Sistema de Apoio para Localização de Pessoas Desaparecidas - SALPD

Arthur Schuelter, Felipe Weiss, Lucas Litter Mentz

24 de junho de 2018

1 Introdução

Este trabalho foi desenvolvido como trabalho final da disciplina Bancos de Dados II do curso Bacharelado em Ciências da Computação.

A aplicação desenvolvida tem como ideia auxiliar na busca e resgate de pessoas desaparecidas, tomando o papel de ser um aplicativo agregador de informações inseridas por agentes, informantes ou anônimos.

Desenvolvemos nossa aplicação em linguagem de programação C++ com a interface objeto-relacional ODB (CodeSynthesis) para auxiliar na persitência dos dados. Nossa aplicação não conseguiu chegar ao ponto de funcionamento pleno mas as funcionalidades implementadas apresentam que é possível fazer um sistema objeto-relacional com esse propósito.

2 Modelagem do banco de dados

No início do nosso desenvolvimento tínhamos como ideia modelar grande parte da aplicação na própria base de dados, com uso de triggers e funções para aplicação das regras de negócio necessárias para a aplicação, então tínhamos o esquema de base de dados similar ao apresentado na figura 1. A diferença entre o diagrama de schema que tínhamos e o apresentado na figura 1 é que o anterior tinha os campos de data/hora usando o tipo de dado `timestamp` do PostgreSQL, enquanto na versão atual temos tipos de dados `bigint` para correlacionar com o `time_t` do C (`long int`).

Nossa escolha por permitir valores nulos em RG e CPF de pessoas se baseia na possibilidade de uma pessoa desaparecida, durante o registro, não possuir documentos ou não ter sido registrada. Ainda assim é necessário encontrá-la, pois todas as vidas importam. O atributo '*nome*' é um único

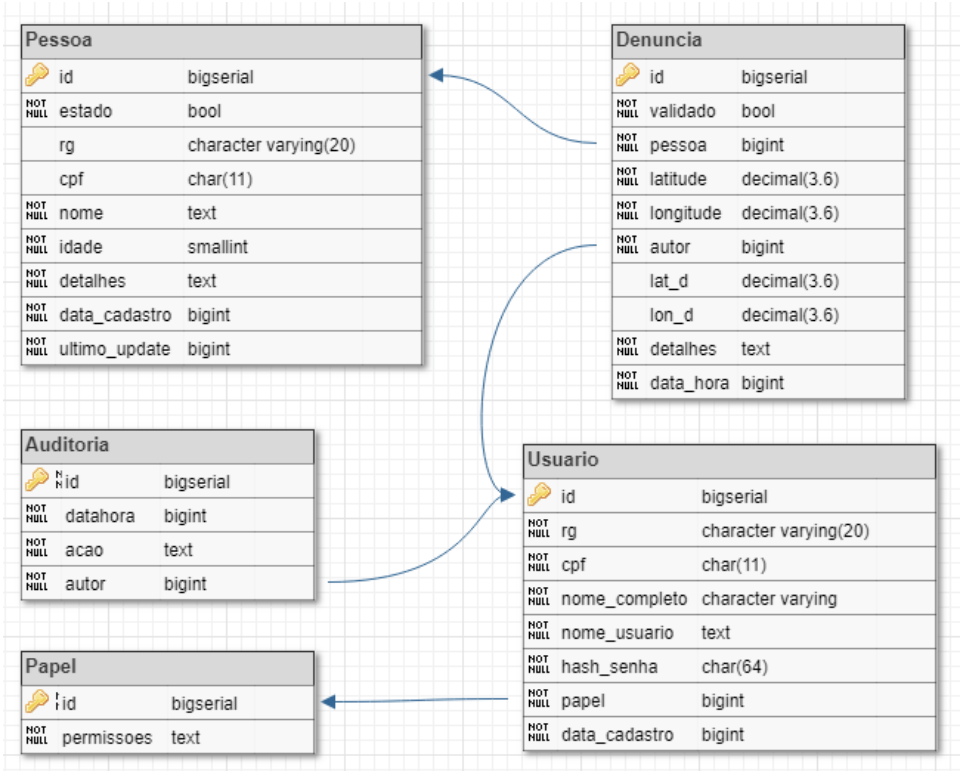

Figura 1: Diagrama de *schema*. Produção própria.

campo para facilitar buscas, que seriam modeladas similarmente a `SELECT ... FROM Pessoa WHERE nome LIKE ('%<nome e/ou sobrenome>%')`. Para facilitar descrição visual da pessoa temos os campos idade e detalhes. Também temos o atributo *estado* que se refere a se a pessoa está atualmente desaparecida (0) ou se foi encontrada (1). Por motivos de simplicidade, e de que provavelmente não cabe a nós mantermos informações de óbito, mantivemos somente essas duas possibilidades de estado de uma pessoa.

Usamos a tabela Denuncia para manter o histórico das denúncias, e não somente a mais recente para cada pessoa. Denúncias tem um atributo *pessoa* que é chave estrangeira para a pessoa que se referem. As denúncias inseridas por usuários cadastrados (informante, agente, gestor) são válidas por padrão, ou seja, são confirmadas como verdadeiras e portanto servem para localização de pessoas desaparecidas. Contudo, denúncias inseridas pelo usuário anônimo – entidade que representa todas as formas possíveis de entrada de informações de maneira anônima, tal como por telefone em

um sistema automatizado, pela internet, em delegacias de polícia por meio de formulário de papel que posteriormente é digitado por um digitador profissional que não necessariamente seja um agente, informante ou gestor do sistema SALPD – são marcadas como não válidas (bool *validado* = false). As denúncias não validadas devem primeiro ser verificadas por usuários verificadores (agente ou gestor) que farão a eventual substituição de informações mal-formatadas ou a remoção de uma entrada que contém informações incompletas ou enganosas – tentativa de trote, por exemplo. Cada denúncia inserida guarda a referência para o autor da denúncia, o usuário que a inseriu. As denúncias possuem campos para entrada de *latitude* e *longitude* de onde a pessoa fora encontrada, e *lat_d* e *lon_d* para latitude e longitude aproximada do denunciante (caso seja anônimo) – essas informações seriam obtidas com triangulação para telefone móvel e localização de cadastro com a operadora, caso seja telefone fixo ou internet. O atributo *detalhes* serve para entrada de informações adicionais, em formato de texto, que não caiba em nenhum campo da tabela, como a fonte da informação (se for anônimo pode incluir endereço IP ou número de telefone), precisão da localização, se a denúncia se trata de relatar que a pessoa foi encontrada, entre outros.

A relação Usuario guarda informações de usuários cadastrados, sendo somente um deles (usuário 0) o usuário anônimo e outro o administrador padrão (usuário 1). Esses dois usuários são cadastrados no momento de inicialização da base de dados, que envolve a execução do arquivo AllClass.sql que contém o schema de dados e o arquivo Inicial.sql que contém os dados iniciais, ambos os arquivos foram gerados automaticamente pelo ODB. Os atributos *nome_usuario* e *hash_senha* definem as credenciais de login de um usuário na aplicação. Por motivos de segurança, *hash_senha* não é a senha do usuário, mas sim o resultado do hash SHA256 da senha dele, dessa forma a base de dados não guarda de nenhuma forma credenciais de usuários, porém consegue verificar se o usuário forneceu a senha correta. Para fins de auditoria e segurança, cada usuário cadastrado tem seu RG, CPF e nome completo no sistema. Nossa modelagem permite somente um papel por usuário, ou seja, se o usuário for um informante, não pode assumir a função de um gestor, e vice-versa.

A tabela Papel guarda os tipos de permissões que usuários podem assumir no seu trabalho. Essa tabela possui somente a coluna *id* para identificar de forma única um papel, e a coluna *permissoes* que guarda texto formatado descrevendo permissões daquele papel. Desta forma, o usuário anônimo teria uma ligação com o item de id 4 desta tabela, que poderia ser definido como: (4, "r pessoa w denuncia"), informando que o papel 4 – Anônimo – possui permissão de leitura na tabela Pessoa e de escrita na tabela Denuncia,

somente.

Auditoria guarda as operações executadas por usuários do sistema. Optamos por uma implementação simples na qual cada operação aciona um trigger que põe a query SQL como conteúdo do atributo *acao* e relaciona qual o usuário que fez a ação, assim como o momento da ação. Modelamos esta tabela já pensando na implementação C++ com interface objeto-relacional ODB.

Após a definição desta modelagem puramente relacional nós decidimos por descartar a implementação das funções e triggers SQL e o código C++ existente (pouco, apenas logava no sistema) para tentar uma solução mais "elegante"com uso de uma interface objeto-relacional em C++, com base nos modelos de dados (classes) do diagrama de schema apresentado na figura 1.

3 Implementação

Como comentado na seção anterior, optamos por implementar nossa versão do SALPD com foco na programação em C++ e uso de uma interface objeto-relacional. Optamos pela interface objeto-relacional CodeSynthesis ODB (ODB) por ter pacotes disponíveis para as distribuições Linux que usamos e por ter relativamente fácil implementação sobre o código C++ existente.

Tal decisão acabou atrasando o desenvolvimento pois saímos de uma implementação com a qual tínhamos certa familiaridade (apenas um "envelopamento"de C++ em cima de consultas SQL) para algo novo que parecia muito atrativo, porém não tínhamos experiência alguma. Por conta de uma boa documentação disponível, não foi difícil ter uma implementação de exemplo – um hello world C++ com persistência – funcionando em poucos dias, porém repensar a implementação das relações entre tabelas, consultas SQL e triggers para ser implementado puramente em C++ nos tomou mais tempo do que o esperado.

Basicamente, na nossa implementação, temos todas as regras de negócio e todas as checagens definidas em C++, e usamos o ODB para garantir persistência e controlar concorrência por meio do uso de transações nas operações que fazem acesso ao banco de dados.

A manipulação de dados, como por exemplo a criação de tabelas, ou busca de dados é feita a partir de um Objeto de Acesso aos Dados (DAO). Implantamos uma visualização (*view*) para facilitar consultas à base de dados de dentro do programa. A *view* utilizada fornece as pessoas desapareci-

das que não tiveram denúncias nos último dia.

O programa começa com uma tela de login onde é definido qual será o usuário que estará acessando o sistema. Dependendo do usuário que fizer o login, abre um menu específico. O admin, por exemplo, vai ter uma tela onde poderá cadastrar novos usuários enquanto que na tela do anônimo teremos as opções de denúncia anônima e de visualização da *view* implementada. Com base nisso, cada usuário só terá acesso as funções dentro de suas limitações.

Todas as grandes funções que envolvem verificações e tratamentos ou acesso ao banco de dados é feito através do DAO. Desta forma, a implementação inteira basicamente é feita sustentado pelas implementações do DAO. Dentro do DAO é feito todos acessos de busca, atualização, remoção e

Implementamos uma possibilidade de *view* que apresenta uma tabela de pessoas que não tiveram atualizações (ou denúncias) nos últimos 15 dias. Essa *view* poderia ser usada para encontrar pessoas de maior risco, ou favorecer buscas para essas pessoas. A *view* está disponível para qualquer usuário (logado como anônimo) para permitir que qualquer pessoa que chegue no sistema possa ver quais pessoas estão perdidas a mais tempo sem ter denúncias.

4 Conclusão

O desenvolvimento deste trabalho uniu conceitos vistos durante todo o semestre de Bancos de Dados II, e serviu de lição para aprendermos que mudar totalmente a forma de fazer um trabalho próximo da data de entrega não é ideal para o desempenho do trabalho.