Relatório
Projeto AppCovid

SGBD 2019/2020

ISTEC

Licenciatura Engenharia Multimédia

Realizado por:

Luís Morgado, 21419

Carlos Duarte, 21421

David Neves, 21418

Introdução

Como organizamos o projeto:

Inicialmente, analisámos como a base de dados iria comunicar com cada uma das diferentes componentes: informações pessoais, histórico médico e rede de contactos.
Utilizamos a ferramenta “draw.io” para facilitar a visualização gráfica de como os dados estariam organizados. Deste modo, foi possível ter uma noção prévia da estrutura, o que ajudou a escolher o tipo de base de dados mais indicado para as diferentes componentes.

Objetivos:

- Criar uma aplicação, suportada por bases de dados, que permite gerir os dados de doentes infetados ou suspeitos com COVID-19 e a sua rede de contactos próximos.
- Utilizar a base de dados que melhor se adapta para cada tipo de dados.
- Desenvolver scripts em python para comunicar com as bases de dados e criar um ambiente virtual para facilitar a gestão de bibliotecas python.

Desenvolvimento

Informações Pessoais:

Para este tipo de dados escolhemos utilizar o SQLAlchemy ORM, visto que são dados com um esquema constante.
A utilização de um ORM (Object Relational Mapper) permite, também, uma abstração do modelo de dados e uma mais fácil utilização da base de dados. O recurso a uma base de dados relacional como o SQLAlchemy permite que os dados garantam disponibilidade e consistência, o que para este tipo de informação pessoal é essencial.

Histórico Médico:

Para este tipo de dados também utilizámos o SQLAlchemy ORM, evitando a utilização de mais bibliotecas python e integrando mais facilmente as duas bases de dados: “informações pessoais” e “histórico médico”.

Desenvolvimento

Rede de contactos:

Para a rede de contactos pareceu-nos óbvio a utilização de uma base de dados em grafo, dado que neste tipo de base de dados as relações têm tanta importância quanto os atributos.
Utilizámos o neo4j para esta base de dados, em que o principal objetivo é atribuir e identificar relações entre pessoas e locais. Recorremos, também, a um sistema que atualiza os dados das pessoas com a base de dados das informações pessoais, de forma a evitar inconsistências entre as duas.

Exercícios 2.2.1

Os scrips para criação, remoção e alteração de dados das bases de dados anteriormente referidas foram feitas com vista a verificar se o argumentos inseridos já existem ou se não na base de dados, de forma a evitar erros e saber qual o problema.

Desenvolvimento

Exercícios 2.2.2

i) Neste exercício importamos o mysql.connector, sqlalchemy e do argv do sys. Criarmos uma engine para fazer ligação à base de dados app_covid, e armazenamos-a na variável db. Adicionamos à variável Base a função declarative_base() para poder criar as classes mais tarde.

Criamos a classe InfPes com os respetivos atributos e nome da tabela.

Estabelecemos uma ligação à base de dados através de sessões, armazenando na variável Session a função sessionmaker() e criando uma sessão através de Session() que é armazenada na variável s.

Guardamos na variável ind uma query que vai buscar à tabela infpes o nif indicado depois do nome do ficheiro, “py procura_pessoa.py nif”. Se não encontrar nenhuma pessoa faz print da mensagem ‘’Pessoa não encontrada: nif” caso contrário faz print da mensagem “Pessoa encontrada: nif nome data de nascimento email telefone código de postal localidade”.

ii) Fazemos os mesmos imports, criamos as variáveis db, Base, Session e s todas já mencionadas e explicadas no exercício anterior. Criamos a mesma classe InfPes. E guardamos na variável res a query à tabela infpes sob a condição de que o atributo localidade tinha de ser igual a argv[1] o 1º elemento do input para chamar o script procura_localidade.py, que seria a localidade.

Se esta query retorna-se tuplos fazia print de todas as as informações das pessoas que têm essa localidade. O print é igual ao formato do 1º exercício.

Caso o res não retorne nada faz print de “Nenhuma pessoa encontrada da localidade: localidade”.

Desenvolvimento

Exercícios 2.2.2 (Continuação)

iii) Importamos e estabelecemos as variáveis db, Base, Session e s. Criamos as classes InfPes, I_H, Consul_Med, Sintomas, Con_Med, Medicacao, Com_Exam e ExamCompl.

Depois armazenamos na variável ind a query às tabelas infpes e examcompl sob as condições de que os atributos resultado e nome, tinha de ser iguais respetivamente a "Positivo" e "COVID-19", e a ligação todas as chaves primárias a todas as chaves estrangeiras.

Depois fizemos print de todos os resultados desta query no formato "nif: nif data do exame: data do exame".

iv) Neste exercício importamos do neo4j a GraphDatabase e do sys o argv. Criamos o driver para poder fazer a ligação à base de dados, que é guardado em driver.

Criamos uma sessão que é armazenada em session, guardamos na variável arg1 o nif passado em argv[1].

Guardamos em res o resultado da query que procura pelas ligações da pessoa do nif indicado em arg1, e depois retorna o nome dessa pessoa, o nome da localidade ou de outra pessoa, como o nif dessa pessoa.

Depois faz print dessas relações em dois formatos "nif tipo de relação nif" caso sejam as duas entidades pessoas, verifica-se isso com "a[2] is not None" pois a localidade para o atributo nif retorna null uma vez que não tem esse atributo. Caso as duas entidades sejam pessoa e localidade(e não duas pessoas), respetivamente, então o formato será este "nif tipo de relação nome".

Desenvolvimento

Exercícios 2.2.2 (Continuação)

v) Importamos e fazemos as mesmas nomeações todas presentes nos exercícios i, ii, iii.

Criamos todas as classes presentes no exercício iii. Depois de estabelecermos uma sessão passo já explicado nos exercícios antes do iv, fizemos os imports presentes na iv e tudo de igual até mesmo a query presente em res, no entanto não fazemos print dos resultados. No lugar guardamos na variável loc uma array vazia para guardar depois todos o locais relacionados com o individuo pesquisado na query presente em res.

Para isso dentro de um for usamos a condição “a[2].type != ‘amigo_de’ para isolar os locais das pessoas.

Depois armazenamos em rd uma array vazia para guardar todas as pessoas relacionada com o locais em loc. Para isso fazemos append em rd da query que pesquisa as relações entre os locais em loc e as pessoas.

Fazemos duplo for em que o 2º está sob a condição “str(i[1]) != arg1” para que a pessoa a que toda esta pesquisa e relacionada seja ignorada. Guardamos em res a query à tabela infpes para encontrar todas as pessoas com o nif que esteja dentro de rd. E fazemos print no formato “localidade nif nome data de nascimento email telefone código de postal localidade”.

Conclusão

Com a realização deste projeto ficou evidente que o sistema a utilizar numa determinada base de dados depende sempre do tipo de dados e qual a sua finalidade.

Apesar de em grande parte dos problemas as bases de dados relacionais conseguirem solucionar, quando um problema é mais especifico como o exemplo deste projeto – a rede de contactos – as bases de dados não relacionais (Not Only SQL) surgem como solução.

Assim, convergimos no sentido de criar um sistema híbrido de bases de dados relacionais e não relacionais, utilizando script python para criar, remover, alterar e analisar dos dados neles contidos.