Data Engineering Test - KZAS

1. Objetivos

Temos duas fontes de dados:

- *Estate ads* (Anúncios imobiliários): Não estruturado, em arquivos JSON
- *Buildings* (Edifícios)*:* Estruturado, em um arquivo CSV

Devemos identificar para cada anúncio qual edifício ele pertence, e inserir seus dados enriquecidos com os dados de seu respectivo edifício em um banco de dados estruturado PostgreSQL.

2. Observações feitas sobre os dados

Os dados estão colocados assim:

- *Estate ads* (10000 arquivos):
  - *idno*
  - *property_type*
  - *city_name*
  - *state*
  - *street*
  - *sale_price*
  - *neighborhood*
  - *built_area_min*
  - *bedrooms_min*
  - *bathrooms_min*
  - *parking_space_min*
  - *lat* (latitude)
  - *lon* (longitude)
  - *street_number*

- *Buildings* (629102 linhas):
  - *id*
  - *address* (street)
  - *address_number*
  - *neighborhood*
  - *city*
  - *state*
  - *cep*
  - latitude
  - longitude

Com isso, já podemos ver alguns dados que estão presentes em ambas as bases que podem ser utilizadas para realizarmos a busca por localização. Vamos utilizar os dados dos anúncios para encontrar os edifícios utilizando esses dados nessa ordem:

state -> city -> neighborhood -> street (address) -> street_number (address_number)

Realizando essa busca, já conseguimos relacionar grande parte dos anúncios, porém existem algumas divergências e más formatações nas bases que fazem com que

alguns dos anúncios não possam ser encontrados nas bases. Existem divergências do modo de escrita das ruas e bairros nas bases, como por exemplo, existe um arquivo de anúncio que estava escrito “Rua VERGUEIRO” e na base de edifícios “Rua Vergueiro”, outro que continha “Rua Pensilvania” e na base de edifícios “Rua Pensilvânia”.

Uma das soluções possíveis para esse problema seria normalizar esses textos, mudando tudo para minúsculo, e retirar as acentuações. Porém como mostrado na imagem abaixo:

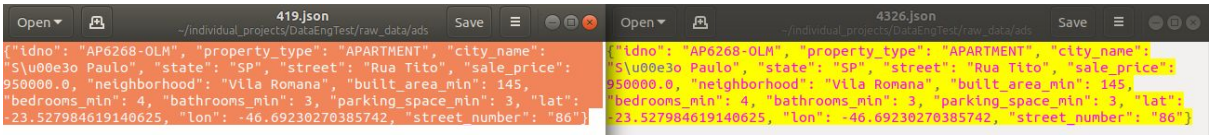
	id	address	address_number	neighborhood	city	state	cep	latitude	longitude
157397	2396488	Rodovia Mario Covas	67	Rodovia Mario covas	Mairinque	SP	18120000	-23.479609	-47.203674
222116	2396740	Rodovia Mario Covas	67	rodovia Mario covas	Mairinque	SP	18120000	-23.479609	-47.203674
229638	2378233	Rodovia Mario Covas	67	rodovia Mario Covas	Mairinque	SP	18120000	-23.479609	-47.203674
230467	2410633	Rodovia Mario Covas	67	rodovia mario covas	Mairinque	SP	18120000	-23.479609	-47.203674
232074	2379110	Rodovia Mario Covas	67	Rodovia Mario Covas	Mairinque	SP	18120000	-23.479609	-47.203674
325896	2397089	Rodovia Mario Covas	67	Rodovia mario covas	Mairinque	SP	18120000	-23.479609	-47.203674

Existem registros com os mesmos nomes de endereço que só se diferenciam por essas pequenas variações, e como eles possuem IDs diferentes, não se tem propriedade o bastante para juntar todos esses registros em um só. Então essa abordagem não seria muito útil por causa disso, além de não ser uma solução geral.

Também existe uma diferença na precisão das latitudes e longitudes das bases, em que mesmo que todos os dados de localização do anúncio batam com os do edifício, os dados das duas coordenadas dificilmente são iguais, o que impossibilita uma busca direta por elas.

Outra divergência está na variação nos nomes dos bairros, como por exemplo: “Brooklin” e “Cidade Monções”. Para isso, foi elaborada uma solução de aproximação, que para os que não foram localizados na primeira busca, iremos realizar uma segunda busca, primeiramente filtrando pelo nome do estado, cidade, rua e número, e após isso, pegando todos os resultados dessa busca, com as coordenadas de latitude e longitude vemos qual dos edifícios retornados está mais próximo do anúncio, e caso ele esteja a menos de 0.5km (já que os dados das coordenadas não são tão precisos, mesmo que seja de fato o mesmo local, existirá uma diferença, mesmo que mínima) podemos dizer que os registros são do mesmo bairro, e assim associar o anúncio ao edifício. Com essa abordagem, conseguimos ainda enriquecer alguns anúncios que não eram encontrados em seus respectivos edifícios, mas ainda assim uma parte deles ficou sem ser relacionado a um edifício devido às divergências apontadas anteriormente.

Nota-se também que se tem anúncios repetidos na base, como por exemplo nos arquivos 419.json e 4326.json:



Sendo assim, de todos os arquivos dos anúncios, temos 6448 únicos.

Por fim, um fato interessante é que todos os anúncios fornecidos são da cidade de São Paulo. Essa informação poderia ter sido utilizada para filtrarmos a base de dados de edifícios para conter apenas registros da cidade de São Paulo também, isso otimizaria a busca e a deixaria mais rápida, porém isso não foi feito pois foi buscado uma solução geral para o problema, sendo que caso no futuro outros anúncios fossem colocados, não fosse necessário alterar o programa.

3. Tabela do banco de dados

Optamos pela criação de apenas uma tabela (“ads_buildings”) com todos os anúncios, sendo os que foram encontrados estarão enriquecidos, e os que não foram encontrados estarão com os dados que seriam do edifício nulos.

Assim está estruturada a tabela:

Coluna	Tipo	Fonte
ads_id (PRIMARY KEY)	VARCHAR(16)	Estate ads
building_id	INT	Buildings
property_type	VARCHAR(28)[]	Estate ads
state	VARCHAR(2)	Estate ads
city	VARCHAR(64)	Estate ads
neighborhood	VARCHAR(128)	Estate ads
street	VARCHAR(128)	Estate ads
street_number	VARCHAR(64)	Estate ads
bedrooms_min	INT	Estate ads
bathrooms_min	INT	Estate ads
parking_space_min	INT	Estate ads
cep	VARCHAR(9)	Buildings
sale_price	DECIMAL(12, 2)	Estate ads
latitude	DECIMAL(10, 8)	Estate ads/Buildings*
longitude	DECIMAL(11, 8)	Estate ads/Buildings*
accurate**	BOOLEAN	-

* Dados em que o *Estate ads* pode conter None e assim ser preenchido com os dados de *Buildings,* caso eles também não sejam vazios

** Como dito antes, realizamos dois tipos de buscas, sendo a primeira exata, e todos os anúncios que passaram nessa primeira busca terão a *flag accurate* como verdadeira, e caso o anúncio não tenha sido encontrado nos edifícios ou encontrado na segunda busca, será colocado como Falso. Isso foi feito para que caso no futuro esses dados sejam utilizados em alguma análise, seja possível filtrar quais dados estão precisos dos que foram aproximados ou não encontrados.

4. Construção do script

O script foi feito na linguagem de programação Python, e com base em uma classe criada com todas as funcionalidades necessárias para inserir os dados no banco de dados. A classe possui as seguintes funções:

```
class Ads_buildings_DB(builtins.object)
 |  Ads_buildings_DB(host, port, user, password, db, buildings_path)
 |
 |  Methods defined here:
 |
 |  __init__(self, host, port, user, password, db, buildings_path)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  create_table(self)
 |      Create `ads_buildings` table in the DB that self.conn is connected.
 |
 |  drop_table(self)
 |      Dropping the `ads_buildings` table, if it exists.
 |
 |  insert_data(self, ads_path)
 |      Inserting data into DB.
 |
 |  table_exists(self)
 |      Checking if the `ads_buildings` table exists in the DB that self.conn is connected.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

a. Requisitos

Para execução do script foi fornecido um arquivo Pipfile para ser utilizado para criar um ambiente pelo *pipenv* já com todas as bibliotecas e pré-requisitos necessários para a execução do script. Mas caso não seja possível utilizar o *pipenv*, o script foi executado com Python 3.7.3, e com as seguintes bibliotecas:

- pandas: 1.2.0
- numpy: 1.19.5
- psycopg2: 2.8.6
- geopy: 2.1.0
- tqdm: 4.56.0
- argparse: 1.1

b. Funcionamento

Logo quando a classe é instanciada se estabelece uma conexão com o banco de dados, e é checado se a tabela “ads_building” existe, e caso não, ela já é

criada automaticamente. Também já é lido e formatado o arquivo com os dados dos edifícios.

Já instanciado, pode-se chamar a função “insert_data” passando o caminho da pasta que contém os arquivos dos anúncios. Essa função já realiza todo o processo para a inserção de dados na tabela, realizando as duas possíveis buscas já explicadas anteriormente, e preenchendo todos os dados que foram possíveis de serem encontrados para cada anúncio, e depois disso injetando eles no banco.

c. Execução

Para a execução, primeiro é necessário descompactar os arquivos, para facilitar, pode-se apenas executar o bash script “get_data.sh” que já descompacta e deixa na pasta padrão do script:

```
bash get_data.sh
```

Já com os dados, caso se tenha o *pipenv* basta realizar o comando:

```
pipenv run python ads_buildings_pgsql.py
```

Caso não se tenha o pipenv, comando é `python ads_buildings_pgsql.py`

O script já está configurado com os dados para conexão do banco e caminho para pasta com os dados (caso o get_data.sh tenha sido executado, deve ser a pasta “data/”), porém caso seja necessário alterar alguma dessas informações, isso pode ser feito através dos argumentos passados na execução do script:

```
optional arguments:
  -h, --help            show this help message and exit
  -host HOST            DB host
  -p PORT, --port PORT  DB port
  -u USER, --user USER  DB user
  -pass PASSWORD, --password PASSWORD
                        DB password
  -db DB, --db_name DB  DB name
  -data DATA_PATH, --data_path DATA_PATH
                        Folder containing the folder "ads/" with json files of
                        the ads,and the file "buildings.csv" with the
                        information of the buildings
```

5. Melhorias

Caso houvesse mais tempo, uma análise mais detalhada sobre a base de dados poderia ser feita, entendendo melhor a inconsistência dos dados e tentando encontrar maneiras para solucioná-las ou talvez também consertá-las.

Outra coisa também poderia ser a utilização de APIs de localização que poderiam ser utilizadas para localização com as coordenadas de latitude e longitude dos anúncios que não foram encontrados na base dos edifícios.