horário e não será aceito entrega fora do sistema.

Leia a descrição do trabalho com atenção e várias vezes, anotando os pontos principais e as possíveis formas de resolver o problema. Comece a trabalhar o quanto antes para não ficar dúvidas e você consiga entregar o trabalho a tempo.

O trabalho deverá ser submetido em formato zip/Makefile contendo o arquivo principal e os arquivos complementares (pares `.h .c`) para solucionar o problema. Serão avaliados não só o resultado final do sistema, mas também a indentação e documentação interna. Atente para o formato do Makefile que o run.codes precisa para corrigir o seu trabalho: compilação com a tag `all` e execução com a tag `run`.

2 Descrição do Problema

Um Sistema de Gerenciamento de Banco de Dados (SGBD) é um conjunto de programas capazes de interpretar e executar comandos para a criação, modificação e diversas outras operações envolvendo banco de dados. Além da correta interpretação dos comandos inseridos pelo usuário, um SGBD deve realizar certos processamentos extras para o gerenciamento otimizado de milhões de dados. Por exemplo, dado uma tabela criada com o comando:

```
create table pessoa (codigo int, nome char[80], idade int, sexo char);
```

podemos inserir qualquer pessoa com um código identificador, nome, idade e gênero no SGBD. Importante dizer que neste trabalho não existe o conceito de chave primária, logo você não precisa se preocupar em verificar a unicidade dos campos inseridos. Após a tabela 'pessoa' ser criada no SGBD, podemos inserir instâncias deste tipo no banco de dados da seguinte forma:

```
insert into pessoa (codigo, nome, idade, sexo) values (1, 'joao', 10, 'M');
insert into pessoa (codigo, nome, idade, sexo) values (2, 'paula', 11, 'F');
```

Após os comandos acima, duas pessoas são inseridas no SGBD. Inseridas tem o sentido de escritas em um arquivo específico (pode ser binário ou ASCII, como você preferir – binário é mais compacto e rápido de manusear, arquivos ASCII são melhores para acompanhamento e depuração), pois como o SGBD tem como propósito gerenciar grandes volumes de dados torna-se

ineficaz e até mesmo impossível carregar e manter todos os dados em memória RAM. O correto armazenamento e tratamento de instâncias é necessário para a realização otimizada de consultas (*queries*) ao SGBD. Por exemplo, seja a consulta:

```
select pessoa codigo '1';
```

para encontrar todas as instâncias com código 1 na tabela 'pessoa'. O programa pode procurar todas as pessoas com código 1 sequencialmente ou ordenar as pessoas por código e retornar as instâncias com código 1 após uma busca binária. Veja que apesar de ser mais rápido encontrar as instâncias de código 1 no arquivo ordenado, toda ordenação tem um custo. Isso quer dizer que se a quantidade de consultas e inserções forem pequena, vale mais a pena buscar sequencialmente do que realizar a ordenação. Suponha que temos 1 bilhão de pessoas já ordenadas e 10 novas pessoas inseridas. Apesar da ordenação ter custo computacional $\mathcal{O}(n \log n)$, é menos custoso computacionalmente buscar por pessoas com código 1 realizando duas buscas, uma sequencial no arquivo de 10 pessoas recém inseridas e outra binária no arquivo ordenado do que ordenar as 10 novas pessoas e realizar uma única busca binária.

Portanto este trabalho consiste em simular um SGBD que realiza processamento e ordenação em arquivos de dados. A seguir estão especificados todos os comandos que o seu SGBD deve suportar e interpretar.

3 Comandos do Sistema

Apesar de um SGBD completo ser considerado um sistema complexo de computador (devido ao tamanho e obrigatórias otimizações) abrangendo milhares de comandos, neste trabalho seu SGBD precisa lidar somente com os comandos descritos a seguir. O símbolo [;] significa que o comando **pode ou não terminar com ponto e vírgula**. Importante notar que os códigos podem ser tanto em maiúsculo quanto em minúsculo.

3.1 CREATE TABLE

Cria uma estrutura de dados com N campos da tabela 'tabela'. Essa estrutura de dados DEVE SER DINAMICAMENTE ALOCADA. Formato de comando:

```
create table tabela(campo1 tipo1,..., campoN tipoN);
```

3.2 INSERT

Esta instrução é responsável por inserir dados (instâncias) do tipo `tabela` em um arquivo temporário. O nome do arquivo temporário é dado pelo nome da tabela mais a extensão ".tmp", sendo assim, para a tabela 'pessoa' o nome do arquivo temporário será 'pessoa.tmp'. Formato do comando:

```
insert into tabela (campo1,..., campoN) values (valor1,..., valorN);
```

Obs: Note que apesar de em um SGBD padrão a ordem de inserção poder ser diferente da ordem de criação dos campos e apenas um subconjunto de campos ser informado com valores, neste trabalho a ordem é a mesma de criação e todos os campos possuem valor.

3.3 CREATE INDEX

Cria um arquivo binário de índice ordenado de forma estável, isto é, caso o campo a ser ordenado possua valores iguais, a ordem deve respeitar a ordem de inserção). Formato do comando:

```
create index tabela(campo);
```

As instruções do tipo acima são responsáveis por criar índices em um arquivo chamado '`tabela-campo`.idx' para busca binária. Antes de criar um arquivo de índice, você deve copiar todos os dados do arquivo temporário relativo à tabela `tabela` em um arquivo de dados cujo nome é dado pelo nome da tabela mais a extensão '.dat' e apagar o arquivo temporário.

Para a tabela 'pessoa', por exemplo, haverá um arquivo de dados chamado 'pessoa.dat' e outro temporário chamado 'pessoa.tmp'. O arquivo de índice conterá, portanto, tuplas ordenadas por `campo` com o valor do `campo` e o índice das instâncias com este campo no arquivo de dados. Dessa maneira, estamos falando para o SGBD quando ele deve ordenar os dados e remover todos os arquivos da busca sequencial (reler Seção 2.).

O campo especificado no comando determinará o requisito de ordenação. Lembre-se que se houver mais de um campo com o mesmo valor, deve-se ordenar por ordem de inserção.

3.4 SELECT

Realiza uma busca binária usando o arquivo de índice '`tabela-campo`.idx' e uma busca sequencial no arquivo temporário '`tabela`.tmp' (pois pode haver conteúdo nesse arquivo). Formato do comando:

```
select tabela campo valor;
```

Esse `select` deve retornar todos as instâncias cujo `campo` é igual a `valor`, segundo sua ordem de inserção no arquivo temporário, ou seja, deve-se garantir estabilidade na ordenação. Se uma instância A passou do arquivo temporário para o de dados antes de outra B e ambas apresentam `campo` igual a `valor`, então deve-se imprimir os dados de A antes dos dados de B.

Observe que também é necessário contar quantas instâncias foram retornados via busca binária e quantos via busca sequencial (necessário para instrução `statistics`).

Os dados a serem impressos devem ter o seguinte formato:

- `char` ou `char[tamanho]` deve ser impresso entre aspas simples na forma:

  ```
  printf("'%s'", valor);
  ```

- Os campos `int`, `float` ou `double` são impressos apenas na forma de seus valores:

  ```
  printf("%d", valor_inteiro);
  printf("%f", valor_float);
  printf("%lf", valor_double);
  ```

- Se `valor` não for encontrado:

  ```
  printf("null");
  ```

Portanto, caso haja uma instância na tabela `tabela` com o `codigo` igual a `valor` deve-se retornar todos os valores desta instância impressos nos seus respectivos tipos na forma:

```
printf("%tipo1, %tipo2, ..., %tipoN\n", valor1,valor2,...,valorN);
```

Obs: Observe que cada instância é apresentada em uma linha. Não se esqueça de pular uma linha após apresentar cada instância, inclusive o último.

3.5 SORT

O comando `sort tabela(campo);` regera um certo índice. Regerar significa apagar o arquivo de índice e produzir um novo com base no arquivo de dados. Se houver qualquer instância no arquivo temporário, deve-se transferi-la para o arquivo de dados e, em seguida, regerar o índice. Logo, este comando é uma maneira mais simples de realizar o comando `create index`.

3.6 SHOW ALL TABLES

O comando `showalltables [;]` imprime no arquivo de saída, para cada tabela criada:

```
printf("\nTablename: %s\n", tablename);
for (i = 0; i < nfields; i++)
{
  printf("\tField: %s Type: %s Size %d\n",
  fieldname[i],
  fieldtype[i],
  fieldsize[i]);
}
printf("\n");
```

3.7 SHOW ALL INDEXES

O comando `showallindexes [;]` imprime no arquivo de saída:

```
// deve estar na ordem de criação dos índices
for (i = 0; i < totalDeIndices; i++)
{
  printf("\nIndex information\n");

  // nome da tabela relacionada ao índice i
  printf("\tTablename: %s\n", tablename[i]);

  // nome do campo indexado
  printf("\tFieldname: %s\n\n", fieldname[i]);
}
```

3.8 SHOW ALL INDEXES

O comando `statistics [;]` imprime no arquivo de saída:

```
printf("#Tables: %d\n", ntables); // número de tabelas criadas até então
printf("#Indexes: %d\n", nindexes); // número de índices criados até então
printf("#Inserts: %d\n", ninserts); // número de inserts feitos até então
printf("#Selects: %d\n", nselects); // número de selects feitos até então
printf("#Sorts: %d\n", nsorts); // número de sorts feitos até então

// número de 'showalltables' feitos até então
printf("#ShowAllTables: %d\n", nshowalltables);

// número de 'showallindexes' feitos até então
printf("#ShowAllIndexes: %d\n", nshowallindexes);

// número de instâncias recuperadas via busca binária na última consulta feita
printf("#Records in last select (binary search): %d\n",
  recordsInLastSelectBinary);

// número de instâncias recuperadas via busca sequencial na última
// consulta feita
printf("#Records in last select (sequential search): %d\n",
  recordsInLastSelectSequential);
```

4 Exemplos

A Figura 1 exemplifica um exemplo do comando `create index pessoa(codigo);`. Na primeira linha, temos um arquivo já ordenado por este comando ('pessoa.dat') e mais quatro novas inserções ('pessoa.tmp'). Quando este comando é novamente executado, o SGBD remove todas instâncias do arquivo temporário e move-os para o arquivo de dados e re-ordena o arquivo de índice. Você pode fazer o arquivo de índice e de dados de forma binária ou com ASCII. Se for binária, o arquivo de índice deve conter a quantidade de bits a serem deslocados para encontrar a instância no arquivo de dados. Se for ASCII, o arquivo de índice pode conter a linha na qual a instância se encontra no conjunto de dados.

pessoa.tmp	pessoa.dat	pessoa-codigo.idx (ASCII)	pessoa-codigo.idx (Binário)
4, maria, 10, F	8, carlos, 60, M	1 4	1 356
3, jose, 5, M	2, ana', 14, F	2 2	2 178
2, felipe duarte', 28, F	14, pedro, 9, M	8 1	8 89
4, moises, 25, M	1, goku, 234, M	14 3	14 267

pessoa.dat	pessoa-codigo.idx (ASCII)	pessoa-codigo.idx (Binário)
8, carlos, 60, M	1 4	1 356
2, ana', 14, F	2 2	2 178
14, pedro, 9, M	2 7	2 623
1, goku, 234, M	3 6	3 534
4, maria, 10, F	4 5	4 445
3, jose, 5, M	4 8	4 712
2, felipe duarte', 28, F	8 1	8 89
4, moises, 25, M	14 3	14 267

Figura 1: Exemplo de arquivos '.tmp' e '.dat' antes e depois de um comando `create index`.

4.1 Exemplo Caso de Teste 1

Entrada:

```
create table pessoa (codigo int, nome char[80], idade int, sexo char );
create table empresa (razao_social char[255], endereco char[255], numero int,
  cidade char[80], estado char[2]);

showalltables

insert into pessoa (codigo, nome, idade, sexo) values (1, 'ana', 2,'F');

select pessoa codigo 1;
select pessoa codigo 2;
```

Saída:

```
\n
Tablename: pessoa
Field: codigo Type: int Size 4
Field: nome Type: char Size 80
Field: idade Type: int Size 4
Field: sexo Type: char Size 1
\n
\n
Tablename: empresa
Field: razao_social Type: char Size 255
Field: endereco Type: char Size 255
```

```
Field: numero Type: int Size 4
Field: cidade Type: char Size 80
Field: estado Type: char Size 2
\n
\n
1, ‘ana’, 2, ‘F’
null
```

4.2 Exemplo Caso de Teste 2

Entrada:

```
create table pessoa (codigo int, nome char[80], idade int, sexo char );
create table empresa (razaosocial char[255], endereco char[255], numero int,
  cidade char[80], estado char[2]);

showalltables;

insert into pessoa (codigo, nome, idade, sexo) values (1, ‘joao’, 10,‘M’);
insert into pessoa (codigo, nome, idade, sexo) values (2, ‘paula’, 11,‘F’);
insert into pessoa (codigo, nome, idade, sexo) values (3, ‘jose’, 90, ‘M’);

insert into empresa (razaosocial, endereco, numero, cidade, estado) values
  (‘Usp’, ‘Av Trabalhador Saocarlense’, 400, ‘Sao Carlos’, ‘SP’);
insert into empresa (razaosocial, endereco, numero, cidade, estado) values
  (‘Empresa A’, ‘Av Sao Carlos’, 30, ‘Sao Carlos’, ‘SP’);
insert into empresa (razaosocial, endereco, numero, cidade, estado) values
  (‘Empresa B’, ‘Av XV’, 132, ‘Araraquara’, ‘SP’);

create index pessoa(codigo);
create index pessoa(nome);

create index empresa(razaosocial);
create index empresa(cidade);

statistics;

insert into pessoa (codigo, nome, idade, sexo) values (2, ‘paulo’, 12,‘M’);
insert into pessoa (codigo, nome, idade, sexo) values (4, ‘carlos’, 20, ‘M’);

insert into empresa (razaosocial, endereco, numero, cidade, estado) values
  (‘Empresa C’, ‘Rua XIII’, 22, ‘Sao Paulo’, ‘SP’);
insert into empresa (razaosocial, endereco, numero, cidade, estado) values
  (‘Empresa D’, ‘Rua Joquim Jose’, 231, ‘Belo Horizonte’, ‘MG’);

select pessoa codigo ‘2’;

statistics
```

Saída:

```
\n
Tablename: pessoa
Field: codigo Type: int Size 4
Field: nome Type: char Size 80
```

```
Field: idade Type: int Size 4
Field: sexo Type: char Size 1
\n
\n
Tablename: empresa
Field: razaosocial Type: char Size 255
Field: endereco Type: char Size 255
Field: numero Type: int Size 4
Field: cidade Type: char Size 80
Field: estado Type: char Size 2
\n
#Tables: 2
#Indexes: 4
#Inserts: 6
#Selects: 0
#Sorts: 0
#ShowAllTables: 1
#ShowAllIndexes: 0
#Records in last select (binary search): 0
#Records in last select (sequential search): 0
2, ‘paula’, 11, ‘F’
2, ‘paulo’, 12, ‘M’
#Tables: 2
#Indexes: 4
#Inserts: 10
#Selects: 1
#Sorts: 0
#ShowAllTables: 1
#ShowAllIndexes: 0
#Records in last select (binary search): 1
#Records in last select (sequential search): 1
\n
```

5 Observações importantes

- Programe as impressões na tela EXATAMENTE como exemplificado no decorrer deste documento. Tome cuidado com pulos de linha, tabs, espaços, etc.
- Nota-se que o SGBD precisa de um *parser* capaz de interpretar comandos executados pelo usuário de forma a ver se estes estão de acordo com a sintaxe e semântica do banco de dados. No entanto, você pode assumir neste trabalho comandos bem formulados, de tal forma que seu trabalho não precisa reconhecer erros, apenas tratar dois casos: i) comandos podem conter inúmeros espaços entre palavras (por exemplo, `create_table_tabela(...` e `create_table_tabela__(...` são comandos válidos, onde _ é um espaço em branco) e ii) os comandos de impressão podem ou não ter ponto e vírgula no final.
- Coloque dentro do zip todos os arquivos de código (`*.h *.c`), o Makefile e um arquivo texto com o nome e número USP.