Relatório Técnico sobre o Desenvolvimento do Banco de Dados da Fundação Oswaldo Cruz

Autor: Beatriz Borges de Oliveira

Data: 13 de Novembro de 2023

RESUMO:

Este relatório técnico apresenta detalhadamente o processo de desenvolvimento de um banco de dados obtido no site do Governo Federal. O projeto abrange desde a importação dos dados em formato CSV, a modelagem do banco de dados, a inserção de registros, a criptografia de dados sensíveis, a criação de uma view e a automatização de todos esses processos através de uma função.

INTRODUÇÃO:

Este trabalho foi proposto com a intenção de desenvolver etapas de normalização, criação de tabelas, exploração de funções e extensões PostgreSQL, entre outros conhecimentos adquiridos ao longo de sua realização. O início do trabalho deu-se ao fazer o download de um arquivo CSV diretamente do site do Governo Federal denominado "***Lista de Terceirizados da Fiocruz***". Este relatório descreve os procedimentos utilizados para o desenvolvimento completo do projeto, desde a importação do arquivo até a criação da função que automatiza os processos.

IMPORTAÇÃO DE DADOS:

O script SQL foi elaborado em etapas utilizando a ferramenta Visual Studio Code. Para rodar os scripts, utilizei um sistema de gerenciamento de banco de dados relacional de código aberto, o PostgreSQL. Inicialmente, utilizei-o para criar um banco de dados e dar início à importação do arquivo para dentro dele. Para criar o banco de dados, dei o comando "*CREATE DATABASE terceirizados_fiocruz;*". Em seguida, no VSCode, o primeiro script SQL foi escrito para realizar a leitura do arquivo CSV e criar uma tabela temporária dentro do banco de dados, tabela que nomeei de “terceirizado”. O comando "*DROP TABLE IF EXISTS terceirizado;*" foi utilizado para garantir a exclusão da tabela caso já existisse. A importação dos dados do arquivo CSV para a tabela "terceirizado" foi realizada com o comando "*COPY... FROM '/caminho do arquivo/terceirizado.csv'*

DELIMITER ';' CSV HEADER", utilizando o ponto e vírgula como delimitador de campo e a primeira linha do arquivo CSV como cabeçalho.

MODELAGEM DO BANCO DE DADOS:

Após a importação inicial, deu-se início à fase de modelagem do banco de dados. Nesse processo, optei por usar um caderno e escrever manualmente possíveis tabelas com os dados, seguindo a 1FN, 2FN e 3FN. Após fazer um rascunho manual de como relacionaria cada tabela, iniciei outro script sql baseado no meu rascunho manual, que cria várias tabelas a partir dos dados não normalizados da tabela "terceirizado", melhorando a integridade dos dados. Essas tabelas possuem estruturas mais específicas para eliminar informações duplicadas e irrelevantes. As tabelas foram criadas com o comando "*CREATE TABLE [nome da tabela]*", especificando o nome de cada coluna e seu respectivo tipo de dado. Um foco especial foi dado para garantir que as chaves estrangeiras e primárias fossem definidas de forma a permitir a interligação das tabelas e a realização de consultas eficientes, facilitando a relação de dados em uma única consulta. É possível ver o arquivo do projeto de criação de tabelas no Apêndice B deste relatório.

INSERÇÃO DE DADOS:

No mesmo script em que criei as tabelas normalizadas, utilizei os comandos "*INSERT INTO*". Esse comando foi responsável por inserir dados a partir do arquivo CSV original nas novas tabelas criadas. Os dados foram selecionados a partir de comandos "*SELECT... FROM terceirizado*", especificando quais colunas de "terceirizado" seriam colocadas em cada nova tabela.

CRIPTOGRAFIA:

Iniciei aqui um outro script sql. Com o objetivo de reforçar a segurança dos dados sensíveis, empreguei a extensão ***pgcrypto*** para cifrar informações que identifiquei como sensíveis nas tabelas "funcionario" e "contrato". A instalação da extensão foi realizada no banco de dados por meio do comando "*CREATE EXTENSION IF NOT EXISTS pgcrypto;*".

Com a finalidade de aprimorar a segurança, adotei a seguinte abordagem:

- Criei um esquema denominado "seguranca" (*CREATE SCHEMA seguranca;*) para assegurar que a chave de criptografia não esteja facilmente acessível no esquema "public".

- Em seguida, instaurei uma tabela denominada "configuracoes" dentro desse esquema (*CREATE TABLE seguranca.configuracoes;*).
- Esta tabela possui duas colunas: uma chamada "chave", que atua como chave primária, e outra chamada "valor".
- Os dados foram inseridos da seguinte maneira: na coluna "chave", inseri o texto "ENCRYPTION_KEY", e na coluna "valor", invoquei uma função da extensão pgcrypto para gerar uma chave aleatória e armazená-la na coluna "valor" toda vez que o script for executado (*INSERT INTO seguranca.configuracoes (chave, valor) VALUES ('ENCRYPTION_KEY', gen_salt('bf');*).
- Posteriormente, determinei que uma variável declarada no início da função (*DECLARE ENCRYPTION_KEY TEXT;*) teria o mesmo valor da coluna "valor" da tabela seguranca.configuracoes (*SELECT valor INTO ENCRYPTION_KEY FROM seguranca.configuracoes WHERE chave = 'ENCRYPTION_KEY';*).

Após a geração da chave de criptografia e a criação de uma tabela segura para armazená-la, ajustei o tamanho das colunas dos dados a serem criptografados. Isso se fez necessário, uma vez que o tamanho originalmente especificado na criação das tabelas não seria suficiente para acomodar o extenso volume de caracteres gerado pela criptografia. Para realizar esse ajuste, utilizei o comando "*ALTER TABLE... ALTER COLUMN... TYPE VARCHAR[qtd. de caracteres]*". Dessa forma, as colunas de cada tabela contendo dados sensíveis foram adaptadas.

Após esses procedimentos, a criptografia dos dados sensíveis foi executada por meio da função *pgp_sym_encrypt( )***,** a qual utiliza o nome das colunas das tabelas contendo os dados a serem criptografados e a variável que armazena a chave de criptografia como parâmetros ([*coluna] = pgp_sym_encrypt([coluna]***,** *ENCRYPTION_KEY)*).

CRIANDO A VIEW:

Com o propósito de viabilizar a visualização dos dados descriptografados e denormalizados, ou seja, sem estarem estruturados em tabelas normalizadas, optou-se por criar uma view denominada "view_dados". Esta view foi projetada não apenas no esquema padrão, mas também armazenada no esquema "seguranca" para armazenar os dados descriptografados de forma mais segura. A view apresenta os dados (considerados sensíveis) após passarem pelo processo de descriptografia, utilizando a função "*pgp_sym_decrypt( )*" que requer como parâmetros o nome das colunas da tabela dos dados a serem descriptografados e a variável que armazena a chave de criptografia.

Para que a view_dados retornasse os dados denormalizados e já descriptografados, foi necessário utilizar a instrução SQL "*CREATE VIEW OR REPLACE VIEW view_dados AS SELECT...*", especificando os nomes dos campos de todas as tabelas no comando "*FROM"* e realizando as junções necessárias. Antes de criar a view, a instrução "*EXECUTE*" foi empregada, para executar dinamicamente a instrução SQL. Neste contexto, a criação da visualização "view_dados" ocorre por meio de SQL dinâmico, permitindo a elaboração e execução de instruções SQL com base em valores conhecidos apenas durante a execução do código.

PROJETO FÍSICO

Para representar o projeto físico do banco de dados, elaborei um diagrama simplificado que exibe todas as tabelas, suas colunas e relacionamentos. Para uma visualização detalhada do diagrama físico do banco de dados, consulte o *Apêndice A*. Chaves primárias e estrangeiras foram destacadas em cada tabela, evidenciando os relacionamentos entre elas. Fiz esse projeto na ferramenta **DB Design** (link nas referências).

CRIANDO A FUNÇÃO

Para concluir, desenvolvi uma função denominada "*executar_script( )*" que engloba todas as etapas do processo, desde a importação dos dados até a criação da visualização descriptografada, declaração de variável e afins. Essa abordagem proporciona uma solução automatizada e eficiente para o gerenciamento do banco de dados. Todos os scripts, originalmente criados separadamente, foram integrados dentro da função, resultando em um único script que executa todas as etapas quando a função é chamada. Essa abordagem permitiu a eliminação dos arquivos SQL individuais, simplificando o código. Ao chamar a função, ela retorna uma tabela com os dados da view localizada no esquema "seguranca". Essa consolidação contribui para a praticidade e eficiência na execução do processo como um todo. A função foi criada por meio do comando *CREATE OR REPLACE FUNCTION executar_script ( ) RETURNS TABLE ([campos da view a serem retornados]) AS $$ DECLARE ENCRYPTION_KEY TEXT; BEGIN …. RETURN QUERY SELECT * FROM seguranca.view_dados; END $$ LANGUAGE plpgsql;).* Por fim, excluí a tabela temporária (intitulada de “terceirizado”) utilizada para importação dos dados.

CONCLUSÃO:

O projeto atingiu com sucesso o objetivo de desenvolver um banco de dados seguro e eficiente, com foco especial na proteção de dados sensíveis. A função simplifica a execução de todas as etapas do projeto, e ao ser chamada executa sem nenhuma falha ou erro. A normalização das tabelas tornou o banco de dados mais gerenciável e eficiente. Além disso, o desenvolvimento deste projeto proporcionou um significativo conhecimento na linguagem SQL.

REFERÊNCIAS:

POSTGRESSQL GLOBAL DEVELOPMENT GROUP. (2022). PostgreSQL Cryptographic Functions (pgcrypto). **PostgreSQL.** Disponível em: <https://www.postgresql.org/docs/current/pgcrypto.html>

GOVERNO FEDERAL. GOV.BR. Recursos Humanos da Fiocruz. Portal Brasileiro de Dados Abertos. Disponível em: <https://dados.gov.br/dataset/recursos-humanos-da-fiocruz>

DB DESIGN. Disponível em: <https://app.dbdesigner.net>

APÊNDICE A

TERCEIRIZADOS FIOCRUZ – BD PROJETO FÍSICO SIMPLIFICADO