UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ROVIAN VOELZ VERONEZ

Busca por similaridade em uma base de dados de genealogia

Trabalho de Graduação.

Prof. Dr. Carlos Alberto Heuser
Orientador

Porto Alegre, junho de 2013.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

BD	Banco de Dados
HTML	HyperText Markup Language
UDF	User Defined Language
TNG	The Next Generation of Genealogy Sitebuilding
SQL	Structured Query Language
UML	Unified Modeling Language
JSON	JavaScript Object Notation
DBMS	DataBase Management System
PHP	PHP: Hypertext Processor
OMG	Object Management Group

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Na área da genealogia, nomes são muitas vezes grafados de várias maneiras diferentes, porém semelhantes. Os motivos para isto são vários, desde mudanças na gramática ao longo dos anos, diferença na grafia de certos nomes em línguas diferentes e até por erros ortográficos cometidos ao longo da história. Portanto, é importante que ao realizar uma pesquisa em uma base de dados genealógica, exista a opção de realizar uma pesquisa por palavras similares, para que resultados relevantes não sejam ignorados por não serem idênticos à palavra pesquisada.

Banco de dados relacionais não oferecem naturalmente o suporte para que uma busca por similaridade seja feita, por isso, este trabalho se propõe a apresentar uma implementação de uma busca por similaridade no software de genealogia TNG, com uma técnica eficiente, evitando assim a perda de resultados relevantes em uma pesquisa histórica.

Palavras-Chave: genealogia, busca por similaridade.

Similarity search in a personal database

ABSTRACT

In genealogy, names are often spelled in different ways, although similar. There are a plenty of reasons for this, from grammar changes over the years, difference between names spellings in different languages and even spelling errors committed throughout history. For this reason, it is important to have the option of making a similarity search when a research is made over a genealogical database, so relevant results will not be ignored because they are not spelled identical to the searched word.

Relational database do not provide naturally support for a similarity search, so this paper aims to present an implementation of a similarity search in TNG genealogy software, with an efficient technique, thus avoiding the loss of relevant results in a historic research.

Keywords: genealogy, similarity search.

1 INTRODUÇÃO

Um dos maiores problemas encontrados por genealogistas, historiadores ou por pessoas que buscam pelo passado de suas famílias é que muitas vezes os nomes de pessoas, de famílias ou de lugares são escritas de formas diferentes. Muitos fatores podem levar a isto, desde uma mudança ocorrida na gramática ao longo da história, mudanças de nomes ocorridas pelo fato de haver uma migração de um país para outro, bem como simples erros tipográficos que acabam sendo propagados ao longo da história. Porém, uma característica destas mudanças é que na maioria das vezes, a mudança no nome se dá por poucas letras, fazendo com que a palavra antiga e a nova sejam parecidas. Por exemplo, ao chegar da Itália ao Brasil, alguns dos membros da família "Veronese" tiveram seu sobrenome registrado como "Veronez", assim como ao chegar da Alemanha, alguns "Völz" tiveram seu sobrenome registrado como "Voelz".

Note que caso uma busca seja realizada pela palavra "Veronez", utilizando uma consulta convencional a um banco de dados, resultados com a palavra "Veronese" seriam omitidos, pois uma busca em banco de dados é realizada com uma comparação exata entre strings. Isso levaria a uma perda importante de informações, como por exemplo, os ascendentes da família "Veronez" que residiam na Itália, pois lá a grafia do sobrenome era "Veronese".

Por isso, torna-se importante que esteja disponível para os pesquisadores uma ferramenta para que uma busca por similaridade seja realizada. Porém, isto não é possível ao realizar uma busca padrão em um banco de dados convencional, pois eles oferecem normalmente apenas busca por strings exata ou por trechos de uma string e isto não é suficiente. Uma alternativa, seria a implementação de UDFs (User Defined Functions), que são funções que o usuário pode definir no banco de dados. Infelizmente, estas funções apresentam um desempenho bastante ineficiente. Outra abordagem seria utilizar ferramentas externas ao banco de dados para pré-processar informações, mas esta abordagem normalmente não é desejável.

Para facilitar que genealogistas mantenham o seu próprio website, Darrin Lithgoe desenvolveu o software TNG (The Next Generation of Genealogy Sitebuilding). O software se propõe a ser uma ferramenta para exibir e gerenciar informações genealógicas, sem que os usuários precisem criar páginas estáticas em HTML, e sim armazenando seus dados em MySQL e exibindo páginas dinâmicas criadas com PHP. Apesar de conter muitas opções de busca, o software não conta com uma opção de realizar uma busca por similaridade.

O presente trabalho se propõe a apresentar uma solução para este problema, desenvolvendo um Mod para o software TNG, utilizando a técnica proposta por Gravano (2005) para implementar uma busca por similaridade. Mods são arquivos de

configurações que permitem que o usuário personalize a sua versão do TNG e com este Mod, será possível a usuários do software que extendam o seu produto para que possam realizar uma busca por similaridade em suas bases de dados pessoais.

Este documento está organizado da seguinte forma. O Capítulo 2 se propõe a apresentar o software TNG, focando-se nos TNG Mods, que são extensões do software que podem ser feitas pelos usuários. O Capítulo 3 faz uma explicação sobre a técnica utilizada para a busca por similaridade. O Capítulo 4 mostra quais são as funcionalidades do sistema, descrevendo os casos de uso disponíveis ao usuário. Finalmente, o Capítulo 5 dá uma visão geral sobre a arquitetura do sistema, mostrando como foi desenvolvido o mod, bem como foi a implementação da técnica de similaridade.

2 TNG: THE NEXT GENERATION OF GENEALOGY SITEBUILDING

O TNG (The Next Generation of Genealogy Sitebuilding) é uma ferramenta para o gerenciamento e exibição de dados genealógicos em seu próprio website (LYTHGOE, 2013). Desenvolvido por Darrin Lithgoe, ela utiliza o DBMS (Database Management System) MySQL e a linguagem de script PHP (PHP: Hypertext Preprocessor) para que o usuário obtenha facilmente um website para compartilhar e administrar suas informações genealógicas.

2.1 TNG Mods

Para que o usuário possa personalizar a sua versão do TNG, são oferecidas algumas opções. Uma delas são os TNG Mods, que são arquivos de configuração que permitem pequenas modificações nos códigos originais para que sejam implementadas novas funcionalidades à aplicação.

Com a criação deste arquivo de configuração, é possível compartilhar as modificações com outros usuários e essas alterações podem ser instaladas facilmente na aplicação a partir de um Gerenciador de Mods existente, assim como removidas caso o usuário não deseje mais utilizá-la. Para tanto, basta que os arquivos do Mod sejam colocados na pasta *mods* presente na raíz do sistema, para que o Mod apareça como uma opção a ser instalado no Gerenciador de Mods.

Para criar um Mod, o desenvolvedor tem basicamente 3 opções, que são: modificar um arquivo existente, criar um novo arquivo e copiar um arquivo. As opções serão detalhadas a seguir.

2.1.1 Modificando arquivos

Para modificar um arquivo, o primeiro passo a ser realizado é idenfiticar qual o arquivo que será modificado, isto é feito utilizando o parâmetro *%target:file_to_change.php%*, onde *file_to_change.php* é o arquivo que se deseja modificar. Opcionalmente, após este parâmetro, pode-se colocar alguma observação, como por exemplo, a linha onde vai ocorrer a alteração. Em seguida, deve ser colocado entre os parâmetros *%location:%* e *%end:%* o local do código em que devem ser feitas alterações. Após indicado o local, existem 3 opções para mudar o código. A tag *%insert:after%* insere após o local indicado pela tag *%location:%* o cógido inserido entre ela e a tag *%end:%*. A tag *%insert:before%* insere antes do local indicado pela tag *%location:%* o código inserido entre ela e a tag *%end:%*. Finalmente, a tag *%replace:%* substitui o código indicado pela tag *%location:%* pelo código inserido entre ela e a tag *%end:%*.

Além das opções acima, ainda existem as tags *%triminsert:after%*, *%triminsert:before%* e *%trimreplace:%* que tem comportamento semelhante com suas correspondentes acima, porém os códigos serão inseridos sem quebra de linha.

É possível inserir várias sequências de parâmetros *%location: - %end:% %insert:after% - %end:%* (ou uma das outras opções) para um mesmo *%target:%* ou definir repetir a tag *%target:%*, não interferindo no funcionamento do mod.

2.1.2 Criando novos arquivos

Criar novos arquivos é possível com a tag *%newfile:new_file.php%* onde *new_file.php* é o nome do arquivo que será criado. O arquivo será criado no diretório raíz do TNG, podendo ser criado também dentro de diretórios, por exemplo, *%newfile:diretorio/new_file.php%.*

Em seguida, deve ser inserida a tag *%fileversion:%*, que deverá indicar, após os dois pontos, para qual versão do TNG o Mod se destina. O conteúdo deste novo arquivo, será todo o código que estiver entre a tag *%fileversion:%* e a tag *%fileend:%*. Ainda deve ser colocado, próximo ao topo do conteúdo do código que está sendo inserido, um comentário com a tag *%version:%*, que indicará também a versão do arquivo, e se não for igual à versão indicada em *%fileversion:%* resultará em um erro quando for instalado o Mod no Gerenciador de Mods.

Outra observação importante, é que não é possível criar um mod apenas com criação de novos arquivos, sendo necessário que haja pelo menos uma modificação de arquivo para o Mod ser válido.

2.1.3 Copiando arquivos

A última opção para o desenvolvedor de um Mod é realizar uma cópia de um arquivo, que deverá estar junto com o arquivo de configuração na pasta *mods*, para os arquivos de sistema. Para isto existem duas tags. A primeira *%copyfile:exemplo.jpg%* copia o arquivo *exemplo.jpg* presente na pasta *mods* para o diretório raíz do TNG. A segunda tag *%copyfile2:diretorio_exemplo/exemplo.jpg%* copia o arquivo *exemplo.jpg* da pasta *mods* para o diretório *diretorio_exemplo.*

Assim como na criação de arquivos, não é possível criar um Mod somente com copias de arquivos, sendo necessário que haja pelo menos uma modificação de arquivo para que o Mod seja válido.

2.1.4 Opções adicionais

As tags *%name:%, %version:%* e *%description:%* são inseridas no início do arquivo de configuração e indicam o nome do Mod, qual a sua versão e uma breve descrição da funcionalidade do Mod.

A tag *%fileoptional:%* deve ser usada após a tag *%target:%* para indicar que caso não exista o arquivo, não deve ser gerado nenhum erro no Gerente de Mods pois este arquivo é opcional.

A tag *%parameter:%* faz com que o Gerente de Mods exiba, após a instalação do Mod uma opção *Editar*, para que o usuário entre com algum parâmetro que modifique um trecho do código do Mod. A tag *%desc:%* é usada para descrever a ação deste parâmetro e as instrução de uso no Gerenciador de Mods.

2.1.5 Exemplo

O exemplo a seguir, ilustra a implementação de um Mod simples, que adiciona um link a um menu, cria a página para onde o link aponta, que simplesmente exibe uma imagem que será copiada para a pasta *extensions*.

```
01. %name:Mod para exemplo%
02. %version:v9.2.0.4%
03. %description:Esta é uma descrição para o Mod de exemplo%
04.
05. %target:genlib.php%
06.    Location 1 – line 390
07. %location%
08.    $menu .= tngddrow(getURL( "famsearchform", 0 ), "fsearch-icon", "",
       "searchfams");
09. %end:%
10. %insert:after%
11.    $menu .= "<li><a href="extensions/imagem.html">Imagem</a></li>";
12. %end:%
13.
14. %newfile:extensions/imagem.html%
15. %fileversion:v9.2.0.4%
16. <html>
17.    <!-- %version:v9.2.0.4% -->
19.    <head></head>
19.    <body>
20.          <img src="foto.jpg" />
21.    </body>
22. </html>
23. %fileend:%
24.
25. %copyfile2:extensions/foto.jpg%
```

As linhas 1 até 3 definem o nome do Mod, sua versão e sua descrição. O código entre as linhas 5 e 12, modificam o arquivo *genlib.php* que é onde é definido o menu de busca da aplicação. A linha 8 indica qual o local onde o novo link será inserido no menu. No exemplo, este local será abaixo da opção *Famílias*. A linha 10 indica que o código será colocado após o código da opção de menu *Famílias*. Finalmente a linha 11 é o código necessário para que uma nova entrada do menu, com um link para a página *extensions/imagem.html* seja criada.

Entre as linhas 14 e 23 é criado o arquivo *imagem.html* que ficará na pasta *extensions*. Note que conforme especificado em 2.1.2, foi definida que a versão do arquivo será *v9.2.0.4*. Isso está indicado na linha 14 e no comentário que é exigido, que está na linha 17. As linhas 16 a 22 são a implementação de uma página simples de HTML que apenas mostra a imagem *foto.jpg*.

Por último, na linha 25, a imagem *foto.jpg*, que deve estar na pasta *mods* junto com o arquivo de configuração, é copiada para a pasta *extensions* para que o arquivo criado anteriormente possa exibi-la.

3 USANDO Q-GRAMS PARA REALIZAR UMA BUSCA POR SIMILARIDADE

Como mencionado anteriormente, realizar uma busca por similaridade em um banco de dados relacional não é uma tarefa trivial pelo fato de os banco de dados não suportarem tal mecanismo naturalmente. Tampouco é desejável utilizar ferramentas externas ao banco de dados ou ainda usar UDFs (User Defineds Functions) nos banco de dados, pois estas apresentam um desempenho ineficiente. Como alternativa a tais abordagens, será adotada na aplicação a solução proposta em (GRAVANO, 2005), que visa realizar uma busca por similaridade mais eficiente, que será explicada ao longo deste capítulo.

3.1 Notações

Será usado *R*, possívelmente com subsctritos, para denotar uma tabela, *A*, possívelmente com substritos, para denotar as colunas de uma tabela, e *t*, possívelmente com subscritos, para denotar os registros de uma tabela.

A notação $R.A_i$ representa a coluna A_i da tabela R e a notação $R.A_i(t_j)$ representa o valor do registro t_j da coluna A_i. Será usado $\sum$ como um alfabeto finito de tamanho $|\sum|$. Serão usados letras gregas minúsculas, como σ, possívelmente com subscritos, para denotar strings em $\sum^*$. Será usado $\sigma \in \sum^*$ para denotar um string de comprimento *n*. Por último, será usado $\sigma[i \ldots j], 1 \leq i \leq j \leq n$, para denotar um substring de σ de comprimento $j - i + 1$ que começa na posição i.

3.2 *Q*-grams

Dada uma string σ, obtemos seus *q*-grams posicionais "deslizando" uma janela de comprimento *q* sobre a string σ. Como os *q*-grams no início e do fim de σ podem ter menor do que *q* caracteres, introduzimos os caracteres "#" e "%", não pertencentes a $\sum$, que extendem σ com $q - 1$ de "#" no início e $q - 1$ ocorrências de "%" no final. Desta maneira, todas os *q*-grams contém exatamento *q* caracteres, embora alguns não pertençam a $\sum$.

Sendo assim, um *q*-gram posicional de uma string σ é um par $(i,\sigma[i...i + q - 1])$, onde $\sigma[i...i + q - 1]$ é o *q*-gram de σ, já extendido, que começa na posição i. O conjunto G_σ de todos os *q*-grams posicionais de uma string σ é o conjunto de todos os $|\sigma| + q - 1$ pares contruídos com todos os *q*-grams de σ.

A ideia por trás do uso de *q*-grams para uma busca de strings por aproximação vem do fato que duas strings que sejam similares uma com a outra, terão em comum um grande número de *q*-grams (UKKONEN, 1992). Considere o exemplo a seguir.

Os *q*-grams posicionais de tamanho *q=3* para a string "joao_silva" são: {(1,##j), (2,#jo), (3,joa), (4,oao), (5,ao_), (6,o_s), (7,_si), (8,sil), (9,ilv), (10,lva), (11,va%), (12,a%%)}. Já a string "joao_da_silva" tem os seguintes *q*-grams posicionais de tamanho *q=3*: {(1,##j), (2,#jo), (3,joa), (4,oao), (5,ao_), (6,o_d), (7,_da), (8, da_), (9, a_s), (10, _si), (11,sil), (12,ilv), (13,lva), (14,va%), (15, a%%)}. Ignorando a informação sobre posição, os dois conjuntos compartilham 10 *q*-grams. Considerando a posição, os conjuntos compartilham apenas os 5 primeiros *q*-grams, porém outros 6 *q*-grams se diferenciam por apenas 3 posições. Isso mostra que, em geral, o uso de *q*-grams posicionais para uma busca de string por similaridade envolverá a comparação entre *q*-grams dentro de uma "faixa" de posições.

3.3 Aproximando strings em um banco de dados

O problema a ser resolvido pode ser formalizado com o seguinte: dada uma tabela *R*, com uma coluna do tipo string $R.A_i$ e uma consulta σ, deve-se retornar todos os resultados $t \in R$ tais que $R.A_i(t)$ sejam similares a σ.

3.3.1 Preparando o banco de dados

Para que seja possível uma busca por aproximação em um banco de dados, é necessário que haja um mecanismo para popular a base de dados com os *q*-grams posicionais dos dados originais do banco de dados.

Para isto, considere que *R* seja uma tabela com um esquema (A_0, A_1, ..., A_m), onde A_0 é a chave e algumas das colunas A_i, i > 0, são do tipo string. Para cada coluna A_i que queremos considerar na busca por aproximação, criamos uma tabela auxiliar $RA_iQ(A_0, Pos, Qgram)$ com três colunas. Para um string σ de um registro de uma coluna A_i de *R*, seus $|\sigma| + q - 1$ *q*-grams são representados por $|\sigma| + q - 1$ registros separados na tabela RA_iQ, onde $RA_iQ.Pos$ identifica a posição do *q*-gram $RA_iQ.Qgram$. Para todos estes registros, o valor na coluna $RA_iQ.A_0$ será o mesmo, e servirá de chave estrangeira com a tabela *R*.

Estas tabelas podem ser criadas facilmente em sistemas de banco de dados tradicionais usando declarações simples de SQL. Para isto, é usada uma tabela *N*, que contém apenas uma coluna *I*, com registros com os valores de 1 até *M*, onde *M* é o valor máximo do comprimento de uma string. Então, é feita uma junção desta tabela com a coluna $R.A_i$, e são pegos todos os *q*-grams de cada string em $R.A_i$ que começam na posição *x*, onde *x* é o valor armazenado no campo *I* da tupla de *N* com $R.A_i$. A query usada para tanto é mostrada na Figura 3.1

```
INSERT  INTO RA_iQ
        SELECT R.A_0, N.I,
           SUBSTR(SUBSTR('#...#',1,q-1) || UPPER(R.A_i) || SUBSTR('%...%',1,q-1), N.I, q)
        FROM R, N
        WHERE N.I ≤ LENGTH(R.A_i)+q-1;
```

Figura 3.1: Criando a tabela auxiliar RA_iQ (GRAVANO, 2005)

3.3.2 Filtrando resultados com as propriedades dos *q*-grams

Para obter corretude e eficiência, é preciso que não aconteçam falsas rejeições e que aconteçam poucos falsos positivos. Para isto, a técnica propõe usar três propriedades dos *q*-grams, cada uma usada como um filtro na consulta ao banco de dados. Esses filtros serão explicados a seguir.

Antes de continuar, será definido como *k*, tal que $k \geq 0$, um parâmetro que indica o quão uma string $t \in R$, tal que $R.A_i(t)$, deve ser parecida com uma consulta σ para que *t* seja um resultado válido. Quanto menor o valor de *k*, mais similar uma string deve ser da outra, sendo que para o valor 0, as strings devem ser iguais.

3.3.2.1 Filtro por contagem

O filtro tem por base as informações contidas nos conjuntos $G_{\sigma 1}$ e $G_{\sigma 2}$, dos *q*-grams das strings σ_1 e σ_2. Intuitivamente, ignorando a informação posicional, se duas strings forem similares, então elas tem um grande número de *q*-grams em comum. Isso pode ser formalizado como:

Proposição 3.1: Condidere as strings σ_1 e σ_2, de tamanho $|\sigma_1|$ e $|\sigma_2|$ respectivamente. Se σ_1 e σ_2 são similares entre si, então a cardinalidade de $G_{\sigma 1} \cap G_{\sigma 2}$, ignorando-se a informação posicional, deve ser de pelo menos $(max(|\sigma_1|, |\sigma_2|) + q - 1) - k * q$.

3.3.2.2 Filtro por posição

Para aproveitar-se também da informação posicional de cada *q*-grams, é proposto o filtro por posição. Um certo *q*-gram de uma string pode não ocorrer em outra string, assim como as posições de *q*-grams iguais podem estar fora de uma faixa válida, devido a inserções ou remoções em uma das strings. Existe ainda a possibilidade de um *q*-gram de uma string ocorrer várias vezes em outra string.

Por isso, é dito que um *q*-gram posicional (i, τ_1) em uma string σ_1 *corresponde* a um *q*-gram posicional (j, τ_2) em uma string σ_2 se $\tau_1 = \tau_2$, após uma sequência de operações de efição que converte σ_1 em σ_2 e afeta apenas a posição do *q*-gram τ_1, se torna o *q*-gram (j, τ_2) na string editada (ULLMANN, 1997). Com isso, apesar da complexidade de corresponder *q*-grams posicionais, um filtro útil pode ser feito a partir da seguinte observação:

Proposição 3.2: Se duas strings σ_1 e são similares entre si, então um *q*-gram posicional de uma *não pode corresponder* a um *q*-gram posicional de outro que difira mais do que *k* posições.

3.3.2.3 Filtro por comprimeto

Por último, é observado que pode-se filtrar rapidamente resultados a partir do comprimento de uma string, como segue:

Proposição 3.3: Se duas strings σ_1 e σ_2 são similares entre si, então seus comprimentos não podem se diferenciar por mais de *k.*

3.3.3 Expressando *Q*-grams em SQL

Tendo preparado a tabela auxiliar descrita na seção 3.3.1, os filtros descritos acima podem ser facilmente usados em uma expressão SQL que pode ser usada em um banco de dados relacional. A Figura 3.2 mostra a implementação de uma consulta SQL que usa os três filtros para realizar uma busca por similaridade

```
SELECT    R.A_0, R.A_i
FROM      R, TQ, RA_iQ
WHERE     R.A_0 = RA_iQ.A_0 AND RA_iQ.Qgram = TQ.Qgram AND
          RA_iQ.Pos ≤ TQ.Pos + k AND RA_iQ.Pos ≥ TQ.Pos − k AND
          LENGTH(R.A_i) ≤ LENGTH(σ) + k AND LENGTH(R.A_i) ≥ LENGTH(σ) − k
GROUP BY  R.A_0, R.A_i
HAVING    COUNT(*) ≥ LENGTH(R.A_i) − 1 − (k − 1) * q AND COUNT(*) ≥ LENGTH(σ) − 1 − (k − 1) * q
```

Figura 3.2: Consulta SQL para busca por similaridade (GRAVANO, 2005)

O obejevo desta consulta é o seguinte: dada uma string σ, obter todos os registros de *R* tal que a similaridade entre σ e o registro seja menor do que *k*. Para isto, primeiramente é criada uma tabela auxiliar *TQ*, contendo as mesmas colunas de *RA*$_i$*Q*, onde serão inseridos todos os *q*-grams posicionais de σ. Isso é feito facilmente de maneira similar à descrita na Figura 3.2.

A consulta em sí basicamente realiza uma junção entre as tabelas *R* e *RA*$_i$*Q*, em suas colunas A_0, e uma junção entre as tabelas *RA*$_i$*Q* e *TQ* em suas colunas *Qgrams*. Com a junção entre *RA*$_i$*Q* e *TQ* é feita a comparação entre os *q*-grams e com a junção entre *R* e *RA*$_i$*Q* são retornados ao usuário os resultados.

O filtro por contagem é implementado na cláusula HAVING, pelas condições $COUNT(*) \geq LENGTH(R.A_i) - 1 - (k-1) * q$ e $COUNT(*) \geq LENGTH(\sigma) - 1 - (k-1) * q$. Com isto, resultados que tenham poucos *q*-grams em comuns são descartados. O número mínimo de *q*-grams necessário é obtido a partir do tamanho da palavra LENGTH(), a precisão desejada *k* e o tamanho do *q*-gram *q*.

O filtro por posição é implementado pelas condições $RA_iQ.Pos \leq TQ.Pos + k$ e $RA_iQ.Pos \geq TQ.Pos - k$ na cláusula WHERE. Com isto, resultados que tenham *q*-grams em comum, mas sua posição seja maior do que uma precisão desejada *k* são descartados.

Finalmente, o filtro por comprimento é implementado na cláusula WHERE pelas condições $LENGTH(R.A_i) \leq LENGTH(\sigma) + k$ e $LENGTH(R.A_i) \geq LENGTH(\sigma) - k$. Desta maneira, eliminamos resultados que tenham um comprimento que diferem da consulta σ por mais do que *k*.

4 FUNCIONALIDADES DO SISTEMA

Neste capítulo, será mostrado quais são as funcionalidades do sistema do ponto de vista do usuário. Para isto, será utilizada a linguagem UML *(Unified Modeling Language).* A UML é uma linguagem de modelagem usada para especificar, visualizar, construir e documentar os artefatos de um sistema (ALHIR, 1999). Controlada e padronizada pelo OMG *(Object Management Group)*, é bastante utilizada na área de Engenharia de Software.

4.1 Diagramas de Caso de Uso

Para demonstrar como os usuários interagem com o sistema, será apresentado o diagrama de casos de uso em UML. Através do diagrama, é possível identificar rapidamente quais são as funcionalidades previstas pelo sistema, bem como quais usuários tem acesso a cada uma delas. O diagrama de casos de uso do sistema é apresentado na figura 4.1 a seguir.

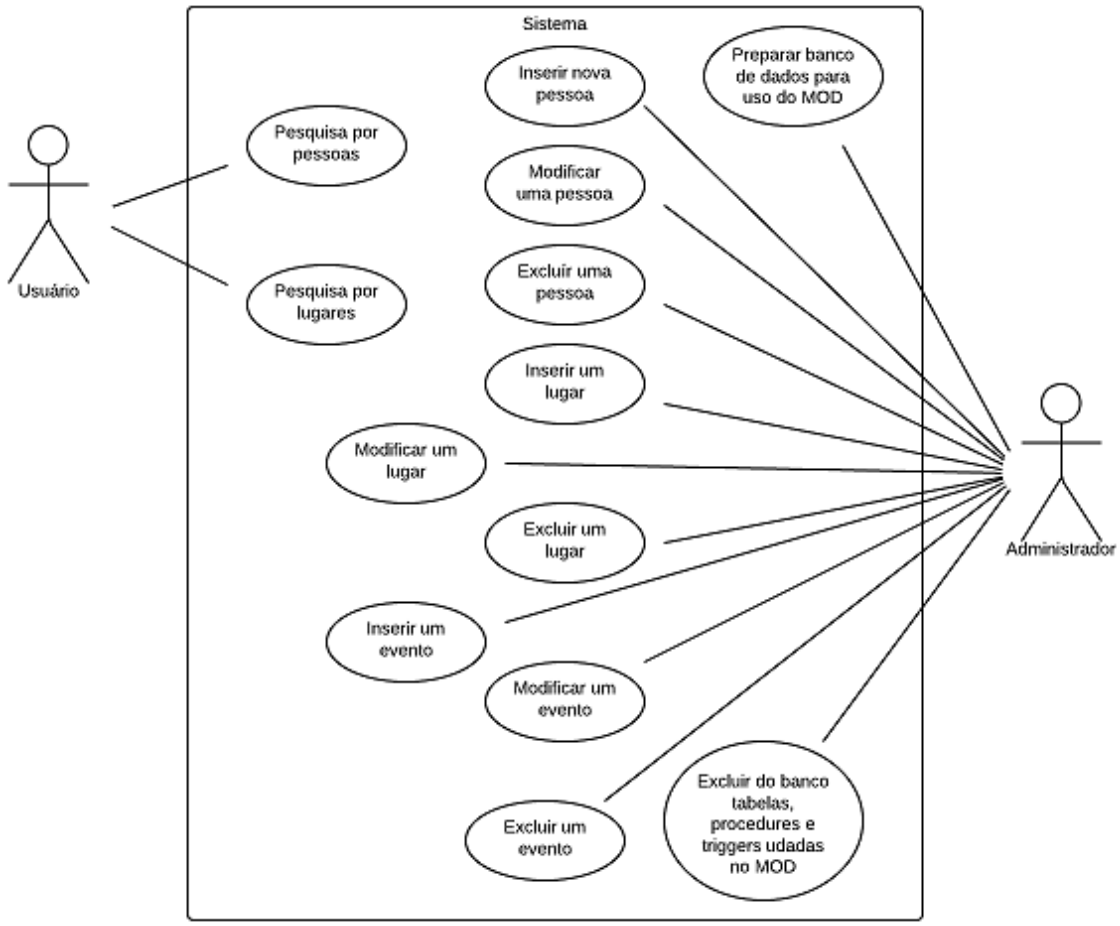

Figura 4.1: Diagrama de Casos de Uso

Podemos observar na Figura 4.1 os três elementos que compõe o diagrama de caso de uso: os atores, os casos de usos e as associações.

Os atores indicam papéis executados por agentes de fora do sistema. Pode ser um usuário ou um outro sistema que utiliza o sistema. Nesta aplicação, existem dois tipos de atores, o Usuário, que é quem faz as pesquisar na aplicação, e o Administrador, que é o responsável por preparar a aplicação para o uso do Usuário, além de ser quem pode modificar os dados presentes no banco de dados. São representados por um boneco.

Os casos de uso representam as ações que são executadas pelo sistema, das quais podemos esperar observar um resultado. O caso de uso não descreve como a funcionalidade é implementada, ele apenas deve descrever o que esperar como resposta do sistema quando o Ator realizar alguma ação. São representados no diagrama por uma elipse com o nome do caso de uso dentro.

O terceiro elemento presente são as associações, que indicam interações entre os elementos. Nesta aplicação temos dois tipos associações, sendo uma a associação entre um ator e um caso de uso, que indica qual ator interage com qual parte do sistema. São representadas no diagrama por uma linha entre elementos. Outra associação se dá entre os casos de uso, sendo usada na aplicação apenas a do tipo precedência, indicando que um caso de uso precede o outro. Na aplicação, o caso de uso “Preparar o banco de dados

para uso do Mod" precede todos os outros, pois sem ele, nenhum outro caso de uso funcionará corretamente.

Os casos de uso "Pesquisa por pessoas" e "Pesquisa por lugares" exibem seus resultados na tela, diferenciando-se um do outro pela origem das informações que serão exibidas, além de como o resultado será exibido ao usuário.

Os casos de uso "Inserir nova pessoa", "Modificar uma pessoa", "Excluir uma pessoa", "Inserir novo lugar", "Modificar um lugar", "Excluir um lugar", "Inserir novo evento", "Modificar um evento" e "Excluir um evento" tem todos a finalidade de manter os dados da tabela de grams consistentes com a das tabelas do TNG sempre que o administrador fizer alguma modificação na tabela de Pessoas, Eventos ou Lugares.

Por último, os casos de uso "Preparar o banco de dados para uso do Mod" e "Excluir do BD tabelas, procedures e triggers usados no Mod" são usadas pelo administrador para começar a usar o Mod, e para usar e deixar o sistema como era originalmente caso não deseje mais usá-lo.

4.2 Descrições dos casos de uso e da interface do sistema

Serão apresentadas a seguir as descrições de cada caso de uso, bem como telas demonstrando como deve ser o resultado de cada caso de uso.

Pré-condição é um estado que o sistema deve estar para que seja possível realizar o caso de uso. Na aplicação, apenas o caso de uso "Preparar o banco de dados para uso do Mod" não tem pré-condição. Para todos os outros, a pré-condição é a realização do caso de uso "Preparar o banco de dados para uso do Mod", conforme é explicado na seção 4.1.

Tabela 4.1: Descrição do caso "Preparar o banco de dados para o uso do Mod"

UC 01 – Preparar o banco de dados para uso do Mod	
Atores	Administrador
Pré-condição:	Nenhuma
Pós-condições:	Banco de dados preparado para uso da aplicação.
Descrição:	Deve ser criado no banco de dados todas as tabelas, procedures e triggers usadas na aplicação, além de já preparar os dados para todos os nomes, eventos e lugares que já existem no banco.
Sequência de eventos:	
Ator	**Sistema**
1. Seleciona os tipos de eventos que serão usados na aplicação e clica em "Preparar"	
	2. Cria no banco de dados as tabelas da aplicação, as procedures que serão usadas,

	as triggers e então executa as procedures que preparam os dados para os nomes, eventos e lugares já existentes no banco de dados. Se não ocorrer nenhum problema, deverá ser exibida uma mensagem de sucesso. O caso de uso é encerrado.
Fluxo alternativo 01:	
1.1 Seleciona os tipos de eventos que serão usados na aplicação e clica em "Preparar"	
	1.2 Cria no banco de dados as tabelas da aplicação, as procedures que serão usadas, as triggers e então executa as procedures que preparam os dados para os nomes, eventos e lugares já existentes no banco de dados. Ocorrendo um problema, deverá ser informado ao usuário que ocorreu um erro.

A tabela 4.1 apresenta o detalhamento da preparação para uso do Mod. Esta opção se encontra na seção administrativa do site, no menu "Setup", na aba "Criação de Tabelas".

Para este caso de uso, é apresentado também um fluxo alternativo, que é a sequência de eventos que deve acontecer caso ocorra uma falha no fluxo principal. Para este caso, caso ocorra algum problema na preparação do banco de dados, deve ser exibida uma mensagem ao usuário informando o erro.

Para os casos de uso que tem como ator Usuário, os casos de uso começam quando o usuário clica na opção "Por Palavra" no menu "Busca" da Aplicação. A Figura 4.2 apresenta como deve ser a tela inicial quando o usuário clica na opção "Por Palavra"

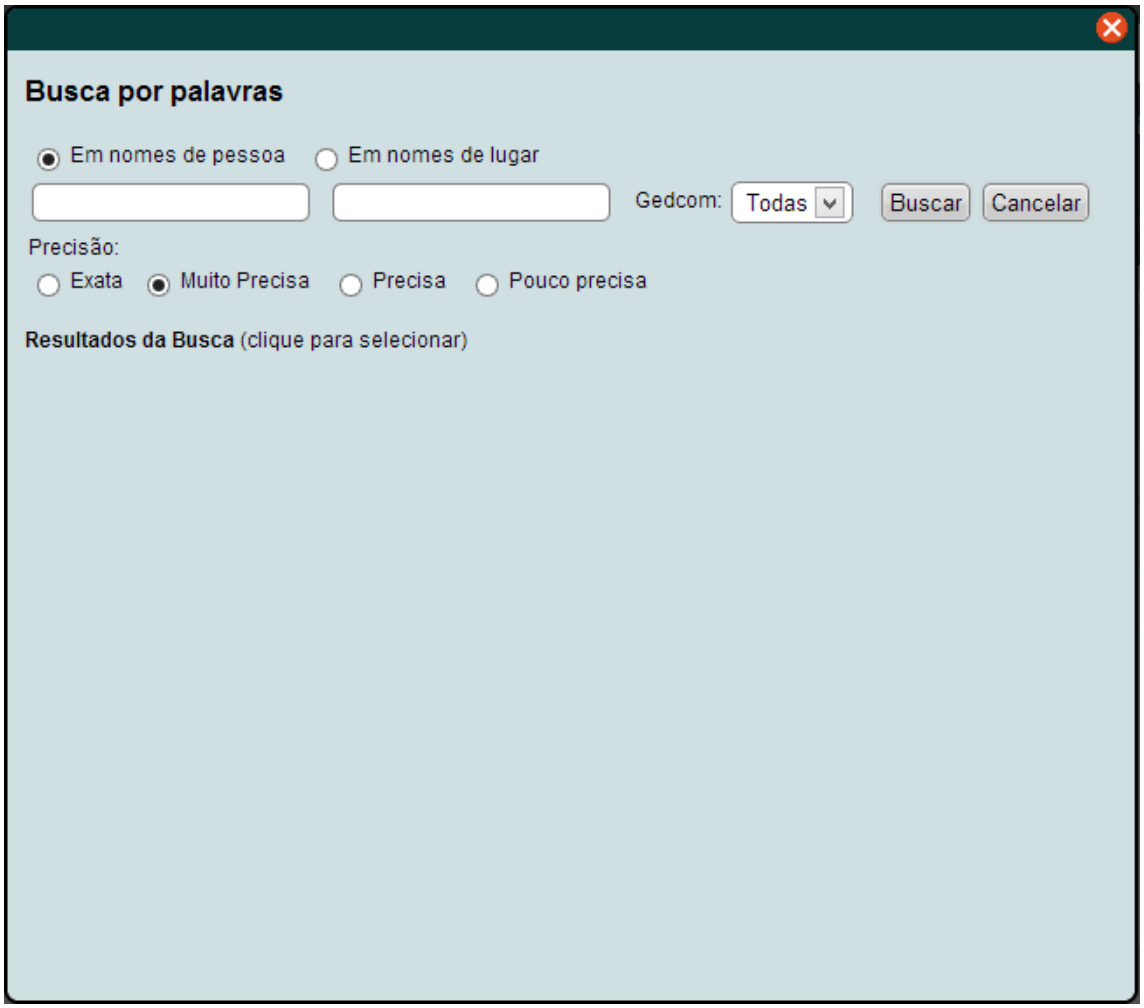

Figura 4.2: Interface para a pesquisa

Na Figura 4.2 é possível observar que por padrão já vem selecionado para o Usuário realizar o caso de uso “Pesquisa por pessoas”, caracterizada pelo Radio Button marcado ao lado de “Em nomes de pessoas”. Selecionando o Radio Button ao lado de “Em nomes de lugares”, a consulta realizada será a descrita no caso de uso “Pesquisa por lugares”. É possível realizar a busca com uma ou duas palavras, por isso existem dois campos de texto para realizar a pesquisa.

Tabela 4.2: Descrição do caso de uso “Pesquisa por pessoas”

UC 02 – Pesquisa por pessoas	
Atores:	Usuário
Pré-condição:	Realização do caso de uso UC 01 - "Preparar o banco de dados para uso do Mod"
Pós-condição	Tabela listando os nomes resultantes da pesquisa com o critério informado pelo usuário
Descrição:	Usuário digita as palavras que deseja

	pesquisar e enquanto digita são listados os nomes resultantes da pesquisa realizada com o que foi digitado até o momento
Sequência de eventos:	
Ator	**Sistema**
1. Começa a digitar as palavras que deseja pesquisar. Usuário também pode selecionar pesquisar em uma Gedcom específica, e também definir a precisão da consulta.	
	2. Apresenta uma tabela listando todos os resultados da pesquisa realizada até o momento. As colunas apresentadas na tabela deverão ser: ID da pessoa, Nome, Data de Nascimento, Data de Falecimento.
3. Usuário clica no nome desejado na lista resultante.	
	4. Sistema redireciona para a página da pessoa selecionada pelo usuário. O caso de uso é encerrado.
RN 01 – Regra de negócio 01	
1. A pesquisa por pessoas deve se realizar cada vez que o usuário inserir uma nova letra em um dos campos de pesquisa	
RN 02 – Regra de negócio 02	
2. Sempre que o usuário mudar uma das opções (Gedcom ou precisão), deverá ser realizada uma nova pesquisa	

Na Tabela 4.2 é detalhado o caso de uso “Pesquisa por pessoas” que ocorre quando o usuário faz uma pesquisa com a opção “Em nomes de pessoas” da interface selecionada.

O caso de uso apresenta também duas regras de negócio, que são regras que não se encaixam na sequência de eventos do fluxo principal, porém devem ser obrigatoriamente seguidas.

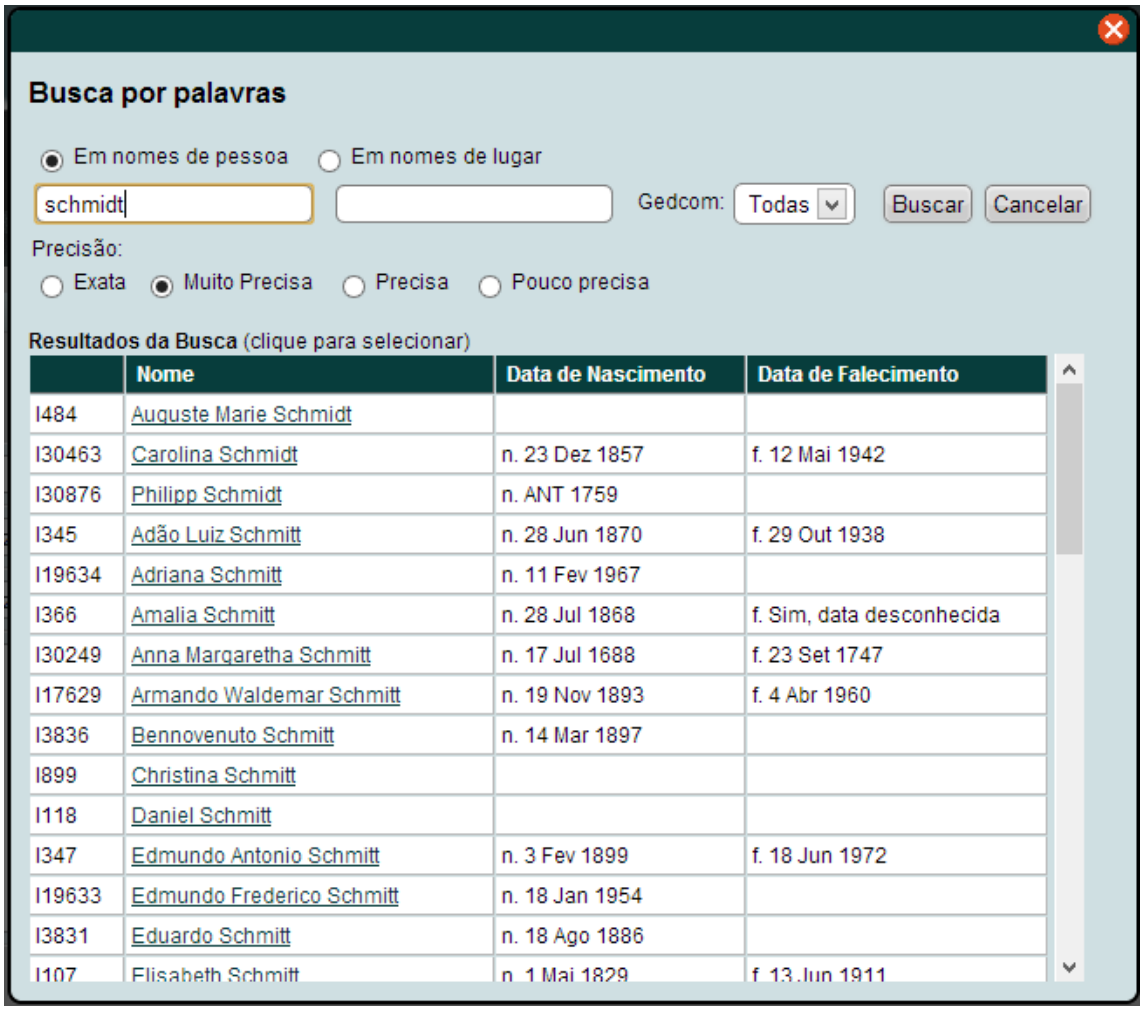

Figura 4.3: Interface do caso de uso “Pesquisa por pessoas”

Na Figura 4.3 é mostrado a pós-condição do caso de uso “Pesquisa por pessoas”, a tabela exibindo os resultados da pesquisa. Conforme descrito anteriormente, temos uma tabela com as colunas ID da pessoa, Nome, Data de Nascimento e Data de Falecimento. O nome da pessoa é um link que leva para a página da pessoa.

Tabela 4.3: Descrição do caso de uso “Pesquisa por lugares”

UC 03 – Pesquisa por lugares	
Atores:	Usuário
Pré-condição:	Realização do caso de uso UC 01 - "Preparar o banco de dados para uso do Mod"
Pós-condição	Lista os lugares resultantes da pesquisa com o critério informado pelo usuário
Descrição:	Usuário digita as palavras que deseja pesquisar e enquanto digita são listados os lugares resultantes da pesquisa realizada com o que foi digitado até o momento

Sequência de eventos:	
Ator	**Sistema**
1. Começa a digitar o lugar que deseja pesquisar. Usuário também pode selecionar pesquisar em uma Gedcom específica, e também definir a precisão da consulta.	
	2. Apresenta uma lista com todos os resultados da pesquisa realizada até o momento.
3. Usuário clica no lugar desejado na lista resultante.	
	4. Sistema redireciona para a página do lugar selecionada pelo usuário. O caso de uso é encerrado.
RN 01 – Regra de negócio 01	
1. A pesquisa por lugares deve se realizar cada vez que o usuário inserir uma nova letra em um dos campos de pesquisa	
RN 02 – Regra de negócio 02	
2. Sempre que o usuário mudar uma das opções (Gedcom ou precisão), deverá ser realizada uma nova pesquisa	

Na Tabela 4.3 é detalhado o caso de uso “Pesquisa por lugares” que ocorre quando o usuário realiza uma pesquisa com a opção “Em nomes de lugares” da interface selecionada.

O caso de uso apresenta as mesmas regras de negócio que devem ser obedecidas na “UC 02 – Pesquisa por pessoas”.

Figura 4.4: Interface do caso de uso “Pesquisa por lugares”

Na Figura 4.4 é mostrado a pós-condição do caso de uso “Pesquisa por lugares”, a lista exibindo os resultados da pesquisa. Conforme descrito anteriormente, temos uma lista com os nomes de lugares resultantes da pesquisa, e caso o lugar tenha uma página, o nome é um link que redireciona para ela.

Estes são os dois casos de uso feitos pelo ator Usuário. A seguir, será detalhado os casos de uso do ator Administrador (além do caso “Preparar o banco de dados para o uso do Mod”, que precede todos). Os casos de uso a seguir apresentam apenas as tabelas de descrição, pois não há nenhuma interface para o usuário, além das já existentes no TNG.

Tabela 4.4: Descrição do caso de uso “Inserir uma pessoa”

UC 04 – Inserir uma pessoa	
Atores:	Administrador.
Pré-condição:	Realização do caso de uso UC 01 - "Preparar o banco de dados para uso do Mod".

Pós-condição:	Nome inserido pelo administrador presente nas tabelas qgrams_names e qgrams_grams.
Descrição:	Ao inserir uma pessoa, é disparada uma trigger no banco de dados que deve preparar o nome digitado para ser usado na busca por similaridade
Sequência de eventos:	
Ator	**Sistema**
1. Insere uma nova pessoa no banco de dados do TNG.	
	2. Insere cada palavra que compõe o nome em uma linha da tabela qgrams_names, gera os grams de cada palavra e os insere em qgrams_names. O caso de uso é encerrado.

Na Tabela 4.4 é detalhado o caso de uso “Inserir nova pessoa” que ocorre sempre que o administrador incluir uma nova pessoa no banco de dados.

Tabela 4.5: Descrição do caso de uso “Modificar uma pessoa”

UC 05 – Inserir uma pessoa	
Atores:	Administrador.
Pré-condição:	Realização do caso de uso UC 01 - "Preparar o banco de dados para uso do Mod".
Pós-condição:	Antigo nome da pessoa retirado das tabelas qgrams_names e qgrams_grams e novo nome inserido pelo administrador presente nas mesmas tabelas.
Descrição:	Ao modificar uma pessoa, é disparada uma trigger no banco de dados que deve excluir as entradas antigas e preparar o nome digitado para ser usado na busca por similaridade
Sequência de eventos:	
Ator	**Sistema**
1. Modifica o nome de uma pessoa no banco de dados do TNG.	
	2. Remove das tabelas qgrams_names e qgrams_grams as linhas com informações da pessoa sendo modificada,

	em seguida insere cada palavra que compõe o novo nome em uma linha da tabela qgrams_names, gera os grams de cada palavra e os insere em qgrams_names. O caso de uso é encerrado.

Na Tabela 4.5 é detalhado o caso de uso “Modificar uma pessoa” que ocorre sempre que o administrador modificar o nome de pessoa no banco de dados.

Tabela 4.6: Descrição do caso de uso “Excluir uma pessoa”

UC 06 – Excluir uma pessoa	
Atores:	Administrador.
Pré-condição:	Realização do caso de uso UC 01 - "Preparar o banco de dados para uso do Mod".
Pós-condição:	Nome da pessoa retirado das tabelas qgrams_names e qgrams_grams.
Descrição:	Ao excluir uma pessoa, é disparada uma trigger no banco de dados que deve excluir as entradas referentes à pessoa excluída das tabelas qgrams_names e qgrams_grams
Sequência de eventos:	
Ator	**Sistema**
1. Exclui uma pessoa do banco de dados do TNG.	
	2. Remove das tabelas qgrams_names e qgrams_grams as linhas com informações da pessoa sendo excluída. O caso de uso é encerrado.

Na Tabela 4.6 é detalhado o caso de uso “Excluir uma pessoa” que ocorre sempre que o administrador excluir uma pessoa do banco de dados.

Tabela 4.7: Descrição do caso de uso “Inserir um evento”

UC 07 – Inserir um evento	
Atores:	Administrador.
Pré-condição:	Realização do caso de uso UC 01 - "Preparar o banco de dados para uso do Mod".

Pós-condição:	Evento inserido pelo administrador presente nas tabelas qgrams_names e qgrams_grams.
Descrição:	Ao inserir um evento, é disparada uma trigger no banco de dados que deve preparar o evento digitado para ser usado na busca por similaridade
Sequência de eventos:	
Ator	**Sistema**
1. Insere um novo evento no banco de dados do TNG.	
	2. Insere cada palavra que compõe o nome do evento em uma linha da tabela qgrams_names, gera os grams de cada palavra e os insere em qgrams_names. O caso de uso é encerrado.

Na Tabela 4.7 é detalhado o caso de uso “Inserir novo evento” que ocorre sempre que o administrador incluir um novo evento no banco de dados.

Tabela 4.8: Descrição do caso de uso “Modificar um evento”

UC 08 – Modificar um evento	
Atores:	Administrador.
Pré-condição:	Realização do caso de uso UC 01 - "Preparar o banco de dados para uso do Mod".
Pós-condição:	Antigo nome do evento retirado das tabelas qgrams_names e qgrams_grams e novo nome do evento inserido pelo administrador presente nas mesmas tabelas.
Descrição:	Ao modificar um evento, é disparada uma trigger no banco de dados que deve excluir as entradas antigas e preparar o nome do evento digitado para ser usado na busca por similaridade
Sequência de eventos:	
Ator	**Sistema**
1. Modifica o nome um evento no banco de dados do TNG.	
	2. Remove das tabelas qgrams_names

	e qgrams_grams as linhas com informações do evento sendo modificado, em seguida insere cada palavra que compõe o novo nome do evento em uma linha da tabela qgrams_names, gera os grams de cada palavra e os insere em qgrams_names. O caso de uso é encerrado.

Na Tabela 4.8 é detalhado o caso de uso "Modificar um evento" que ocorre sempre que o administrador modificar o nome de um evento no banco de dados.

Tabela 4.9: Descrição do caso de uso "Excluir um evento"

UC 09 – Excluir um evento	
Atores:	Administrador.
Pré-condição:	Realização do caso de uso UC 01 - "Preparar o banco de dados para uso do Mod".
Pós-condição:	Nome do evento retirado das tabelas qgrams_names e qgrams_grams.
Descrição:	Ao excluir um evento, é disparada uma trigger no banco de dados que deve excluir as entradas referentes ao evento excluído das tabelas qgrams_names e qgrams_grams
Sequência de eventos:	
Ator	**Sistema**
1. Exclui um evento do banco de dados do TNG.	
	2. Remove das tabelas qgrams_names e qgrams_grams as linhas com informações do evento sendo excluídos. O caso de uso é encerrado.

Na Tabela 4.9 é detalhado o caso de uso "Excluir um evento" que ocorre sempre que o administrador modificar excluir um evento do banco de dados.

Tabela 4.10: Descrição do caso de uso "Inserir um lugar"

UC 10 – Inserir um lugar	
Atores:	Administrador.
Pré-condição:	Realização do caso de uso UC 01 - "Preparar o banco de dados para uso do

	Mod".
Pós-condição:	Lugar inserido pelo administrador presente nas tabelas qgrams_names e qgrams_grams.
Descrição:	Ao inserir um lugar, é disparada uma trigger no banco de dados que deve preparar o lugar digitado para ser usado na busca por similaridade
Sequência de eventos:	
Ator	**Sistema**
1. Insere um novo lugar no banco de dados do TNG.	
	2. Insere cada palavra que compõe o nome do lugar em uma linha da tabela qgrams_names, gera os grams de cada palavra e os insere em qgrams_names. O caso de uso é encerrado.

Na Tabela 4.10 é detalhado o caso de uso “Inserir novo lugar” que ocorre sempre que o administrador incluir um novo lugar no banco de dados.

Tabela 4.11: Descrição do caso de uso “Modificar um lugar”

UC 11 – Modificar um lugar	
Atores:	Administrador.
Pré-condição:	Realização do caso de uso UC 01 - "Preparar o banco de dados para uso do Mod".
Pós-condição:	Antigo nome do lugar retirado das tabelas qgrams_names e qgrams_grams e novo nome do lugar inserido pelo administrador presente nas mesmas tabelas.
Descrição:	Ao modificar um lugar, é disparada uma trigger no banco de dados que deve excluir as entradas antigas e preparar o nome do lugar digitado para ser usado na busca por similaridade
Sequência de eventos:	
Ator	**Sistema**
1. Modifica o nome um lugar no banco de dados do TNG.	

	2. Remove das tabelas qgrams_names e qgrams_grams as linhas com informações do lugar sendo modificado, em seguida insere cada palavra que compõe o novo nome do lugar em uma linha da tabela qgrams_names, gera os grams de cada palavra e os insere em qgrams_names. O caso de uso é encerrado.

Na Tabela 4.11 é detalhado o caso de uso "Modificar um lugar" que ocorre sempre que o administrador modificar o nome de um lugar no banco de dados.

Tabela 4.12: Descrição do caso de uso "Excluir um lugar"

UC 12 – Excluir um lugar	
Atores:	Administrador.
Pré-condição:	Realização do caso de uso UC 01 - "Preparar o banco de dados para uso do Mod".
Pós-condição:	Nome do lugar retirado das tabelas qgrams_names e qgrams_grams.
Descrição:	Ao excluir um lugar, é disparada uma trigger no banco de dados que deve excluir as entradas referentes ao lugar excluído das tabelas qgrams_names e qgrams_grams
Sequência de lugars:	
Ator	**Sistema**
1. Exclui um lugar do banco de dados do TNG.	
	2. Remove das tabelas qgrams_names e qgrams_grams as linhas com informações do lugar sendo excluídos. O caso de uso é encerrado.

Na Tabela 4.12 é detalhado o caso de uso "Excluir um lugar" que ocorre sempre que o administrador modificar excluir um lugar do banco de dados.

Tabela 4.13: Descrição do caso "Excluir do BD tabelas, procedures e triggers usados no Mod"

UC 13 – "Excluir do BD tabelas, procedures e triggers usados no Mod"	
Atores	Administrador

Pré-condição:	Realização do caso de uso UC 01 - "Preparar o banco de dados para uso do Mod"
Pós-condições:	Todas tabelas, procedures e triggers usadas pelo Mod não devem mais existir no banco de dados do TNG
Descrição:	Quando o usuário não pretende mais usar o Mod, com um clique deve ser possível remover do banco de dados todas modificações feitas para tornar o Mod usável.
Sequência de eventos:	
Ator	**Sistema**
1. Usuário clica no botão “Deletar”	
	2. Exibe a mensagem “Deseja realmente deletar”, com as opções “OK” e “Cancelar”
3. Usuário clica no botão “OK”	
	4. É feita a exclusão das tabelas, procedures e triggers utilizadas pelo mod. Uma mensagem de sucesso é então exibida para o Ator. O caso de uso é encerrado.
Fluxo alternativo 01:	
1.1 Usuário clica no botão “Deletar”	
	1.2 Exibe a mensagem “Deseja realmente deletar”, com as opções “OK” e “Cancelar”
1.3 Usuário clica no botão “OK”	
	1.4 É feita a exclusão das tabelas, procedures e triggers utilizadas pelo mod. Ocorre então um erro na tentativa de deletar. É exibida uma mensagem de erro para o Ator. O caso de uso é encerrado.

A tabela 4.13 apresenta o detalhamento da exclusão das tabelas, procedures e triggers usadas pelo Mod. Esta opção se encontra na seção administrativa do site, no menu “Setup”, na aba “Criação de Tabelas”.

Para este caso de uso, é apresentado também um fluxo alternativo, que é a sequência de eventos que deve acontecer caso ocorra uma falha no fluxo principal. Para

este caso, caso ocorra algum problema na preparação do banco de dados, deve ser exibida uma mensagem ao usuário informando o erro.

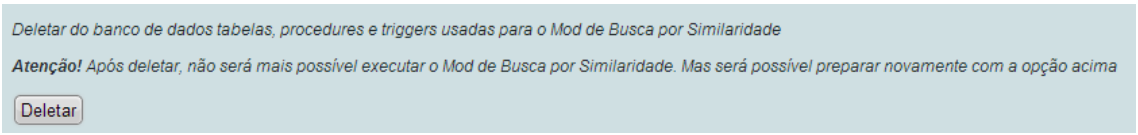

Figura 4.5: Interface do caso de uso “Excluir do BD tabelas, procedures e triggers usados no Mod”

Na Figura 4.5 é mostrada a interface que permite que Administrador exclua do banco das tabelas, procedures e triggers usada no Mod. Ao clicar no botão Deletar, começa o caso de uso.

5 ARQUITETURA DO SISTEMA

5.1 Modelo de dados

A modelagem do banco de dados foi criada a partir da modelagem já existente da aplicação TNG. Das tabelas do TNG, 3 são utilizadas pela aplicação.

A Figura 5.1 procura mostrar a modelagem ER (entidade-relacionamento) do banco de dados. O modelo não apresentará todas as tabelas utilizadas pela aplicação TNG, limitando-se a apresentar apenas as 3 tabelas da TNG utilizadas mais as tabelas criadas pelo Mod de Busca por Similaridade.

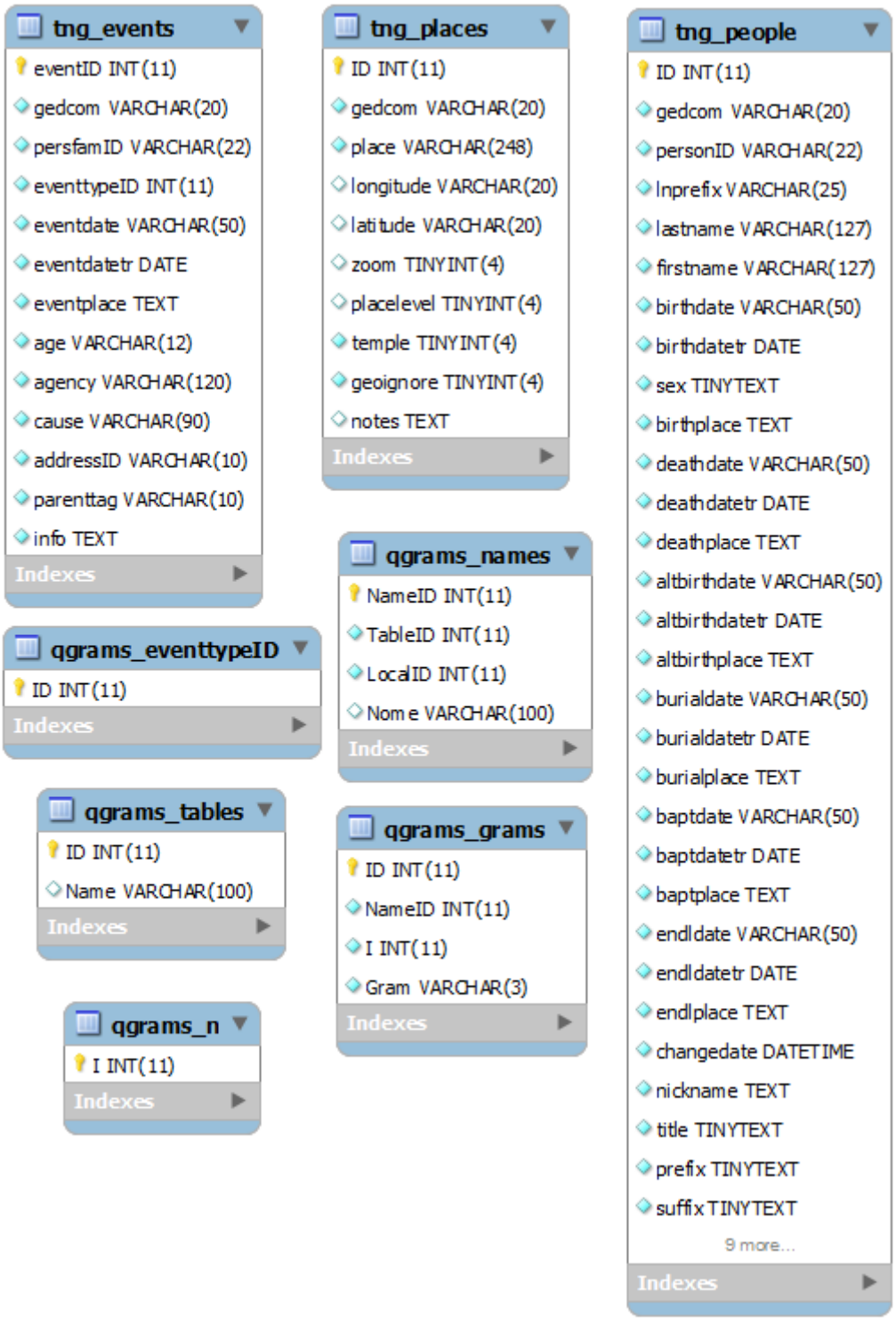

Figura 5.1: Modelo ER da aplicação

A tabela *tng_people* contém informações sobre pessoas, a tabela *tng_events* informações sobre eventos e a tabela *tng_places* informações sobre lugares.

A tabela *qgrams_eventtypeID* é usada para armazenar quais tipos de eventos serão usados pelo Mod. A tabela *qgrams_tables* é usada para armazenar qual o nome sendo usado pelas tabelas de pessoas, eventos e lugares. Isso é necessário pois o TNG permite

a personalização destes nomes, sendo *tng_people*, *tng_events* e *tng_places* apenas os nomes padrões sugeridos.

Como tanto os nomes de localidades como de pessoas em sua grande maioria são compostos, por exemplo, nome e sobrenome, foi criada a tabela *qgrams_names*, que é uma implementação da tabela RA_i descrita na seção 3.3.1, onde todos os nomes da tabelas *tng_places*, *tng_people* e *tng_events* são armazenados em registros separados, isto é, "joão silva" será armazenado em *qgrams_names* com dois registros "joão" e "silva". Isso é feito para que quando uma consulta feita, por exemplo, pelo sobrenome "silva", o registro "joão silva" seja retornado, pois caso o nome inteiro fosse armazenado em *qgrams_names*, as duas strings não seriam similares.

A tabela *qgrams_grams* armazena os *q*-grams de todos os registros presentes na tabela *qgrams_names*. Por último, a tabela *qgrams_I* é uma implementação da tabela *I* mencionada na seção 3.3.1.

5.2 Utilizando *Q*-grams na aplicação

Para exemplificar a implementação, será apresentado uma consulta de uma pessoa, tendo como critério a string *@query* e uma precisão *@k*. Como é uma consulta por pessoa, será necessário realizar uma união entre os resultados das pesquisas na tabela *tng_people* e *tng_events*, pois tabela *tng_events* é usada para armazenar eventos ocorridos com uma pessoa ou em um lugar, mas como esses eventos são personalizáveis muitas vezes é criado um tipo de evento para ser usado como informação de uma pessoa, como por exemplo um apelido. Por isso, na aplicação a busca por pessoas deve considerar tanto a tabela *tng_people* quanto a tabela *tng_events*.

```
01. SELECT *
02. FROM
03. (
04.    (
05.           SELECT       TP.firstame,TP.lastname,
06.                        COUNT(*) as count
07.           FROM         qgrams_names QN
08.           INNER JOIN   tng_people TP ON QN.LocalID = TP.ID
09.           INNER JOIN   qgrams_grams QG ON QG.NameID = QN.NameID
10.           INNER JOIN   t_qgrams TMPG ON TMPG.gram = QG.gram
11.           WHERE        QN.TableID = 1 AND
12.                        QN.NameID = QG.NameID AND
13.                        TMPG.Gram = QG.Gram AND
14.                        QG.I <= TMPG.I + @k AND
15.                        QG.I >= TMPG.I - @k AND
16.                        LENGTH(QN.Nome) <= LENGTH(@query) + @k AND
17.                        LENGTH(QN.Nome) >= LENGTH(@query) - @k
18.           GROUP BY     QG.NameID, QN.Nome
19.           HAVING       COUNT(*) >= LENGTH(QN.Nome) - 1 - (@k-1) * 3 AND
20.                        COUNT(*) >= LENGTH(@query) - 1 - (@k-1)* 3
21.    )
22.
23.    UNION
24.
25.    (
26.           SELECT       TP.firstame, TP.lastname, COUNT(*) as count
```

```
27.          FROM         qgrams_names
28.          INNER JOIN   tng_events TE ON QN.LocalID = TE.eventID
29.          INNER JOIN   tng_people TP ON TP.personID = TE.persfamID
30.          INNER JOIN   qgrams_grams QG ON QG.NameID = QN.NameID
31.          INNER JOIN   t_qgrams2 TMPG ON TMPG.gram = QG.gram
32.          WHERE        QN.TableID = 3 AND
33.                       QN.NameID = QG.NameID AND
34.                       TMPG.Gram = QG.Gram AND
35.                       QG.I <= TMPG.I + @k AND
36.                       QG.I >= TMPG.I - @k AND
37.                       LENGTH(QN.Nome) <= LENGTH(@query) + @k AND
38.                       LENGTH(QN.Nome) >= LENGTH(@query) - @k
39.          GROUP BY     QG.NameID, QN.Nome
40.          HAVING       COUNT(*) >= LENGTH(QN.Nome) - 1 - (@k-1) * 3 AND
41.                       COUNT(*) >= LENGTH(@query) - 1 - (@k-1)* 3
42.    )
43.
44. ) AS I
45. GROUP BY I.ID
46. ORDER BY I.count DESC, I.firstname, I.lastname
```

Duas buscas são feitas neste exemplo, a primeira, entre as linhas 5 e 20 realiza uma busca por aproximação na tabela *tng_people* e a segunda, entre as linhas 26 e 41 realiza uma busca por aproximação na tabela *tng_events*. Como os resultados desejados são apenas de pessoas, na segunda consulta, é realizado um *join*, na linha 29, entre a tabela *tng_people* e *tng_events*. Com este *join* realizado, as duas consultas tornam-se praticamente iguais, será detalhado apenas a primeira.

As linhas 5 e 6 indicam que como resultado da consulta sejam retornados os nomes, sobrenomes e o número de *q*-grams em comum entre a string *@query* e o registro do banco de dados. Este número será usado adiante.

Entre as linhas 7 e 10 são indicadas todas as tabelas que serão necessárias para a consulta. A restrição da linha 11, indica que queremos apenas comparar registros de *qgrams_names* que tenham *TableID = 1*, ou seja, apenas nomes que sejam da tabela *tng_people.* As linhas 14 e 15 realizam a implementação do filtro por posição, as linhas 16 e 17 implementam o filtro por comprimento, enquanto que o filtro por contagem é implementado nas linhas 19 e 20.

Com esta consulta, temos todos os registros da tabela *tng_people* que, com uma precisão *@k* sejam similares a *@query*. Com a segunda consulta, teremos os mesmos resultados vindos da tabela *tng_events.* Então é feita uma união, na linha 23, entre todos os resultados. Os resultados são agrupados pelo seu *ID*, na linha 45, para que resultados duplicados não sejam exibidos, e finalmente ordenados primeiramente pelo número de *q*-grams em comum citado anteriormente, em ordem decrescente, seguido pelo nome e após pelo sobrenome. Desta maneira, ao ordenar em uma ordem decrescente do número de *q*-grams em comum, os nomes mais similares ao consultado serão exibidos primeiro.

A consulta de lugares também é feita usando a mesma lógica acima, porém não é necessário usar duas consultas ao banco, pois os dados de lugares vem apenas da tabela *tng_places*.

Caso a pesquisa seja feita usando duas palavras e não uma, são realizadas duas consultas como a exemplificada acima, e é então realizada uma intersecção entre os

resultados, exibindo assim apenas os que duas de suas palavras são similares às duas pesquisadas.

5.3 Diagramas de Sequência

Com os diagramas de sequência da UML, será é dada uma noção temporal das mensagens trocadas pelos objetos do sistema. Por mensagem, entendemos as solicitações feitas por um objeto ao outro, bem como as respostas devolvidas.

Serão apresentados os diagramas de sequência para os casos de uso “Pesquisa por Pessoas”, “Pesquisa por Lugares”, “Preparar o banco de dados para uso do Mod” e “Excluir do BD tabelas, procedures e triggers usados no Mod”. Como os casos de uso “Inserir nova pessoa”, “Modificar uma pessoa”, “Excluir uma pessoa”, “Inserir novo lugar”, “Modificar um lugar”, “Excluir um lugar”, “Inserir novo evento”, “Modificar um evento” e “Excluir um evento” apresentam todos uma estrutura semelhante, será apresentado apenas o caso de uso “Inserir nova pessoa”, mas a lógica usada nele pode ser empregada nos outros casos.

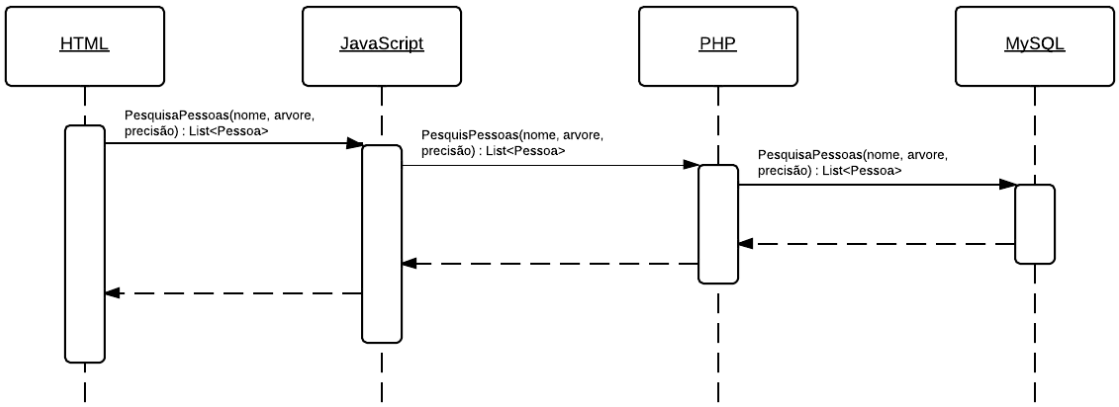

Figura 5.2: Diagrama de sequência do caso de uso “Pesquisa por Pessoas”

A Figura 5.3 detalha as mensagens trocadas entre os objetos no caso de uso “Pesquisa por Pessoas”. É enviada uma solicitação começando no HTML, contendo como parâmetros o nome, a precisão e a árvore desejadas. A mensagem é propagada até o MySQL, que por sua vez retorna uma lista de Pessoas, que é enviada de volta para ser exibida na tela.

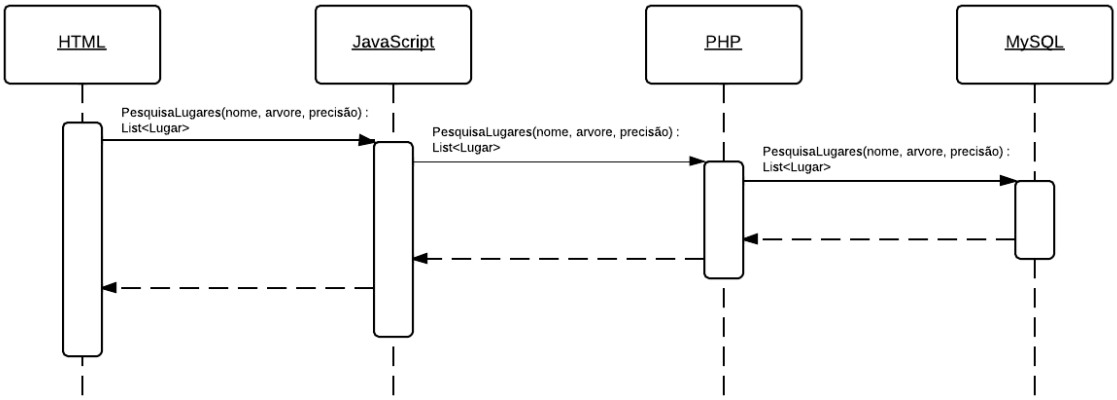

Figura 5.3: Digrama de sequência do caso de uso “Pesquisa por Lugares”

A Figura 5.4 detalha as mensagens trocadas entre os objetos no caso de uso “Pesquisa por Lugares”. É enviada uma solicitação começando no HTML, contendo como parâmetros o nome, a precisão e a árvore desejadas. A mensagem é propagada até o MySQL, que por sua vez retorna uma lista de Lugares, que é enviada de volta para ser exibida na tela.

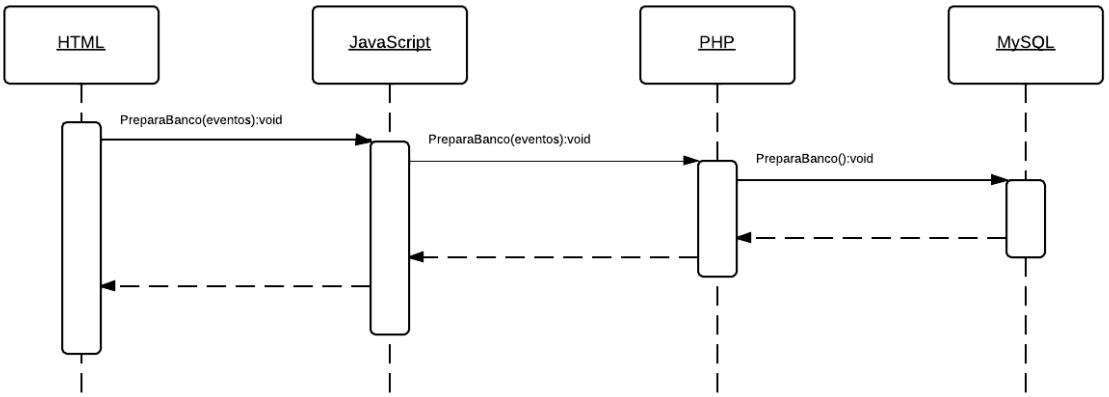

Figura 5.4: Diagrama de sequência do caso de uso “Preparar o banco de dados para uso do Mod”

A Figura 5.5 detalha as mensagens trocadas entre os objetos no caso de uso “Preparar o banco de dados para uso do Mod”. É enviada uma solicitação começando no HTML, contendo como parâmetro os eventos que o usuário deseja que sejam tratados pelo Mod de Busca por Similaridade. A mensagem é propagada até o PHP, que então de posse do parâmetro “eventos” prepara a query e então envia uma mensagem ao MySQL. Não há mensagem de retorno, apenas a confirmação de que a operação foi realizada com sucesso.

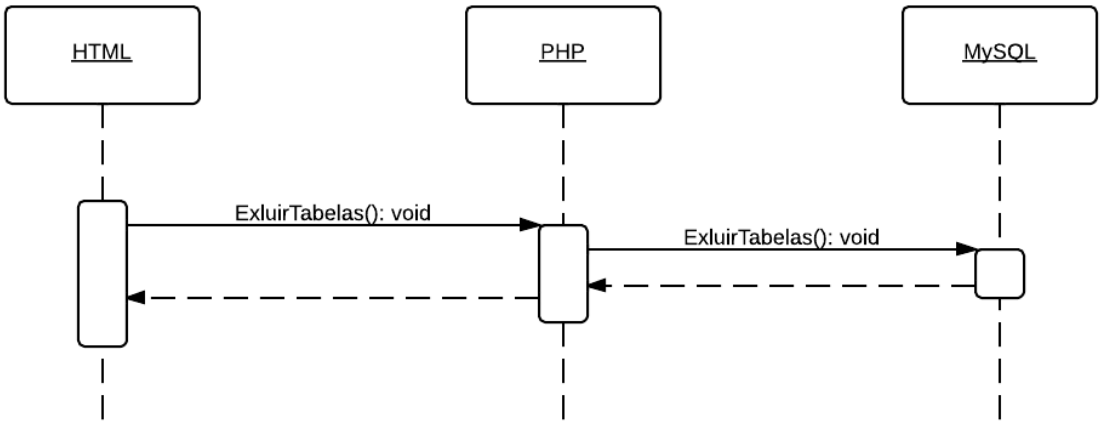

Figura 5.5: Diagrama de sequência do caso de uso “Excluir do BD tabelas, procedures e triggers usados no Mod”

A Figura 5.6 detalha as mensagens trocadas entre os objetos no caso de uso “Excluir do BD tabelas, procedures e triggers usados no Mod”. É enviada uma solicitação começando no HTML, que é propagada até o MySQL. Não há mensagem de retorno, apenas a confirmação de que a operação foi realizada com sucesso.

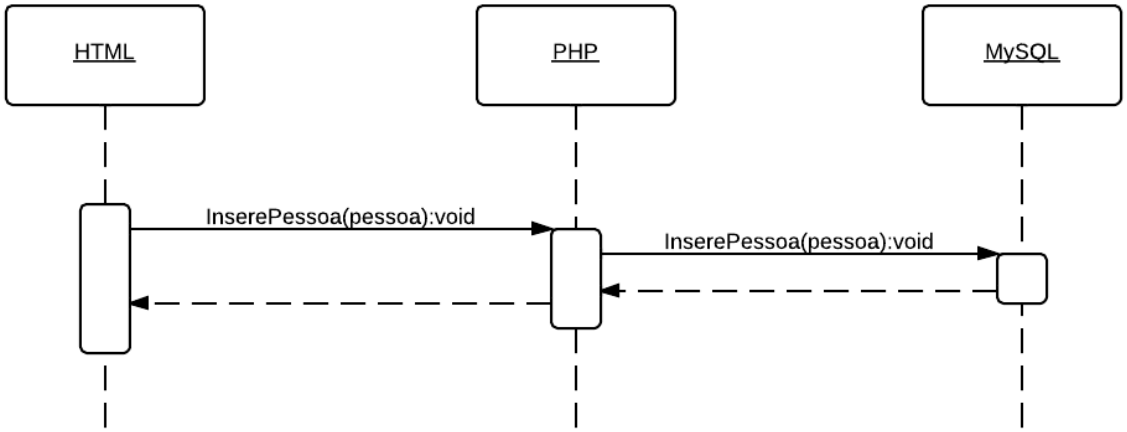

Figura 5.6: Diagrama do caso de uso “Inserir nova pessoa”

A Figura 5.7 detalha as mensagens trocadas entre os objetos no caso de uso “Inserir nova pessoa”. É enviada uma solicitação começando no HTML, tendo como parâmetro a pessoa que se deseja inserir. Que é propagada até o MySQL. Não há mensagem de retorno, apenas a confirmação de que a operação foi realizada com sucesso. Lembrando que, conforme mencionado anteriormente, para os demais casos, a lógica aplicada aqui é a mesma, apenas substituindo a ação por atualizar ou excluir, e o parâmetro por lugar ou evento.

5.4 Detalhes do Mod

Nesta seção, serão apresentadas quais as mudanças que se fizeram necessárias no código PHP, no código Javascript e no banco de dados MySQL utilizado pelo software. Estas mudanças foram feitas em um arquivo de configuração, implementado conforme o explicado na seção 2.1.

5.4.1 Modificações no código PHP e JavaScript

No arquivo *genlib.php*, que funciona como uma biblioteca que é usada por todas as páginas da seção pública do site, foram incluídas referências para os arquivos JavaScript necessários para aplicação, assim como um link para a interface de busca por similaridade, que será apresentado no menu de busca.

Foi criado o arquivo *qgrams_findssform.php* que implementa a interface gráfica para a consulta. O arquivo *qgrams_findplaces.php*, também criado, é responsável por realizar a busca por lugares, por meio da uma consulta ao MySQL, bem como criar a lista com os resultados. O arquivo *finditems.php*, por sua vez, foi modificado para realizar a busca por pessoas, novamente por meio de uma consulta ao MySQL e criar a tabela com os quais os resultados de pessoas serão exibidos.

O arquivo *selectutils.js* foi modificado para implementar a realização da busca a cada letra que o usuário digita nos campos de pesquisa. É ele o responsável por, por meio de JSON, realizar uma solicitação aos arquivos *qgrams_findplaces.php* ou *finditems.php* para que eles realizem a busca, e então ele exibe na página *qgrams_findssform.php* os resultados retornados.

Para as mudanças no setor administrativo do site, foi necessário modificar arquivo, *admin_setup.php.* Nele foi implementado a interface para que o usuário possa realizar a preparação necessária no banco de dados para que o Mod funcione corretamente. Criou-se o arquivo *qgrams_admin_prepare_database.php,* que contém uma função JavaScript, que por meio de JSON faz uma solicitação ao arquivo *qgrams_tabledefs.php.* Este arquivo, também criado, contém as scripts que realizam as modificações necessárias no banco de dados MySQL. Os arquivos *qgrams_admin_delete_database.php* e *qgrams_delete_tabledefs.php* foram criados com a mesma estrutura dos dois anteriores, porém realizam a exclusão das modificações realizadas no banco de dados para o uso do Mod.

Finalmente, foram realizadas modificações que inserem os textos usados nas intefaces nos arquivos *text.php* e *admintext.php*, em suas versões em português, inglês e alemão. Tais modificações permitem que o Mod seja usado em qualquer uma das três línguas, de acordo com a língua que está definida para ser usada pelo TNG. Caso outra linguagem esteja selecionada, não serão exibidos textos na interface.

5.4.2 Modificações no banco de dados MySQL

Conforme citado na seção anterior, o arquivo *qgrams_tabledefs.php* executa no banco de dados os scripts necessários para prepara-lo para ser usado pelo Mod. Em um primeiro momento, são executados scripts que criam as tabelas descritas na seção 5.1.

Como as tabelas *tng_people*, *tng_events* e *tng_places* provavelmente já conterão vários dados, faz-se necessário realizar um processamento dos dados já presentes no banco para serem usados para a busca por similaridade. Para isto, são criadas 5 *procedures* no banco de dados: *explode_people(), explode_events(), explode_places(), explode_places_virgula()* e *explode_places_espaco().* Conforme mencionado na seção 3.3.4, a busca sempre será feita por strings que contenham apenas uma palavra, e para que isto seja feito, estas *procedures* selecionam todos os nomes de suas respectivas tabelas do TNG e inserem na tabela *qgrams_names* todos os nomes já separados, cada palavra em um registro diferente. Após a inserção em *qgrams_names* já são gerados os *q*-grams da palavra e os mesmos são inseridos em *qgrams_grams.*

As *procedures qgrams_places_virgula() e qgrams_places_espaco()* se fazem necessárias pois os lugares são armazenados, em sua maioria, contendo mais de uma informação, separadas por vírgula, como por exemplo: “Porto Alegre, Rio Grande do Sul, Brasil”. Por isso, é necessário primeiro separar os lugares por vírgula, para então realizar a separação de cada palavra.

Tais procedures precisam ser chamadas apenas uma vez, após a instalação do Mod, e, uma vez executados, todas as pessoas e lugares presentes no banco de dados já podem ser procuradas por similaridade.

Com o tempo, o usuário provavelmente fará alterações nos seus dados, como inserir novas pessoas, modificar lugares já existentes ou mesmo excluír algum registro. Por isso, é necessário que o Mod tenha um mecanismo para que os *q*-grams estejam sempre atualizados com as informações do banco de dados. Isso é feito por meio de *triggers*, que é um recurso existente no MySQL, que permite que algo seja feito quando um determinado evento ocorra em alguma tabela.

Para cada uma das tabelas *tng_people*, *tng_events* e *tng_places*, foram criados 3 *triggers*. O primeiro tipo de *trigger* criado é disparado sempre que um registro é

inserido na tabela. São eles: *people_insert*, na tabela *tng_people; events_insert* na tabela *tng_events* e *places_insert*, na tabela *tng_places.* Ao ser inserido um novo nome, automaticamente as palavras do nome são separadas e inseridas em *qgrams_names* e os *q*-grams de cada palavra inseridos em *qgrams_grams*.

O segundo tipo de *trigger* criado é disparado sempre que um registro de uma das tabelas é atualizado. São excluídos das tabelas *qgrams_names* e *qgrams_grams* os valores antigos associados ao registro, e os novos valores são automáticamente excluídos. Os *triggers* criados são: *people_update*, na tabela *tng_people; events_update* na tabela *tng_events* e *places_update*, na tabela *tng_places.*

O terceiro e último tipo é disparado sempre que um registro é excluído de uma das tabelas. Os dados associados ao registro excluído são excluídos também das tabelas *qgrams_names* e *qgrams_grams.* Os *triggers* criados são: *people_delete*, na tabela *tng_people; events_delete* na tabela *tng_events* e *places_delete*, na tabela *tng_places.*

Para que os novos valores sejam inseridos nas tabelas *qgrams_names* e *qgrams_grams* quando uma *trigger* é disparada, são criadas as *procedures trigger_people(string), trigger_events(string), trigger_places_virgula(string), trigger_places_espaco(string)* e *trigger_places(string).* Assim como as *procedures* mencionadas anteriormente, estas separam as palavras dos nomes que estão sendo inseridos ou atualizados no banco de dados, inserem em *qgrams_names* e obtém todos os *q*-grams das palavras e inserem em *qgrams_grams.* O que as diferencia das *procedures* anteriores são que estas trabalham com apenas 1 registro por vez, enquanto as outras trabalham com todos os registros do banco de dados.

6 CONCLUSÃO

Esta trabalho apresenta uma personalização para o software TNG que possibilita que os usuários realizem uma busca por similaridade. O software permite que genealogistas apresentem suas informações em um website de uma maneira rápida e dinâmica. Para realizar a personalização, foi desenvolvido um Mod, que são arquivos de configuração que permitem que alterações sejam feitas no código do TNG.

Apesar de apresentar muitas opções para a pesquisa nos dados do software, tais pesquisas necessitam que o usuário saiba a palavra exata que deseja pesquisar, o que não é o ideal em pesquisas genealógicas, vistos que nomes de pessoas e lugares são suscetíveis a erros de grafia e a mudanças que ocorrem ao longo do tempo ou por variações entre linguagens.

Para a realização da busca por similaridade, foi utilizada a técnica apresenada por Gravano (2005). Com ela, é possível realizar uma busca em um banco de dados a partir de strings aproximadas, sem que seja necessário o uso de aplicações externas ou com o baixo desempenho das UDFs.

Assim, o Mod desenvolvido realiza modificações no código em PHP do software para exibir uma interface gráfica para a busca por similaridade e também realiza modificações no banco de dados MySQL na aplicação, gerando novas tabelas e utilizando *triggers* para manter as informações sempre atualizadas.

Após algum tempo de uso por usuários do Mod, seria interessante obter um retorno sobre sugestões de melhoria na interface da busca por similaridade. Algumas melhorias podem ser feitas, como por exemplo, exibir nos resultados o apelido da pessoa caso ele exista. Com a versão atual, caso uma busca seja feito, por exemplo, pelo apelido “Chico”, somente serão exibidos na tela resultados com os nomes “Francisco”, o que pode confundir o usuário.

Outro ponto interessante de realizar uma melhoria é no número de palavras que pode ser pesquisado. Hoje é permitido realizar uma pesquisa apenas com 1 ou 2 palavras. Seria interessante aumentar esse número, ou ainda implementar uma maneira dinâmica, onde o usuário poderia entrar com quantas palavras achar necessário.

REFERÊNCIAS

TNG. Disponível em http://lythgoes.net/genealogy/software.php. Acesso em jun. 2013

ALHIR, Sinan Si: Understanding the Unified Modeling Language (UML). In: **Methods & Tools – Spring 1999**. Vevey, Suiça: Martinig & Associates, 1999. p. 11 – 18.

GRAVANO, Luis, et. al.: Using q-grams in a DBMS for Approximate String Processing. In: **Bulletin of the Technical Committee on Data Engineering.** Washington, D.C, EUA: IEEE Computer Society, 2001. p. 28 – 34.

UKKONEN, Esko: **Approximate string matching with q-grams and maximal matches**. Theoretical Computer Science, 1992. p. 191–211.

ULLMANN, J. R.: **A binary n-gram technique for automatic correction of substitution, deletion, insertion and reversal errors in words.** The Computer Journal, 1997. p. 141–147.