Função ~~delete~~

Nicolas;
Vitor;
Willian.

```c
strcpy(sqlCopy, sql);
                    (char [2])" "
token = strtok(sqlCopy, " ");

while(token != NULL) { ////enquanto existir token
    // Se i == 2, o token vai ser o nome da tabela
    if(i == 2) {
        strcpy(tableName, token);
    } else if(i == 5) { // se i == 5, é o atributo
        strcpy(attributeName, token);
    } else if(i == 8) {
        break;
    } else
        token = strtok(NULL, " ");
      printf("Token: %s | I: %d\n", token, i);
    i++;
}
token = strtok(token, "\n");
```

- Através da query sql, encontramos a tabela onde deve ocorrer a remoção e qual atributo será usado como avaliação.

Query	delete	from	teste	where	id	=	2
Valor I	0	1	2 - 3	4	5 - 6	7	8

```c
snprintf(pageName, sizeof(pageName), "%s/header.dat", tableName); /

FILE *headerPage = fopen(pageName, "rb+"); //abre o arquivo em modo
if(!headerPage) { //se não conseguiu abrir o arquivo de cabeçalho
    printf("Can not read %s/header.dat\n", tableName);
}

fread(&qtdFields, sizeof(int), 1, headerPage); // le o numero de ca
```

- Encontrada a tabela onde deve ocorrer a remoção, é feita a tentativa de abrir o arquivo.
- Se ocorrer falha (arquivo não existir) a leitura será cancelada.
- Com os 4 primeiros bytes do arquivo header é obtido a quantidade de campos presente na tabela.

```
for(int i = 0; i < qtdFields; i++) { //laço para percorrer os
    fread(attributes[i].name, 15, 1, headerPage);
    fread(&attributes[i].size, sizeof(int), 1, headerPage);
    fread(&attributes[i].type, 1, 1, headerPage);
}
printf("\n");
```

```
b'\x03\x00\x00\x00a\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x04\x00\x00\x00I
b\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x04\x00\x00\x00Ic\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x04\x00\x00\x00I\x01\x00\x00\x00'
```

- Possuindo a quantidade de campos da tabela, percorremos o arquivo header coletando as informações sobre cada campo.

```
snprintf(pageName, sizeof(pageName), "%s/page%d.dat", tableName, numPage);
FILE *page = fopen(pageName, "rb+"); // abre a pagina

header head;

item items;
fread(&head.memFree, sizeof(int), 1, page); // espaço disponivel na pagina
fread(&head.next, sizeof(int), 1, page); // posição da proxima pagina
fread(&head.qtdItems, sizeof(int), 1, page); // quantidade de itens na pagi
```

- Abre a página 1, que contém o espaço disponível na página nos primeiros 4 bytes, seguido da posição do próximo item e da quantidade de itens.

```
\xbe\x1f\x00\x00\xc9\x1f\x00\x00\x01\x00\x00\x00
```

Memória livre: 1fbe -> 8126

Próximo item: 1fc9 -> 8137

Quantidade de itens: 1 -> 1

```
for(int i = 0; i < head.qtdItems; i++) {
    fseek(page, 0, SEEK_SET); // posiciona no inicio do arquivo
    moveItem = sizeof(item) * (i + 1); // define a posição do item
    fseek(page, moveItem, SEEK_SET); // posiciona o ponteiro do arquivo no item

    fread(&readItem.offset, sizeof(int), 1, page); // le e armazena o offset
    fread(&readItem.totalLen, sizeof(int), 1, page);
    fread(&readItem.writed, sizeof(int), 1, page);

    if(readItem.writed == 0)
        continue;

    fseek(page, readItem.offset, SEEK_SET);
```

- Percorrendo todos os itens, vamos à posição do item e coletamos as informações de deslocamento, tamanho e escrita.
- Caso o item ainda não tenha sido removido, será posicionado sobre a tupla.

```
for(int j = 0; j < qtdFields; j++) {
    delete = 1;
```

- Já estando sobre a tupla, deve-se percorre-lá para verificar se será removida ou não.
- Para isso, é verificado todos os campos da tupla.

```c
if(attributes[j].type == 'C') {
    charInFile = ' ';
    stopChar = 0;
    if(j > 0)
        fread(&charInFile, 1, 1, page); // Read '\0' varchar
    // percorre até o final do valor('\0' delimita fim do char)
    for(int l = 0; l < attributes[j].size; l++) {
        fread(&charInFile, 1, 1, page);
        if(delete == 1 && charInFile == '\0')
            stopChar = 1;
        //printf("%c = %c\n", charInFile, token[l]);
        if(charInFile != token[l] && !stopChar)
            delete = 0;
    }
    if(delete && !(strcmp(attributeName, attributes[j].name))) {
        fseek(page, moveItem, SEEK_SET);
        readItem.writed = 0;
        fwrite(&readItem.offset, sizeof(int), 1, page);
        fwrite(&readItem.totalLen, sizeof(int), 1, page);
        fwrite(&readItem.writed, sizeof(int), 1, page);
        printf("Attribute deleted\n");
        count++;
        break;
    } else if(j > 0)
        fseek(page, -1, SEEK_CUR); // Read '\0' varchar
```

- Caso for do tipo CHAR será comparado byte-a-byte se o valor da query é o mesmo com o armazenado.
- Se for e o campo da query for o mesmo do armazenado, ele será deletado, que é escrever 0 no atributo “writed”.

```
}else if(attributes[j].type == 'I') {
    if(j > 0)
        fread(&charInFile, 1, 1, page); // Read '\0' varchar

    fread(&intInFile, sizeof(int), 1, page);
    //printf("From file: %d\n", intInFile);
    inputDelete = atoi(token); //inputDelete é o valor do where ->
    if(intInFile != inputDelete)
        delete = 0;
    if(delete && !(strcmp(attributeName, attributes[j].name))) {
        fseek(page, moveItem, SEEK_SET);
        readItem.writed = 0;
        fwrite(&readItem.offset, sizeof(int), 1, page);
        fwrite(&readItem.totalLen, sizeof(int), 1, page);
        fwrite(&readItem.writed, sizeof(int), 1, page);
        printf("Attribute deleted\n");
        count++;
        break;
    } else if(j > 0)
        fseek(page, -1, SEEK_CUR); // Read '\0' varchar
```

- Caso o campo for do tipo INT, haverá a conversão de string para int, em seguida é comparado se o valor da query é o mesmo do valor armazenado.
- Se forem iguais, assim como o nome do campo, a tupla será deletada sobre-escrevendo o campo “writed”.

```
} else if(attributes[j].type == 'V') {
    k = 0;
    if(j > 0)
        fread(&charInFile, 1, 1, page); // Read '\0' varchar
    while(charInFile != '$') {
        fread(&charInFile, 1, 1, page);
        //printf("%c = %c\n", charInFile, token[k]);
        if(delete == 1 && charInFile == '$')
            break;
        if(charInFile != token[k])
            delete = 0;
        k++;
    }
    if(delete && !(strcmp(attributeName, attributes[j].name))) {
        fseek(page, moveItem, SEEK_SET);
        readItem.writed = 0;
        fwrite(&readItem.offset, sizeof(int), 1, page);
        fwrite(&readItem.totalLen, sizeof(int), 1, page);
        fwrite(&readItem.writed, sizeof(int), 1, page);
        printf("Attribute deleted\n");
        count++;
        break;
    } else if(j > 0)
        fseek(page, -1, SEEK_CUR); // Read '\0' varchar
}
```

- Caso seja do tipo VARCHAR, será comparado o valor da query com o armazenado byte-a-byte até encontrar “$”.
- Se os valores forem iguais, assim como o nome dos campos, a tupla será deletada, re-escrevendo o valor “writed” para 0.

```
printf("Attributes deleted = %d\n", count);
fseek(page, 8191, SEEK_SET);
fread(&special, 1, 1, page);
if(special == '1') {
    numPage++;
    fclose(page);
    deleteFrom(sql, numPage);
} else
    fclose(page);
```

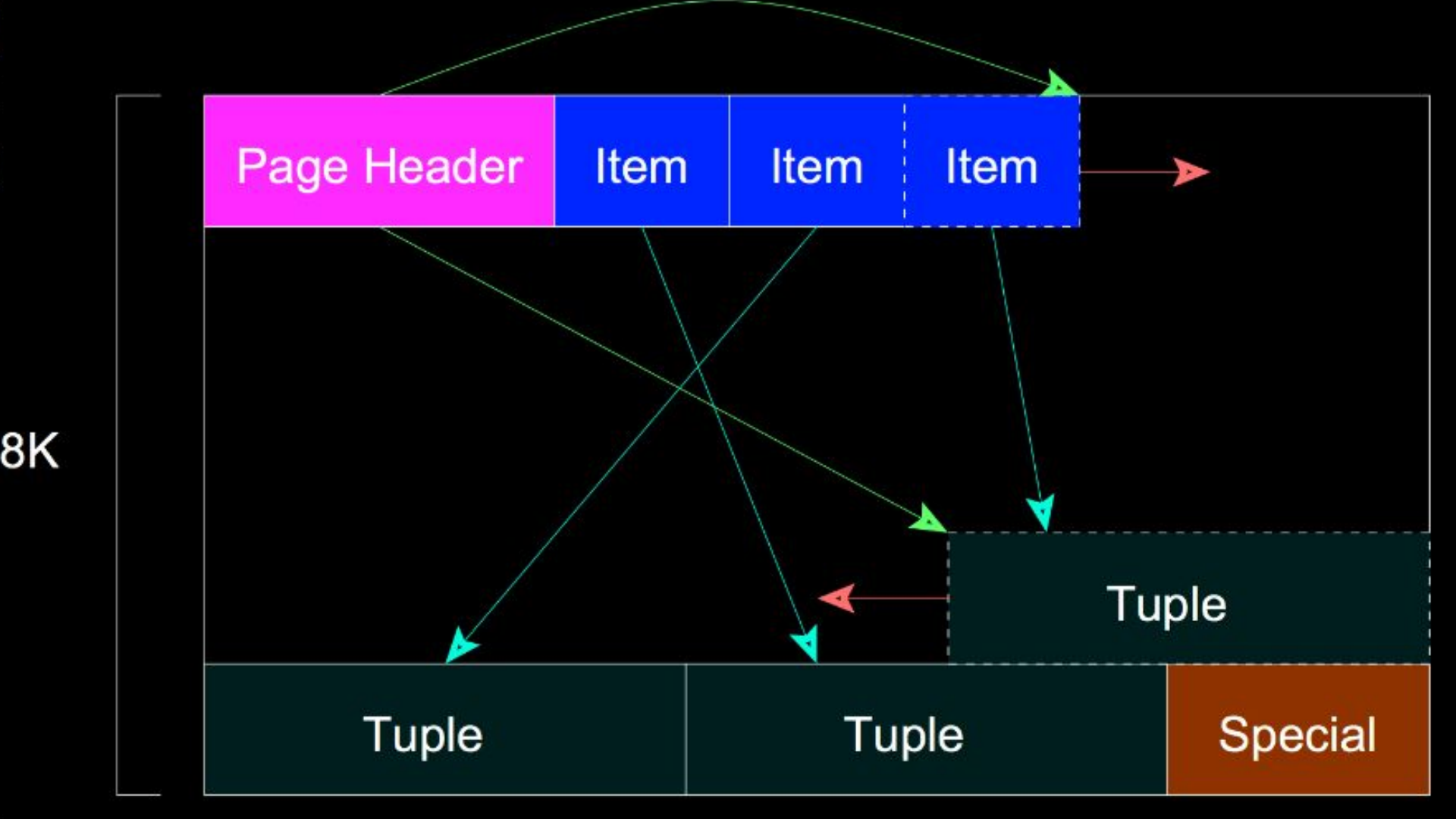
- Após varrer todos os itens da página é verificado se a tabela possui mais páginas com tuplas.
- Se houver é refeito as verificações lá, caso contrário o arquivo page será fechado e a função delete encerrada.

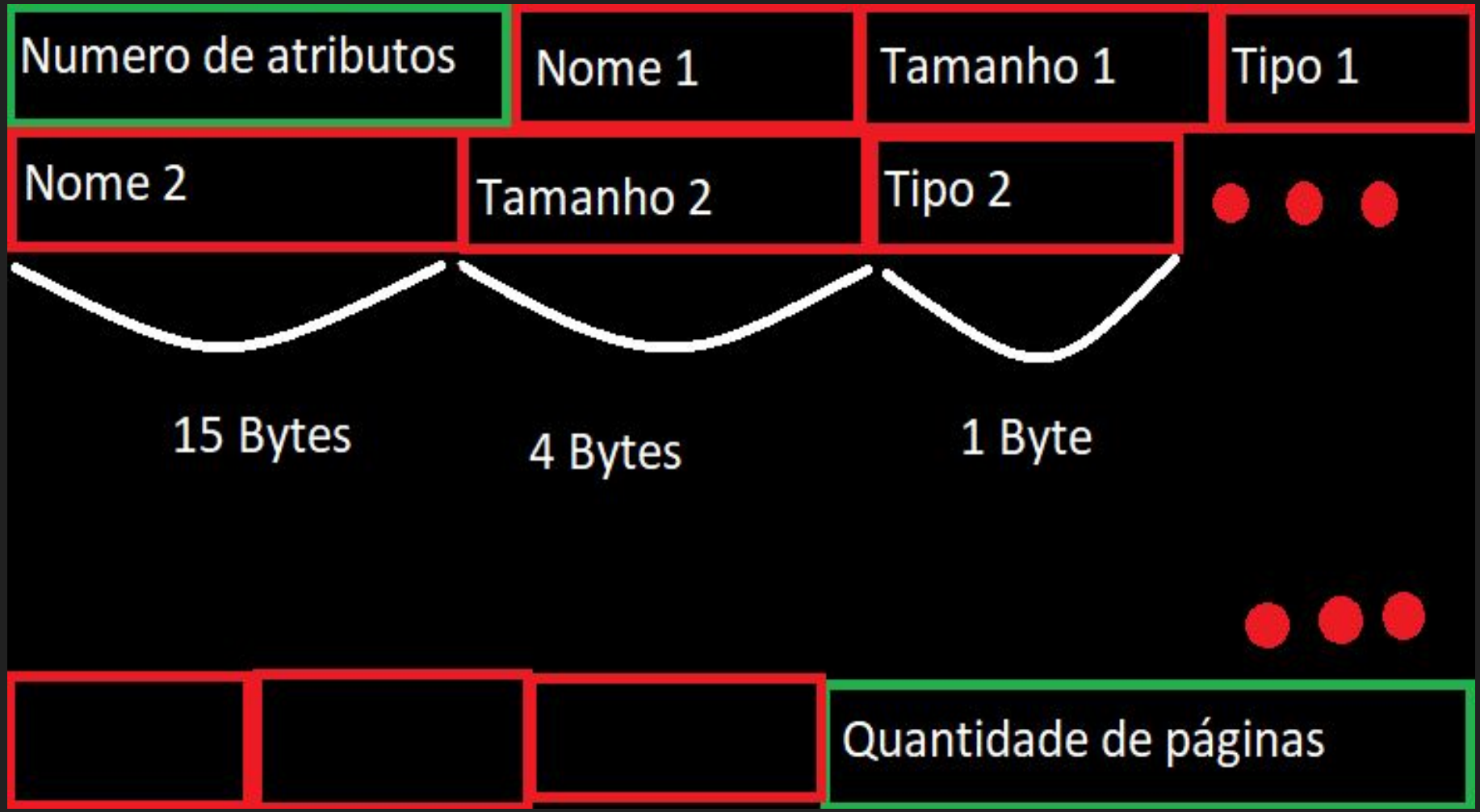
Bugs

- Usar um int na cláusula where quando o int não é o primeiro atributo causa inconsistências ao deletar;
- Queries inválidas (não há tratamento);

Sugestões de melhoria

- Deletar sem cláusula “where”;
- Deletar usando operadores diferente de ‘=’;
- Liberar o espaço do item/tupla.


Page Header
Item
Item
Item
8K
Tuple
Tuple
Tuple
Special


Numero de atributos
Nome 1
Tamanho 1
Tipo 1
Nome 2
Tamanho 2
Tipo 2
15 Bytes
4 Bytes
1 Byte
Quantidade de páginas