



TRABALHO 01

INDEXAÇÃO

Prazo para entrega: 21/04/2021 – 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_T01.c, ex: 123456_ORI_T01.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 história do sistema bancário começou quando os impérios precisavam de uma forma de pagar por bens e serviços estrangeiros com algo que pudesse ser trocado com mais facilidade. Moedas de vários tamanhos e metais diferentes serviam no lugar de cédulas de papel frágeis e impermanentes.

No entanto, essas moedas precisavam ser guardadas em um lugar seguro. Como as casas da antiguidade não possuíam o benefício de um cofre, a maioria das pessoas ricas guardava moedas em seus templos. Numerosas pessoas, como sacerdotes e oficiantes religiosos, que se esperava que fossem devotos e honestos, sempre ocuparam os templos, acrescentando uma sensação de segurança. Registros históricos da Grécia, Roma, Egito e da antiga Babilônia sugerem que os templos emprestavam dinheiro, além de mantê-lo seguro, atuando como centros financeiros primitivos.

Hoje, cerca de 4.000 anos depois, o sistema bancário está muito mais modernizado (*apesar de ainda possuir incontáveis linhas em COBOL*). Nos últimos anos, a evolução do sistema financeiro foi marcada pelo advento do *internet banking*, *fintechs* e, recentemente no Brasil, do PIX, sistema que permite pagamentos e transferências instantâneas gratuitamente entre contas bancárias de quaisquer instituições financeiras brasileiras.

Pensando nisso, você, estudante com enorme potencial de sucesso, vendo este segmento de mercado promissor, teve a brilhante ideia de desenvolver um forte concorrente aos bancos brasileiros (e, futuramente, internacionais): a **UFSPay**, *fintech* brasileira com foco no público universitário.

Antes de colocar esse grande projeto em produção, é necessário o desenvolvimento de um MVP, o produto mínimo viável. Isso se resume em empregar suas habilidades em C para manter a base de dados do sistema de transações e gerenciamento de usuários.

2 Base de dados da aplicação

O sistema será 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. 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 AAAAMDDHHmmSS.
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](#).

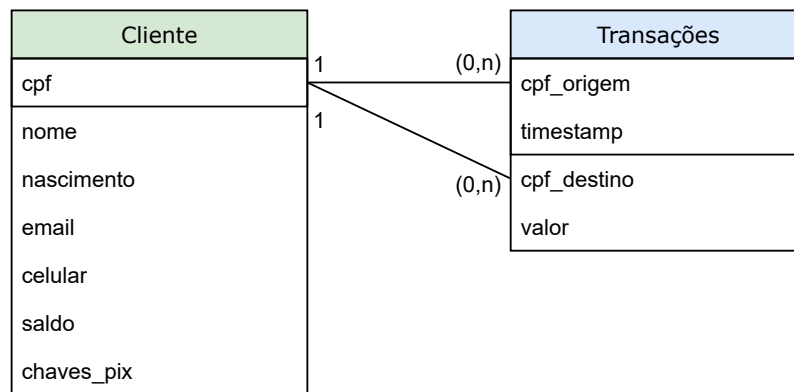


Figura 1: Modelo relacional das tabelas de transações e usuários da UFSPay

```
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. Caso a conta não exista, seu programa deverá exibir a mensagem padrão ERRO_REGISTRO_NAO_ENCONTRADO. A remoção na base de dados deverá ser feita por meio de um marcador, conforme descrito na [Seção 7](#). Em adicional, as chaves PIX cadastradas para este cliente também devem ser removidas. Se a operação se concretizar, exiba a mensagem padrão SUCESSO.

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 apresentado 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 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 ou o cliente que possui a chave foi removido, 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`.

3.6 Busca

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

3.6.1 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.

3.6.2 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 `AAAAMDDHhmmSS`. 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.

3.7 Listagem

As seguintes operações de listagem de clientes e transações deverão ser implementadas.

3.7.1 Clientes

(a) Por CPF:

```
SELECT * FROM clientes ORDER BY cpf ASC;
```

Exibe todos os clientes ordenados de forma crescente pelo CPF. Caso nenhum registro for retornado, seu programa deverá exibir a mensagem padrão `AVISO_NENHUM_REGISTRO_ENCONTRADO`.

3.7.2 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.

3.8 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.

3.9 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.

3.10 Imprimir índices secundários

O sistema deverá imprimir os índices secundários da seguinte maneira:

(a) Índice de chaves PIX:

```
\echo index chaves_pix_index
```

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

(b) Índice de transações:

```
\echo index timestamp_cpf_origem_index
```

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

3.11 Finalizar

```
\q
```

Libera memória e encerra a execução do programa.

4 Criação de índices

Para que as buscas e as listagens ordenadas desses dados sejam otimizadas, é necessário criar e manter índices em memória (que serão liberados ao término do programa).

Pelo menos os seguintes índices deverão ser criados:

4.1 Índices primários

- `clientes_index`: índice primário que contém o CPF do cliente (chave primária) e o RRN do respectivo registro no arquivo de dados, ordenado pela chave primária;
- `transacoes_index`: índice primário 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.

4.2 Índices secundários

- `chaves_pix_index`: índice secundário que contém as chaves PIX de todos os clientes (em ordem lexicográfica) e a chave primária do respectivo cliente.
- `timestamp_cpf_origem_index`: índice secundário 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.

Deverá ser desenvolvida uma rotina para a criação de cada índice. Os índices serão sempre criados e manipulados em memória principal na inicialização e liberados ao término do programa. Note que o ideal é que os índices primários sejam criados primeiro, depois os secundários.

5 Arquivos de dados

Como este trabalho será corrigido automaticamente por um juiz online que não aceita funções que manipulam arquivos, os registros serão armazenados e manipulados em *strings* que irão simular os arquivos abertos. Para isso, você deverá 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*.

`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*).

5.1 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;;#####
#####
#####

```

5.2 Exemplo de arquivo de dados das transações

```

44535687915145789658150000000025.4020150203142345
44535687915145789658150000000142.6020150405123522
14578965815445356879150000001045.9020150615114802

```

Como os registros possuem tamanho fixo, não necessitam ser armazenados com delimitadores. Porém, esse seria o registro equivalente, caso existissem delimitadores (para maior clareza):

```

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

```

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

6 Inicialização do programa

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

1. Inserir o comando `SET ARQUIVO_CLIENTES TO '<DADOS DE CLIENTES>'`; para informar os dados contidos no arquivo de clientes ou `SET ARQUIVO_CLIENTES TO ''`; caso o arquivo esteja vazio;
2. Inserir o comando `SET ARQUIVO_TRANSACOES TO '<DADOS DE TRANSAÇÕES>'`; para informar os dados contidos no arquivo de transações ou `SET ARQUIVO_TRANSACOES TO ''`; caso o arquivo esteja vazio;
3. Inicializar as estruturas de dados dos índices.

7 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 RRN correspondente ao registro removido deverá ser substituído por -1.
- **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*.

8 Implementação

Implementar rotinas que contenham obrigatoriamente as seguintes funcionalidades:

- Estruturas de dados adequadas para armazenar os índices 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 os índices na memória principal;
- Buscar por registro: busca pela chave primária ou por uma das chaves secundárias;
- 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 todos os registros ordenados pela chave primária ou por uma das chaves secundárias;
- Liberar espaço: organizar o arquivo de dados e refazer os índices.

9 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;
- É permitido (e recomendado) utilizar as funções `qsort` e `bsearch`. Leia atentamente a documentação antes de usá-las;
- Para o funcionamento ideal do seu programa, é necessário utilizar a busca binária, porém tenha em mente que ela não garante que irá retornar o primeiro elemento do tipo, caso haja repetição.

An algorithm must be seen to be believed.

(Um algoritmo deve ser visto para que seja acreditado)

— *Donald Knuth*