Universidade Federal de São Carlos
Bacharelado em Ciência da Computação
Organização e Recuperação da Informação
Prof. Tiago A. Almeida <talmeida@ufscar.br>

Trabalho 03

Tabelas *Hash*

Prazo para entrega: 09/06/2021 – 23:59

Atenção

- **Sistema de submissão:** http://judge.sor.ufscar.br/ori/
- **Arquivo:** deverá ser submetido um único código-fonte seguindo o padrão de nomenclatura `<RA>_ORI_T03.c`, ex: `123456_ORI_T03.c`;
- **E/S:** tanto a entrada quanto a saída de dados devem ser de acordo com os casos de testes abertos;
- **Identificadores de variáveis:** escolha nomes apropriados;
- **Documentação:** inclua comentários e indente corretamente o programa;
- **Erros de compilação:** nota **zero** no trabalho;
- **Tentativa de fraude:** nota **zero na média** para todos os envolvidos. Fraudes, como tentativas de compras de soluções ou cópias de parte ou de todo código-fonte, de qualquer origem, implicará na reprovação direta na disciplina. Partes do código cujas **ideias** foram desenvolvidas em colaboração com outro(s) aluno(s) devem ser devidamente documentadas em comentários no referido trecho. O que **NÃO** autoriza a cópia de trechos de código, a codificação em conjunto, compra de soluções, ou compartilhamento de tela para resolução do trabalho. Portanto, compartilhem ideias em alto nível, modos de resolver o problema, mas não o código;
- Utilize as mensagens pré-definidas (`#define`).

1 Contexto

A **UFSPay** está sendo utilizado em larga escala e agora você contratou uma equipe para dar continuidade e manutenção. O analista de dados da sua equipe identificou que agora a maior parte das operações é de busca por clientes e chaves PIX, além da crescente quantidade de transações. Sendo assim, concluiu-se que utilizar uma estrutura de *hashing* nos índices de clientes e chaves PIX poderá melhorar o desempenho do sistema, permitindo que a maioria das buscas seja realizada com poucos acessos ao disco.

2 Base de dados da aplicação

Relembrando, o sistema é composto por dados de usuários e de transações conforme descrito a seguir.

2.1 Dados dos usuários

- **CPF**: identificador único de um cliente (chave primária) que contém 11 números. Não poderá existir outro valor idêntico na base de dados. Ex: `57956238064`;
- **Nome completo**;
- **Data de nascimento**: data no formato `<DD>/<MM>/<AAAA>`. Ex: `12/01/1997`;
- **E-mail**;
- **Celular**: número de contato no formato `<99><999999999>`. Ex: `15924835754`;
- **Saldo**: saldo do cliente no formato `<9999999999>.<99>`. Ex: `0000004605.10`;
- **Chaves PIX**: campo multivalorado separado pelo caractere '`&`'. O cliente poderá cadastrar até 4 chaves, apenas uma de cada tipo: `C` (CPF), `N` (número de celular), `E` (email) e `A` (aleatória). As chaves deverão ser únicas e derivadas do cadastro do usuário, exceto para a chave aleatória. No arquivo, o primeiro caractere será o tipo da chave, e as chaves estão dispostas na ordem de inserção. Ex: `C57956238064&N15924835754&Ejose@gmail.com&A123e4567-e12b-12d1-a456-426655440000`.

2.2 Dados das transações

- **CPF origem**: CPF do cliente que enviou o dinheiro através de uma chave PIX;
- **CPF destino**: CPF do cliente que recebeu o dinheiro através de uma chave PIX;
- ***Timestamp***: data e hora em que a transação ocorreu no formato `AAAAMMDDHHmmSS`. Ex: `20210318143000`;
- **Valor da transação**: no formato `<9999999999>.<99>`. Ex: `0000000042.00`.

O identificador único de uma transação é uma chave composta pelo CPF de origem e *timestamp*. Garantidamente, nenhum campo de texto receberá caracteres acentuados.

2.3 Modelo Relacional e DDL

A base de dados será mantida em forma de arquivos, porém se fosse observar um modelo relacional, seria equivalente ao observado na Figura 1.

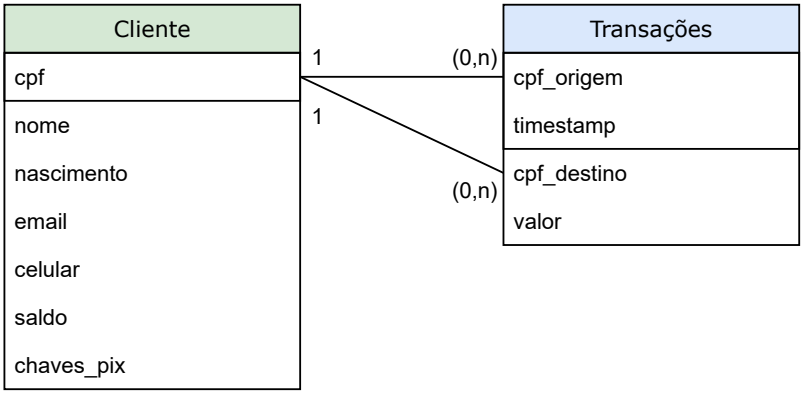


Figura 1: Modelo relacional das tabelas de transações e usuários da UFSPay

```
CREATE TABLE clientes (
  cpf          varchar(11) NOT NULL PRIMARY KEY,
  nome         text NOT NULL,
  nascimento   varchar(10) NOT NULL,
  email        text NOT NULL,
  celular      varchar(11) NOT NULL,
  saldo        numeric(12, 2) DEFAULT 0,
  chaves_pix   text[4] DEFAULT '{}'
);

CREATE TABLE transacoes (
  cpf_origem   varchar(11) NOT NULL,
  cpf_destino  varchar(11) NOT NULL,
  valor        numeric(12, 2) NOT NULL,
  timestamp    varchar(14) NOT NULL,
  PRIMARY KEY  (cpf_origem, timestamp)
);
```

Em uma implementação real, haveriam mecanismos adicionais para verificar a unicidade das chaves PIX antes da inserção de uma chave, atualização de saldo, adicionar o *timestamp* antes de inserir a transação, e situações de erro.

3 Operações suportadas pelo programa

Deve ser possível interagir com o programa através do console/terminal (modo texto) usando uma sintaxe similar à SQL, sendo que as operações a seguir devem ser fornecidas.

3.1 Cadastro

```
INSERT INTO clientes VALUES ('<CPF>', '<nome completo>', '<data de nascimento>',
'<email>', '<celular>');
```

Uma nova conta é criada. Seu programa deve ler os seguintes campos: CPF, nome completo, data de nascimento, e-mail e celular. Inicialmente, a conta não possui saldo (R$0,00) e não há nenhuma chave PIX cadastrada. Garantidamente, os campos serão fornecidos de maneira regular, não sendo necessário um pré-processamento da entrada. Não é permitido inserir usuários com um mesmo CPF cadastrado, portanto, caso já exista no sistema, deverá ser apresentada a mensagem padrão `ERRO_PK_REPETIDA`. Se a operação se concretizar, exiba a mensagem padrão `SUCESSO`.

Lembre-se de atualizar todos os índices necessários durante a inserção.

3.2 Remoção

```
DELETE FROM clientes WHERE cpf = '<CPF>';
```

O usuário deverá ser capaz de remover uma conta dado um CPF, sendo que remoção na base de dados deverá ser feita por meio de um marcador, conforme descrito na Seção 5. As chaves PIX cadastradas para este cliente também deverão ser removidas. Se a operação se concretizar, exiba a mensagem padrão `SUCESSO`. Caso a conta não exista, seu programa deverá exibir a mensagem padrão `ERRO_REGISTRO_NAO_ENCONTRADO`.

3.3 Depósito e saque

```
UPDATE clientes SET saldo = saldo + <valor> WHERE cpf = '<CPF do cliente>';
```

O usuário deverá ser capaz de depositar ou sacar dinheiro de uma conta informando seu CPF e o valor. Caso a conta para dado CPF não exista, seu programa deverá exibir a mensagem padrão `ERRO_REGISTRO_NAO_ENCONTRADO`. O valor é positivo em caso de depósito (ex: 25.40) e negativo em caso de saque (ex: -50.00). Caso o saque exceda o saldo disponível do cliente, a operação não poderá ser realizada e deverá ser apresentada a mensagem padrão `ERRO_SALDO_NAO_SUFICIENTE`. Caso o valor seja nulo (0.00), a operação não poderá ser realizada e deverá ser apresentada a mensagem padrão `ERRO_VALOR_INVALIDO`. Se a operação se concretizar, exiba a mensagem padrão `SUCESSO`.

3.4 Cadastro de chave PIX

```
UPDATE clientes SET chaves_pix = array_append(chaves_pix, '<tipo da chave PIX>')
WHERE cpf = '<CPF do cliente>';
```

O usuário deverá ser capaz de inserir uma nova chave PIX para um cliente. Seu programa deverá solicitar como entrada o CPF do cliente e o tipo da chave PIX (C: CPF, N: número de celular, E: email, A: chave aleatória) e o sistema deverá gerar a chave PIX com base nos dados do cliente. O cliente poderá ter uma única chave de cada tipo. Todas as chaves devem ser únicas. Caso o cliente não exista, deverá ser apresentada a mensagem padrão `ERRO_REGISTRO_NAO_ENCONTRADO`. Caso o usuário tente cadastrar chaves já cadastradas por outros clientes, deverá ser apresentada a mensagem padrão `ERRO_CHAVE_PIX_REPETIDA`. Caso o cliente informe um tipo inválido, deve ser apresentada a mensagem padrão `ERRO_TIPO_PIX_INVALIDO`. Caso o cliente tente cadastrar uma chave de um tipo que ele já possui, deverá ser apresentada a mensagem padrão `ERRO_CHAVE_PIX_DUPLICADA`. A chave PIX aleatória deverá ser gerada a partir da função `void new_uuid(char buffer[37])` fornecida no código base e com exemplo de uso. Se a operação se concretizar, exibir a mensagem padrão `SUCESSO`.

3.5 Remoção de chave PIX

```
UPDATE clientes SET chaves_pix = array_remove('<tipo da chave PIX>', chaves_pix)
WHERE cpf = '<CPF do cliente>';
```

O usuário deverá ser capaz de remover uma chave PIX. Caso ela não exista, seu programa deverá exibir a mensagem padrão `ERRO_REGISTRO_NAO_ENCONTRADO`. A chave PIX deve ser removida do registro do cliente de modo que a ordem das chaves PIX desse cliente no arquivo de dados seja igual após a remoção. Se a operação se concretizar, exiba a mensagem padrão `SUCESSO`.

3.6 Transferência por chave PIX

```
INSERT INTO transacoes VALUES ('<chave PIX origem>', '<chave PIX destino>',
<valor>);
```

O usuário deverá ser capaz de transferir dinheiro da sua conta para outra a partir das respectivas chaves PIX. Seu programa deverá solicitar como entrada a chave PIX da conta de origem, a chave PIX da conta de destino e o valor a ser transferido. Caso a chave não exista ou ambas as chaves pertençam ao mesmo CPF, deverá ser apresentada a mensagem padrão `ERRO_CHAVE_PIX_INVALIDA`. Caso a transferência exceda o saldo disponível do cliente de origem, deve ser apresentada a mensagem padrão `ERRO_SALDO_NAO_SUFICIENTE`. Caso o valor seja menor ou igual a zero, deve ser apresentada a mensagem padrão `ERRO_VALOR_INVALIDO`. Para concretizar a operação, será necessário atualizar os saldos de ambos os clientes e obter o *timestamp* do momento que ocorreu a transação utilizando a função `void current_timestamp(char buffer[15])` fornecida no código base e com exemplo de uso. Se a operação se concretizar, exibir a mensagem padrão `SUCESSO`.

3.7 Busca

O usuário poderá buscar por clientes e transações pela(s) chave(s).

Antes de imprimir os resultados das buscas nos índices do tipo Árvore-B (`transacoes_idx` e `timestamp_cpf_origem_idx`), deverão ser impressos os RRNs dos nós percorridos. Exemplo:

```
Nos percorridos: 4 7 2 3
```

Para os índices do tipo tabela *hash* (`clientes_idx` e `chaves_pix_idx`), deverão ser impressos os índices percorridos na tabela. Exemplo:

```
Indices percorridos: 9 10 0 1 2 (clientes_idx)
Indices percorridos: 22 0 1 2 3 (chaves_pix_idx)
```

As seguintes operações de busca por clientes e transações deverão ser implementadas.

3.7.1 Clientes

O usuário deverá ser capaz de buscar clientes pelos seguintes atributos:

(a) Por CPF:
`SELECT * FROM clientes WHERE cpf = '<CPF>';`

Solicitar ao usuário o CPF do cliente. Caso a conta não exista, seu programa deverá exibir a mensagem padrão `ERRO_REGISTRO_NAO_ENCONTRADO`. Caso a conta exista, todos os seus dados deverão ser impressos na tela de forma formatada.

(b) Por chave PIX:
`SELECT * FROM clientes WHERE '<chave PIX>' = ANY (chaves_pix);`

Solicitar ao usuário uma chave PIX. Caso a chave não exista, seu programa deverá exibir a mensagem padrão `ERRO_REGISTRO_NAO_ENCONTRADO`. Caso a chave exista, todos os dados do cliente que possui a chave deverão ser impressos na tela de forma formatada.

3.7.2 Transações

O usuário deverá ser capaz de buscar transações pelos seguintes atributos:

(a) Por CPF de origem e data específica:
`SELECT * FROM transacoes WHERE cpf_origem = '<CPF>' AND timestamp = '<data>';`

Solicitar ao usuário um CPF de origem e uma data no formato `AAAAMMDDHHmmSS`. Caso a transação não exista, seu programa deverá exibir a mensagem padrão `ERRO_REGISTRO_NAO_ENCONTRADO`. Caso a transação exista, todos os dados da transação deverão ser impressos na tela de forma formatada.

3.8 Listagem de transações

(a) Por período:

```
SELECT * FROM transacoes WHERE timestamp BETWEEN '<data início>' AND '<data fim>'
ORDER BY timestamp ASC;
```

Exibe todas as transações realizadas em um período de tempo (`timestamp` entre `<data início>` e `<data fim>`), em ordem cronológica. Caso nenhum registro for retornado, seu programa deverá exibir a mensagem padrão `AVISO_NENHUM_REGISTRO_ENCONTRADO`.

(b) Por CPF de origem e período:

```
SELECT * FROM transacoes WHERE cpf_origem = '<CPF>'
AND timestamp BETWEEN '<data início>' AND '<data fim>'
ORDER BY timestamp ASC;
```

Exibe todas as transações realizadas por um CPF (`cpf_origem`) em um período de tempo (`timestamp` entre `<data início>` e `<data fim>`), em ordem cronológica. Caso nenhum registro for retornado, seu programa deverá exibir a mensagem padrão `AVISO_NENHUM_REGISTRO_ENCONTRADO`.

3.9 Liberar espaço

```
VACUUM clientes;
```

O arquivo de dados `ARQUIVO_CLIENTES` deverá ser reorganizado com a remoção física de todos os registros marcados como excluídos e os índices deverão ser atualizados. A ordem dos registros no arquivo "limpo" não deverá ser diferente do arquivo "sujo". Se a operação se concretizar, exibir a mensagem padrão `SUCESSO`.

3.10 Imprimir arquivos de dados

O sistema deverá imprimir os arquivos de dados da seguinte maneira:

(a) Dados dos clientes:
`\echo file ARQUIVO_CLIENTES`

Imprime o arquivo de dados de clientes. Caso esteja vazio, apresentar a mensagem padrão `ERRO_ARQUIVO_VAZIO`;

(b) Dados das transações:
`\echo file ARQUIVO_TRANSACOES`

Imprime o arquivo de dados de transações. Caso esteja vazio, apresentar a mensagem padrão `ERRO_ARQUIVO_VAZIO`.

3.11 Imprimir índices

O sistema deverá imprimir os índices da seguinte maneira:

(a) Índice de clientes:
`\echo index clientes_idx`

Imprime o índice primário para clientes (`clientes_idx`). Caso o índice esteja vazio, imprimir `ERRO_ARQUIVO_VAZIO`;

(b) Índice de transações:
`\echo index transacoes_idx`

Imprime o arquivo de índice primário para transações (`transacoes_idx`). Caso o arquivo esteja vazio, imprimir `ERRO_ARQUIVO_VAZIO`;

(c) Índice de chaves PIX:
`\echo index chaves_pix_idx`

Imprime índice secundário para chaves PIX (`chaves_pix_idx`). Caso o índice esteja vazio, imprimir `ERRO_ARQUIVO_VAZIO`;

(d) Índice de *timestamp*:
`\echo index timestamp_cpf_origem_idx`

Imprime o arquivo de índice secundário para o *timestamp* e CPF do cliente de origem das transações (`timestamp_cpf_origem_idx`). Caso o arquivo esteja vazio, imprimir `ERRO_ARQUIVO_VAZIO`.

Indica se o comando SQL de entrada deve ser impresso na saída.

3.12 Finalizar

`\q`
Libera memória eventualmente alocada e encerra a execução do programa.

4 Arquivos de dados

Novamente, nenhum arquivo ficará salvo em disco. O arquivo de dados e os de índices serão simulados em *strings* e os índices serão sempre criados na inicialização do programa e manipulados em memória RAM até o término da execução. Suponha que há espaço suficiente em memória RAM para todas as operações.

Deve-se utilizar as variáveis globais `ARQUIVO_CLIENTES` e `ARQUIVO_TRANSACOES` e as funções de leitura e escrita em *strings*, como `sprintf` e `sscanf`, para simular as operações de leitura e escrita em arquivo. Os arquivos de dados devem ser no formato ASCII (arquivo texto).

`ARQUIVO_CLIENTES`: deverá ser organizado em registros de tamanho fixo de 256 *bytes* (256 caracteres). Os campos nome, e-mail e chaves PIX devem ser de tamanho variável. Os demais campos devem ser de tamanho fixo: CPF (11 *bytes*), data de nascimento (10 *bytes*), celular (11 *bytes*) e saldo (13 *bytes*). Portanto, os campos de tamanho fixo de um registro ocuparão 45 *bytes*. Os campos do registro devem ser separados pelo caractere delimitador '`;`' (ponto e vírgula), cada registro terá 7 delimitadores (um para cada campo). O campo multivalorado será separado pelo caractere '`&`'. Caso o registro tenha menos de 256 *bytes*, o espaço remanescente deverá ser preenchido com o caractere '`#`' de forma a completar os 256 *bytes*. Como são 45 *bytes* fixos + 7 *bytes* de delimitadores, então os campos variáveis devem ocupar no máximo 204 *bytes* (incluindo os delimitadores do campo multivalorado), para evitar que o registro exceda 256 *bytes*.

Exemplo de arquivo de dados de clientes:

```
44535687915;JOSE AUGUSTO;12/11/1951;joseaugusto@hotmail.com;1599
6587458;0000042580.80;C44535687915&Ejoseaugusto@hotmail.com;####
############################################################
############################################################
14578965815;MARIANA DIAS;18/05/1995;marianadias@gmail.com;159817
54878;0000021850.00;N15981754878&C14578965815;##################
############################################################
############################################################
34672961815;CARLA AMARAL;20/03/1990;carlaamaral@gmail.com;159917
34220;0000311050.20;;#########################################
############################################################
############################################################
```

`ARQUIVO_TRANSACOES`: o arquivo de transações deverá ser organizado em registros de tamanho fixo de 49 *bytes* (49 caracteres). Todos os campos possuem tamanho fixo: CPF de origem (11 *bytes*), CPF de destino (11 *bytes*), valor (13 *bytes*) e *timestamp* (14 *bytes*).

Exemplo de arquivo de dados das transações:

```
44535687915 14578965815 0000000025.40 20150203142345
44535687915 14578965815 0000000142.60 20150405123522
14578965815 44535687915 0000001045.90 20150615114802
```

Como os campos de transações possuem tamanho fixo, não são necessários delimitadores (foram inseridos espaços para maior clareza).

Note que não há quebras de linhas nos arquivos (elas foram inseridas aqui apenas para facilitar a visualização da sequência de registros).

5 Instruções para as operações com os registros

- **Inserção**: cada cliente e transação devem ser inseridos no final de seus respectivos arquivos de dados, e atualizados os índices.

- **Remoção**: o registro deverá ser localizado acessando o índice primário. A remoção deverá colocar o marcador `*|` nas primeiras posições do registro removido. O espaço do registro removido não deverá ser reutilizado para novas inserções. Observe que o registro deverá continuar ocupando exatamente 256 *bytes*. Além disso, no índice primário, o registro deverá ser marcado como excluído.

- **Atualização**: o único campo alterável é o saldo. O registro deverá ser localizado acessando o índice primário e o novo saldo deverá ser atualizado no registro na mesma posição em que está (não deve ser feita remoção seguida de inserção). Note que o campo de saldo sempre ocupará 13 *bytes*.

6 Criação de índices

No cenário atual, há um grande volume de dados e alguns índices não cabem em RAM, portanto, para simular essa situação, as únicas informações que você deverá manter todo tempo em memória é o RRN da raiz de cada Árvore-B e as tabelas *hash*. Assuma que um nó do índice de Árvore-B corresponde a uma página e, portanto, cabe no *buffer* de memória. Dessa forma, trabalhe apenas com a menor quantidade de nós necessários das árvores por vez, pois isso implica em reduzir a quantidade de *seeks* e de informação transferida entre as memórias primária e secundária. **É terminantemente proibido implementar os índices em Tipos Abstratos de Dados (TADs) que não os especificados neste documento**.

Todas as árvores terão a mesma ordem (m) e **a promoção deverá ser sempre pelo sucessor imediato** (menor chave da sub-árvore à direita). Todo novo nó criado deverá ser inserido no final do respectivo arquivo de índice.

Cada tabela *hash* terá um tamanho (T) e, em ambas as versões, a função de *hash* será dada por:

$$h(k) = [\sum_{i=1}^{min(n,\ 9)} {k_i}^i] \ \ mod \ \ T$$

ou seja, para uma chave de tamanho n

$$h(k) = [{k_1}^1 + {k_2}^2 + {k_3}^3 + ... + {k_n}^n] \ \ mod \ \ T$$

onde:

$h(k)$ = função de *hash*
k = chave com n caracteres
T = tamanho da tabela *hash*. Deve ser obrigatoriamente um número primo
k_i = caractere na posição i

6.1 Índices primários

6.1.1 `clientes_idx`

Índice primário, do tipo tabela *hash*, que contém o CPF do cliente (chave primária) e o RRN do respectivo registro no arquivo de dados. A função de *hash* é aplicada sobre os valores de cada

número que compõem o CPF do cliente. A técnica para tratamento de colisões aplicada neste índice é a de **endereçamento aberto** com **reespalhamento linear**.

Por exemplo, considere a tabela *hash* de tamanho 11 a seguir:

```
 [0] {55465467898, 0000}
 [1] {27184728937, 0004}
 [2] {78657265480, 0006}
 [3] {12478147955, 0001}
 [4] {56759676675, 0005}
 [5] {43562538970, 0002}
 [6] {23758975870, 0007}
 [7] {23555875892, 0008}
 [8] {23798479315, 0009}
 [9] {76486446896, 0003}
[10]
```

Observe que, no exemplo, a tabela é limitada a 11 entradas. À medida que o número de clientes cresce, a quantidade de colisões aumenta prejudicando o desempenho do sistema. No limite, a tabela pode ficar totalmente cheia e o número de colisões tenderá ao infinito. Por conta disso, é necessário criar um mecanismo para monitorar o desempenho e reconstruir a tabela caso necessário. Este mecanismo funcionará da seguinte forma:

- Por questões de desempenho, o tamanho da tabela *hash* precisará ser aumentado quando o número médio de comparações $C(n) > 3$, sendo que $C(n) = \frac{(2-\alpha)}{(2-2\alpha)}$ e:

  $\alpha = \frac{n}{T}$ = fator de carga $(0 \leq \alpha \leq 1)$
  n = número de chaves na tabela *hash*
  T = tamanho da tabela *hash*

- Quando $C(n) > 3$, será necessário redimensionar o tamanho da tabela para o próximo primo maior que o dobro do tamanho atual. Todas as chaves na tabela deverão ser espalhadas na tabela redimensionada usando a função de *hash* atualizada.

6.1.2 `transacoes_idx`

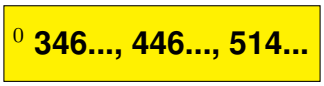
Índice primário do tipo Árvore-B que contém o CPF de origem da transação (em ordem crescente) e o *timestamp* (em ordem cronológica), e o RRN respectivo do registro no arquivo de transações.

Cada registro da Árvore-B para o índice `transacoes_idx` é composto por:

- *Contador de chaves:* 3 bytes para a quantidade de chaves;
- *Chaves ordenadas:* $(m-1)$ * (25 bytes de chave primária [11 do CPF + 14 do *timestamp*] + 4 bytes para o RRN do arquivo de dados) bytes para armazenas as chaves. Para as chaves não usadas, preencha todos os bytes com '`#`';
- *Indicador de folha:* 1 byte para indicar se o nó é folha '`T`' (True) ou não '`F`' (False);

- *Apontadores para os filhos:* (m * 3 bytes para cada RRN filho) bytes para indicar os RRNs dos nós descendentes do nó atual. Note que esse RRN se refere ao próprio arquivo de índice primário. Utilize '`***`' para indicar que aquela posição do vetor de descendentes é nula.

Exemplo de arquivo de índice primário de transações representado por uma Árvore-B de ordem 4, após a inserção das chaves na ordem: '`44678965437 20210325091500 0002`', '`34678965321 20210912151010 0000`' e '`51478965098 20210101183012 0001`'.




```
003 | 34678965321 20210912151010 0002 | 44678965437 20210325091500 0000 |
      51478965098 20210101183012 0001 | T | *** *** *** ***
```

Em resumo, um nó da Árvore-B de transações, de ordem m, possui as seguintes informações:

```
{QUANTIDADE DE CHAVES}
{CPF_1} {TIMESTAMP_1} {RRN_1}
{CPF_2} {TIMESTAMP_2} {RRN_2}
...
{CPF_(m-1)} {TIMESTAMP_(m-1)} {RRN_(m-1)}
{FOLHA}
{RRN FILHO_1} {RRN FILHO_2} ... {RRN_FILHO_m}
```

Novamente, não há quebras de linhas no arquivo. Elas foram inseridas apenas para exemplificar a sequência de registros.

6.2 Índices secundários

6.2.1 `chaves_pix_idx`

Este índice secundário é do tipo tabela *hash* e contém a chave PIX do cliente e seu CPF (chave primária). A técnica aplicada para solucionar as colisões neste índice é a de **encadeamento** e a lista encadeada deve ser ordenada pela chave PIX. A função de *hash* é aplicada sobre os valores dos códigos ASCII de cada caractere que compõem a chave de *hash*.

Por exemplo, considere a tabela *hash* de tamanho 11 a seguir:

```
 [0]
 [1]
 [2] {15998765645, 66545678765}
 [3] {ga.augusto@gmail.com, 99876556782}, {galmeida@gmail.com, 54654367865}
 [4] {343b6d0b-2c58-4423-b759-d987cb9d25d1, 66545678765}
     {ge.santana@gmail.com, 44565434213}
 [5] {44565434213, 44565434213}
 [6]
 [7] {14565436782, 14565436782}
 [8] {14998786754, 54654367865}, {mariaeugenia@gmail.com, 66545678765}
 [9] {99876556782, 99876556782}, {melissa@gmail.com', 14565436782}
[10] {66545678765, 66545678765'}
```

- Por questões de desempenho, o tamanho da tabela *hash* precisará ser aumentado quando o número médio de comparações $C(n) > 2$, sendo que $C(n) = 1 + \frac{\alpha}{2}$ e:

  $\alpha = \frac{n}{T}$ = fator de carga ($\alpha > 0$)
  n = número de chaves na tabela *hash*
  T = tamanho da tabela *hash*

- Quando $C(n) > 2$, será necessário redimensionar o tamanho da tabela para o próximo primo maior que o dobro do tamanho atual. Todas as chaves na tabela deverão ser espalhadas na tabela redimensionada usando a função de *hash* atualizada.

6.2.2 `timestamp_cpf_origem_idx`

Índice secundário do tipo Árvore-B que contém o *timestamp* das transações (em ordem cronológica) e o CPF do cliente de origem (em ordem crescente). Note que esses campos já compõem a chave primária da transação.

Cada registro da Árvore-B para o índice `timestamp_cpf_origem_idx` é composto por:

- *Contador de chaves:* 3 bytes para a quantidade de chaves;
- *Chaves ordenadas:* $(m - 1)$ * (25 bytes de chave primária [14 do *timestamp* + 11 do CPF]) bytes para armazenar as chaves. Para as chaves não usadas, preencha todos os bytes com '`#`';
- *Indicador de folha:* 1 byte para indicar se o nó é folha '`T`' (True) ou não '`F`' (False);
- *Apontadores para os filhos:* (m * 3 bytes para cada RRN filho) bytes para indicar os RRNs dos nós descendentes do nó atual. Note que assim como no índice primário, esse RRN se refere ao próprio arquivo de índice secundário. Utilize '`***`' para indicar que aquela posição do vetor de descendentes é nula.

Exemplo de arquivo de índice primário representado por uma Árvore-B de ordem 4, após a inserção das chaves na ordem: '`20210325091500 44678965437`', '`20210912151010 34678965321`' e '`20210101183012 51478965098`'.


```
003 | 20210101183012 51478965098 | 20210325091500 44678965437 |
      20210912151010 34678965321 | T | *** *** *** ***
```

Em resumo, um nó da Árvore-B de transações, de ordem m, possui as seguintes informações:

```
{QUANTIDADE DE CHAVES}
{TIMESTAMP_1} {CPF_1}
{TIMESTAMP_2} {CPF_2}
...
{TIMESTAMP_(m-1)} {CPF_(m-1)}
{FOLHA}
{RRN FILHO_1} {RRN FILHO_2} ... {RRN_FILHO_m}
```

Novamente, não há quebras de linhas no arquivo, espaços ou *pipes* ('`|`'), Eles foram inseridos apenas para exemplificar a sequência de registros.

7 Inicialização do programa

Para que o programa inicie corretamente, deve-se realizar o seguinte procedimento:

1. Inserir o comando `SET HASH_CLIENTES_IDX_ORDER <TAMANHO DA TABELA HASH>;` para informar o tamanho do índice *hash* de clientes. Caso não informado, é definido como 503 por padrão;
2. Inserir o comando `SET HASH_CHAVES_PIX_IDX_ORDER <TAMANHO DA TABELA HASH>;` para informar o tamanho do índice *hash* de chaves PIX. Caso não informado, é definido como 1009 por padrão;
3. Inserir o comando `SET BTREE_ORDER <ORDEM DAS ÁRVORES-B>;` para informar a ordem das Árvores-B. Caso não informado, é definido como 3 por padrão;
4. Inserir o comando `SET ARQUIVO_CLIENTES '<DADOS DE CLIENTES>';` para informar os dados contidos no arquivo de clientes ou `SET ARQUIVO_CLIENTES '';` caso o arquivo esteja vazio;
5. Inserir o comando `SET ARQUIVO_TRANSACOES '<DADOS DE TRANSAÇÕES>';` para informar os dados contidos no arquivo de transações ou `SET ARQUIVO_TRANSACOES '';` caso o arquivo esteja vazio;
6. Inicializar as estruturas de dados dos índices.

8 Implementação

Implemente suas funções utilizando o código-base fornecido. **Não é permitido modificar os trechos de código pronto ou as estruturas já definidas**. Ao imprimir um registro, utilize as funções `exibir_cliente(int rrn)` ou `exibir_transacao(int rrn)`.

Implementar rotinas que contenham obrigatoriamente as seguintes funcionalidades:

- Estruturas de dados adequadas para armazenar os índices do tipo Árvore-B em arquivos simulados por meio de *strings* e índices de tabela *hash* na memória principal;
- Verificar se os arquivos de dados existem;
- Criar os índices primários: deve refazer os índices primários a partir dos arquivos de dados;
- Criar os índices secundários: deve refazer os índices secundários a partir dos arquivos de dados;
- Inserir um registro: modifica os arquivos de dados e de índices;
- Buscar por registro: busca um registro por sua(s) chave(s) primária(a);
- Alterar um registro: modifica o campo do registro diretamente no arquivo de dados;
- Remover um registro: modifica o arquivo de dados e o índice primário na memória principal;
- Listar registros: listar os registros ordenados pelo intervalo de chaves dadas.

9 Resumo de alterações em relação ao T02

Para facilitar o desenvolvimento do T03, é possível reutilizar as funções já implementadas no T01 e T02, com ênfase nas seguintes alterações:

1. A operação de listagem de clientes foi removida;
2. As operações de remoção de clientes e liberar espaço foram adicionadas novamente;
3. Agora é possível remover as chaves PIX de um cliente;
4. Os índices `transacoes_idx` e `timestamp_cpf_origem_idx` continuam sendo do tipo Árvore-B, e `clientes_idx` e `chaves_pix_idx` foram substituídos por tabelas *hash*;
5. Os tamanhos das tabelas *hash* são informados na inicialização do programa;
6. As tabelas *hash* serão manipuladas integralmente na memória RAM.

10 Dicas

- Ao ler uma entrada, tome cuidado com caracteres de quebra de linha (`\n`) não capturados;
- Você nunca deve perder a referência do começo do arquivo, então não é recomendável percorrer a *string* diretamente pelo ponteiro `ARQUIVO`. Um comando equivalente a `fseek(f, 256, SEEK_SET)` é `char *p = ARQUIVO + 256`;
- Diferentemente do `fscanf`, o `sscanf` não movimenta automaticamente o ponteiro após a leitura;
- O `sprintf` adiciona automaticamente o caractere `\0` no final da *string* escrita. Em alguns casos você precisará sobrescrever a posição manualmente. Você também pode utilizar o comando `strncpy` para escrever em *strings*, esse comando, diferentemente do `sprintf`, não adiciona o caractere nulo no final;
- A função `strtok` permite navegar nas *substrings* de uma certa *string* dado o(s) delimitador(es). Porém, tenha em mente que ela deve ser usada em uma cópia da *string* original, pois ela modifica o primeiro argumento.

Simplicity, carried to the extreme, becomes elegance.
(Simplicidade, levada ao extremo, torna-se elegância)
— Jon Franklin