UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA

TRABALHO 01
CRIAÇÃO BASE DE DADOS CORREIOS

02/10/2020

VICTOR GABRIEL CASTÃO DA CRUZ – 11911ECP004

UBERLÂNDIA
2020

1 – RESUMO

O trabalho proposto consistia na criação de uma base de dados dos correios, e para tal, os dados a serem utilizados estavam contidos em um arquivo csv (fornecido pelo professor), cuja inserção deveria ocorrer através de um programa (no caso deste trabalho, desenvolvido na linguagem de programação Python). Após, os dados serviriam de base para a criação de tabelas, chaves (primárias e estrangeiras) e views, exemplificando possíveis aplicações para esses dados.

Os procedimentos para a realização deste trabalho serão apresentados a seguir.

2 – DESENVOLVIMENTO

As etapas para a construção da base de dados foram divididas em funções, que seriam executadas separadamente, evitando com que possíveis erros sejam difíceis de identificar. O ambiente de desenvolvimento escolhido foi o PyCharm Community 2019.3, e as bibliotecas utilizadas serão apresentadas a seguir.

2.1 – Módulos e Bibliotecas

Com base na solução proposta para o trabalho, o script necessitaria apenas de dois módulos: CSV (para gerenciar planilhas) e PSYCOPG2 (para comunicação com o banco de dados), cuja importação se deu pelos seguintes comandos:

```
import csv, psycopg2
from psycopg2.extensions import ISOLATION_LEVEL_AUTOCOMMIT
```

2.2 – Criação da Database

A base de dados a ser utilizada poderia ser criada via comandos no próprio editor sql do gerenciador, entretanto, tal processo foi realizado via função definida no script, resultando em uma base de nome “base_correios”:

```
def criar_database():
        conexao = psycopg2.connect(dbname='base_2020_02', user='postgres', password='banco', host='localhost', port='5432')
        conexao.set_isolation_level(ISOLATION_LEVEL_AUTOCOMMIT)
        cur = conexao.cursor()
```

```
        cur.execute("create database base_correios")
        conexao.commit()
        cur.close()
        conexao.close()
        return 1

passo1 = criar_database()
```

2.3 - Criação do Schema

Após a criação da base de dados, criou-se um schema de nome “dados_ceps”:

```
def criar_schema():
        conexao = psycopg2.connect(dbname='base_correios', user='postgres', password='banco', host='localhost', port='5432')
        cur = conexao.cursor()
        cur.execute("create schema dados_ceps")
        conexao.commit()
        cur.close()
        conexao.close()
        return 1

passo2 = criar_schema()
```

2.4 – Criação das Tabelas

Para manuseio da base de dados, criaram-se quatro tabelas para auxílio neste processo. A primeira delas, por sua vez, consistiria nos dados presentes no arquivo csv, que se referiam aos CEPs de diversas cidades do Brasil. Com isso, seus devidos campos, assim como a declaração de chaves primárias e/ou estrangeiras, foram criados:

```
def criar_tabela_1():
        conexao = psycopg2.connect(dbname='base_correios', user='postgres', password='banco', host='localhost', port='5432')
        cur = conexao.cursor()
        comando = " ".join([
                'create table dados_ceps.informacoes_gerais(',
```

```
                'id_cep varchar(10) constraint nn_id_cep not null,',
                'uf varchar(2) constraint nn_uf not null,',
                'cidade varchar(100) constraint nn_cidade not null,',
                'bairro varchar(100),',
                'endereco varchar(100) constraint nn_endereco not null,',
                'constraint pk_id_cep primary key (id_cep)',
                ');'
            ])
            cur.execute(comando)
            conexao.commit()
            cur.close()
            conexao.close()
            return 1

    passo3 = criar_tabela_1()
```

Após, para demonstrar uma aplicação para esses dados, elaborou-se uma segunda tabela, desta vez contendo informações acerca dos clientes de determinado local, cujo endereço seria referenciado pelo número do CEP contido na tabela anterior:

```
    def criar_tabela_2():
        conexao = psycopg2.connect(dbname='base_correios', user='postgres', password='banco', host='localhost', port='5432')
        cur = conexao.cursor()
        comando = " ".join([
            'create table dados_ceps.cadastro_cliente(',
            'cpf varchar(11) constraint nn_cpf not null,',
            'nome varchar(20) constraint nn_nome not null,',
            'sobrenome varchar(50) constraint nn_sobrenome not null,',
            'cep varchar(10) constraint nn_cep not null,',
            'numero varchar(5),',
            'complemento varchar(10),',
            'telefone varchar(12),',
            'constraint pk_cpf primary key (cpf),',
            'constraint fk_cep foreign key (cep) references
            dados_ceps.informacoes_gerais (id_cep)',
```

```
                ');'
        ])
        cur.execute(comando)
        conexao.commit()
        cur.close()
        conexao.close()
        return 1

passo4 = criar_tabela_2()
```

Semelhante a tabela acima, foi elaborada, agora, uma tabela contendo informações acerca de empresas, também referenciadas pelo CEP:

```
def criar_tabela_3():
        conexao = psycopg2.connect(dbname='base_correios', user='postgres', password='banco', host='localhost', port='5432')
        cur = conexao.cursor()
        comando = " ".join([
                'create table dados_ceps.cadastro_empresa(',
                'cnpj varchar(14) constraint nn_cnpj not null,',
                'nome varchar(100) constraint nn_nome not null,',
                'cep varchar(10) constraint nn_cep not null,',
                'numero varchar(5),',
                'complemento varchar(10),',
                'telefone varchar(12),',
                'constraint pk_cnpj primary key (cnpj),',
                'constraint fk_cep foreign key (cep) references dados_ceps.informacoes_gerais (id_cep)',
                ');'
        ])
        cur.execute(comando)
        conexao.commit()
        cur.close()
        conexao.close()
        return 1
```

```
passo5 = criar_tabela_3()
```

Por fim, a quarta tabela feita consiste nos dados referentes a uma compra, referenciada pelo CPF do destinatário e CNPJ do remetente. Com isso, indiretamente é possível descobrir o endereço de origem e destino do produto vendido:

```
def criar_tabela_4():
    conexao = psycopg2.connect(dbname='base_correios', user='postgres', password='banco', host='localhost', port='5432')
    cur = conexao.cursor()
    comando = " ".join([
        'create table dados_ceps.cadastro_encomendas(',
        'id_encomendas integer constraint nn_id_encomendas not null,',
        'descricao varchar(100) constraint nn_descricao not null,',
        'valor numeric(8,2) constraint nn_valor not null,',
        'cnpj_entregador varchar(14) constraint nn_cnpj_entregador not null,',
        'cpf_destinatario varchar(11) constraint nn_cpf_destinatario not null,',
        'constraint pk_id_encomendas primary key (id_encomendas),',
        'constraint fk_cnpj_entregador foreign key (cnpj_entregador) references dados_ceps.cadastro_empresa (cnpj),',
        'constraint fk_cpf_destinatario foreign key (cpf_destinatario) references dados_ceps.cadastro_cliente (cpf)',
        ');'
    ])
    cur.execute(comando)
    conexao.commit()
    cur.close()
    conexao.close()
    return 1

passo6 = criar_tabela_4()
```

2.5 – Inserção de Dados

Com as devidas tabelas já criadas, restava, então, inserir os devidos dados em cada uma. Inicialmente, pode-se pensar em inserir cada dado, um a um, até que a tabela esteja completa. Entretanto, para tarefas como a inserção dos CEPs, o volume de dados torna essa prática completamente ineficiente. Com base nisso, o script desenvolvido, juntamente com as bibliotecas importadas, era capaz de iterar sobre as linhas da planilha utilizada, e com isso, transformar cada uma em uma estrutura denominada dicionário (para tal, a planilha teve uma linha inserida no começo, onde a primeira célula de cada coluna passaria a ter o nome do campo). Assim, seus dados poderiam ser extraídos e adicionados aos devidos campos da tabela.

Entretanto, alguns campos da planilha possuíam em seus nomes o caracter de aspas simples, o que ocasionava erros no momento de inserir os devidos dados. Pensando nisso, uma função auxiliar foi desenvolvida para formatar dados que apresentassem essa característica para um formato no qual a inserção seria bem sucedida, permitindo, assim, com que todos os dados pudessem estar contidos na tabela.

```
def correcao_string(texto):
	resp = ""
	for c in texto:
		resp += c
		if c == "'":
			resp += "'"
	return resp


def inserir_dados_cep():
	conexao = psycopg2.connect(dbname='base_correios', user='postgres', password='banco', host='localhost', port='5432')
	cur = conexao.cursor()
	with open('utfcepos.csv', newline='') as arquivo:
		leitura = csv.DictReader(arquivo)
		for tupla in leitura:
			t_cep = tupla['CEP']
			t_uf = tupla['UF']
			t_cidade = tupla['CIDADE']
			if "'" in t_cidade:
```

```
                t_cidade = correcao_string(t_cidade)
            t_bairro = tupla['BAIRRO']
            if "'" in t_bairro:
                t_bairro = correcao_string(t_bairro)
            t_endereco = tupla['ENDERECO']
            if "'" in t_endereco:
                t_endereco = correcao_string(t_endereco)
            comando = "" + \
                "insert into dados_ceps.informacoes_gerais (id_cep, uf,
cidade, bairro, endereco)" + \
                " values ('" + \
                str(t_cep) + "', '" + \
                str(t_uf) + "', '" + \
                str(t_cidade) + "', '" + \
                str(t_bairro) + "', '" + \
                str(t_endereco) + "'" + \
            ');'
            cur.execute(comando)
            conexao.commit()
            cur.close()
            conexao.close()
            return 1

    passo7 = inserir_dados_cep()
```

Os dados das outras tabelas também foram inseridos. Porém, como a quantidade de dados destas era pequena (utilizado somente para teste), sua inserção se deu de maneira mais simples:

```
    def preencher_tabelas():
        conexao = psycopg2.connect(dbname='base_correios', user='postgres',
password='banco', host='localhost', port='5432')
        cur = conexao.cursor()
        comando = "" + \
```

```
                    "insert into dados_ceps.cadastro_cliente (cpf, nome, sobrenome, cep,
numero, telefone) " + \
                    "values
('44212321398','Valter','Barreto','14177230','529','16988345567'); " + \
                    "insert into dados_ceps.cadastro_cliente (cpf, nome, sobrenome, cep,
numero, telefone) " + \
                    "values ('12345678910','Pedro','da
Silva','11010200','59','11988763232'); " + \
                    "insert into dados_ceps.cadastro_cliente (cpf, nome, sobrenome, cep,
numero, telefone) " + \
                    "values
('54334512312','Sérgio','Pacheco','13455806','300','14993074287'); " + \
                    "insert into dados_ceps.cadastro_cliente (cpf, nome, sobrenome, cep,
numero, telefone) " + \
                    "values
('23412985673','Carlos','Sanchez','33120120','08','34992013243'); " + \
                    "insert into dados_ceps.cadastro_cliente (cpf, nome, sobrenome, cep,
numero, telefone) " + \
                    "values ('76512324867','Roberto','de
Aquino','35701259','1231','37988520345'); " + \
                    "insert into dados_ceps.cadastro_empresa (cnpj, nome, cep, numero,
telefone) " + \
                    "values ('11222333000159','Colombianas
S.A','35701263','402','37998520311'); " + \
                    "insert into dados_ceps.cadastro_empresa (cnpj, nome, cep, numero,
telefone) " + \
                    "values ('22333444000121','Casas Ceará
LTDA','60750280','45','34998450311'); " + \
                    "insert into dados_ceps.cadastro_empresa (cnpj, nome, cep, numero,
telefone) " + \
                    "values ('55444213000197','Minas Shop
LTDA','38073065','978','3739524359'); " + \
                    "insert into dados_ceps.cadastro_empresa (cnpj, nome, cep, numero,
telefone) " + \
                    "values ('99222111000134','São Luís
Modas','65048730','48','4139756545'); " + \
```

```
                "insert into dados_ceps.cadastro_empresa (cnpj, nome, cep, numero,
telefone) " + \
                "values ('44777888000121','Pedra Bonita
Presentes','68740100','S/N','6534569877'); " + \
                "insert into dados_ceps.cadastro_encomendas (id_encomendas,
descricao, valor, cnpj_entregador, cpf_destinatario) " + \
                "values (1,'Notebook i3 4gb
RAM',1950.00,'55444213000197','54334512312'); " + \
                "insert into dados_ceps.cadastro_encomendas (id_encomendas,
descricao, valor, cnpj_entregador, cpf_destinatario) " + \
                "values (2,'Tênis Nike Dart
XII',250.00,'99222111000134','23412985673'); " + \
                "insert into dados_ceps.cadastro_encomendas (id_encomendas,
descricao, valor, cnpj_entregador, cpf_destinatario) " + \
                "values (3,'Relógio Digital
Casio',234.90,'44777888000121','76512324867'); " + \
                "insert into dados_ceps.cadastro_encomendas (id_encomendas,
descricao, valor, cnpj_entregador, cpf_destinatario) " + \
                "values (4,'Tv 32
Polegadas',1799.90,'22333444000121','44212321398'); " + \
                "insert into dados_ceps.cadastro_encomendas (id_encomendas,
descricao, valor, cnpj_entregador, cpf_destinatario) " + \
                "values (5,'Geladeira
Electrolux',2399.98,'22333444000121','54334512312');"
        cur.execute(comando)
        conexao.commit()
        cur.close()
        conexao.close()
        return 1

passo8 = preencher_tabelas()
```

2.6 – Criação de Views

Com todos os dados devidamente inseridos, era preciso, então, criar algumas views para melhor visualização dos dados, bem como obter informações específicas destes.

A primeira dessas informações era saber os endereços presentes no estado de São Paulo:

```
def criar_views_1():
    conexao = psycopg2.connect(dbname='base_correios', user='postgres', password='banco', host='localhost', port='5432')
    cur = conexao.cursor()
    comando = "" + \
        "create or replace view dados_ceps.ruas_sp as " + \
        "select endereco from dados_ceps.informacoes_gerais " + \
        "where uf = 'SP';"
    cur.execute(comando)
    conexao.commit()
    cur.close()
    conexao.close()
    return 1

passo9 = criar_views_1()
```

De forma análoga, criou-se também uma view que exibisse as cidades do estado de Minas Gerais:

```
def criar_views_2():
    conexao = psycopg2.connect(dbname='base_correios', user='postgres', password='banco', host='localhost', port='5432')
    cur = conexao.cursor()
    comando = "" + \
        "create or replace view dados_ceps.cidades_mg as " + \
        "select cidade from dados_ceps.informacoes_gerais " + \
        "where uf = 'MG' " + \
        "group by (cidade);"
    cur.execute(comando)
    conexao.commit()
    cur.close()
    conexao.close()
```

```
        return 1

passo10 = criar_views_2()
```

Como exigência, a terceira view deveria conter um agrupamento (utilizando “group by”). Dessa forma, elaborou-se uma que apresentava o número de cidades distintas existentes em cada estado (de acordo com a planilha):

```
def criar_views_3():
        conexao = psycopg2.connect(dbname='base_correios', user='postgres', password='banco', host='localhost', port='5432')
        cur = conexao.cursor()
        comando = "" + \
                "create or replace view dados_ceps.contador_cidades as " + \
                "select count(distinct(cidade)), uf from dados_ceps.informacoes_gerais " + \
                "group by (uf);"
        cur.execute(comando)
        conexao.commit()
        cur.close()
        conexao.close()
        return 1

passo11 = criar_views_3()
```

Por fim, a última view apresentaria os dados da tabela de encomendas (especificamente seu id, CNPJ da empresa e CPF do cliente):

```
def criar_views_4():
        conexao = psycopg2.connect(dbname='base_correios', user='postgres', password='banco', host='localhost', port='5432')
        cur = conexao.cursor()
        comando = "" + \
                "create or replace view dados_ceps.encomendas as " + \
```

```
                "select id_encomendas, cnpj_entregador, cpf_destinatario from dados_ceps.cadastro_encomendas;"
        cur.execute(comando)
        conexao.commit()
        cur.close()
        conexao.close()
        return 1

passo12 = criar_views_4()
```

Vale destacar também que, considerando o intersse em exibir a view no próprio terminal, uma função que realizasse tal procedimento foi criada. Diferente das outras, essa recebe um parâmetro, que corresponde ao nome da view que deverá ser exibida:

```
def view(nome):
        conexao = psycopg2.connect(dbname='base_correios', user='postgres', password='banco', host='localhost', port='5432')
        cur = conexao.cursor()
        comando = "select * from dados_ceps." + str(nome) + ";"
        cur.execute(comando)
        tupla = cur.fetchall()
        for campo in tupla:
                for info in campo:
                        print(info, end=' ')
                print()
        conexao.commit()
        cur.close()
        conexao.close()
        return 1

passo13 = view('ruas_sp')
passo14 = view('cidades_mg')
passo15 = view('contador_cidades')
passo16 = view('encomendas')
```

Executando a função “view(‘contador_cidades’)”, por exemplo, obtém-se o seguinte resultado no terminal:

1 AC
3 AL
3 AM
1 AP
15 BA
7 CE
1 DF
10 ES
13 GO
4 MA
34 MG
5 MS
4 MT
12 PA
6 PB
18 PE
2 PI
30 PR
27 RJ
3 RN
4 RO
1 RR
23 RS
14 SC
1 SE
102 SP
3 TO

2.7 – Conclusão

A partir das funções definidas anteriormente, verifica-se que, por meio delas, foi possível a construção de uma base de dados com um volume de dados significativo, bem como a apresentação desses dados nas mais diversas formas.

Tais apresentações poderiam, ainda, ser expandidas com a criação de novas views de forma análoga às anteriores. Além disso, a função criada para manusear o arquivo csv, com algumas adaptações, poderia, também, inserir uma enorme quantidade de dados nas outras tabelas, criando um banco de dados maior, mais abrangente e feito de forma eficiente, demonstrando o quanto scripts como esses são de grande valia para essa área.