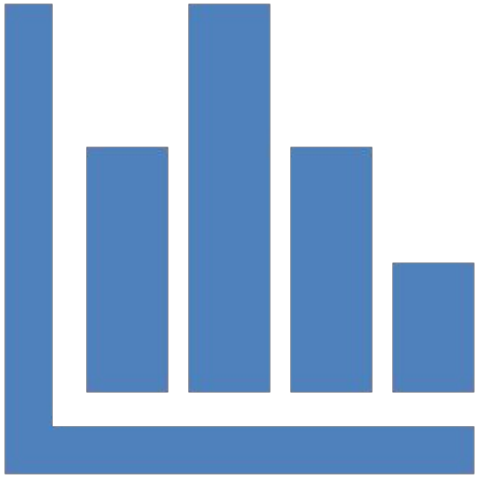


LGPD, Banco de Dados e Segurança da Informação

Aula prática e estudo de caso

# Anonimização de Dados em Banco de Dados com Python

# Objetivos da Aula

Entender LGPD e conceitos de privacidade e anonimização.

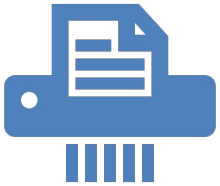
Conhecer técnicas: mascaramento, generalização e criptografia.

Implementar rotinas em Python integradas com SQL Server (pyODBC).

Resolver o estudo de caso prático e gerar relatório anonimizado.

# O que é a LGPD? (resumo)

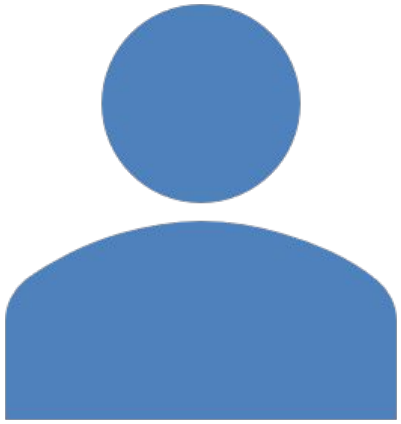
Lei Geral de Proteção de Dados — em vigor desde Sep/2020 no Brasil.



Protege direitos dos titulares: acesso, correção, exclusão e portabilidade.

Princípios principais: consentimento, finalidade, necessidade e segurança.

# Dados Pessoais vs Dados Sensíveis



- Dados pessoais: identificam direta ou indiretamente (nome, CPF, e-mail).
- Dados sensíveis: origem racial, convicções religiosas, saúde, biometria.
- Tratamento de dados sensíveis exige cuidados e bases legais reforçadas.

# Anonimização x Pseudoanonimização

| Aluno | Média | Frequência |
|---|---|---|
| André Cassiano | 8,0 | 75% |
| Paulo Henrique | 5,0 | 80% |
| Ana Cristinna | 10,0 | 50% |

| Aluno | Média | Frequência |
|---|---|---|
| 2562 | 8,0 | 75% |
| 5533 | 5,0 | 80% |
| 439 | 10,0 | 50% |

- Anonimização: torna impossível reatribuir os dados a um indivíduo.
- Pseudoanonimização: reversível com informação adicional (chave separada).
- LGPD trata dados pseudoanonimizados como dados pessoais (ainda sujeitos).

# Técnica: Supressão / Mascaramento

| CPF | Plano | e-mail |
|---|---|---|
| 094******0-90 | Master | p***o@u***.com.br |
| 094******0-91 | Master | s***za@a**.com.br |
| 094******0-92 | Fidelidade | u***o@***.com |

- Substitui total ou parcialmente os dados por caracteres fixos (e.g., *****).
- Vantagem: simples e eficaz para proteger identificadores diretos.
- Desvantagem: perda de detalhe analítico (não reutilizável para certas análises).

# Técnica: Generalização

| ID | Idade | Endereço |
|---|---|---|
| 2562 | 20 - 30 | Rua Padre Vieira |
| 5533 | 30 - 40 | Rua Marechal Unitro |
| 5439 | 20 - 30 | Alameda 32 |

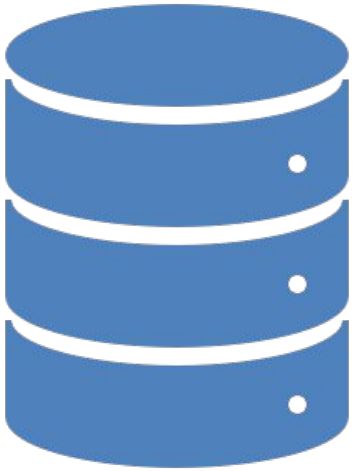
- Agrupa valores em categorias (e.g., idade 23 → 20<Idade<=30).
- Preserva utilidade para análises estatísticas, reduz risco de identificação.
- Requer cuidado com categorias muito finas que permitam reidentificação.

# Técnica: Criptografia


- Transforma dados em forma cifrada — exige chave para descriptografar.
- Simétrica: a chave secreta pode ser descriptografada com mais facilidade, pois utiliza mesma chave para cifrar/decifrar, e qualquer usuário com privilégio de administrador pode utilizar.
- Assimétrica: mais lenta que a simétrica, mas utiliza um par de chaves pública/privada , sendo que a privada fica arquivada em repositório diferente (mais difícil de descriptografar).

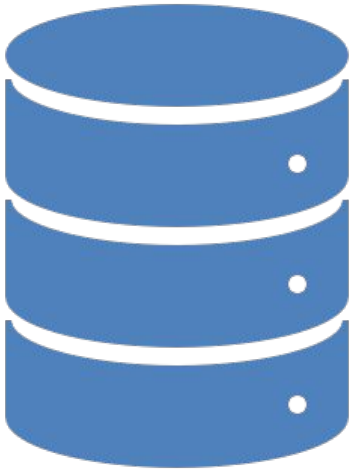
# Python como ferramenta

- Linguagem legível, rica em bibliotecas (pandas, hashlib, pyodbc, sqlalchemy).
- Útil para preparar, transformar e anonimizar bases de dados.
- Permite integração direta com bancos (pyodbc) e exportação de relatórios.



# SQL Server + Python

- A Linguagem de Consulta Estruturada, conhecida como SQL (Structured Query Language), é uma linguagem de programação utilizada para gerenciar, consultar e recuperar dados em bancos de dados relacionais. É a linguagem padrão dos sistemas de gerenciamento de banco de dados relacional (RDBMS), tais como PostgreSQL, SQL Server, MySQL e Oracle Database.
- A integração do SQL Server com o Python é possível por meio de bibliotecas e ferramentas que fornecem conectividade e capacidade de consulta ao banco de dados. A seguir apresentamos algumas das opções disponíveis para realizar essa integração.



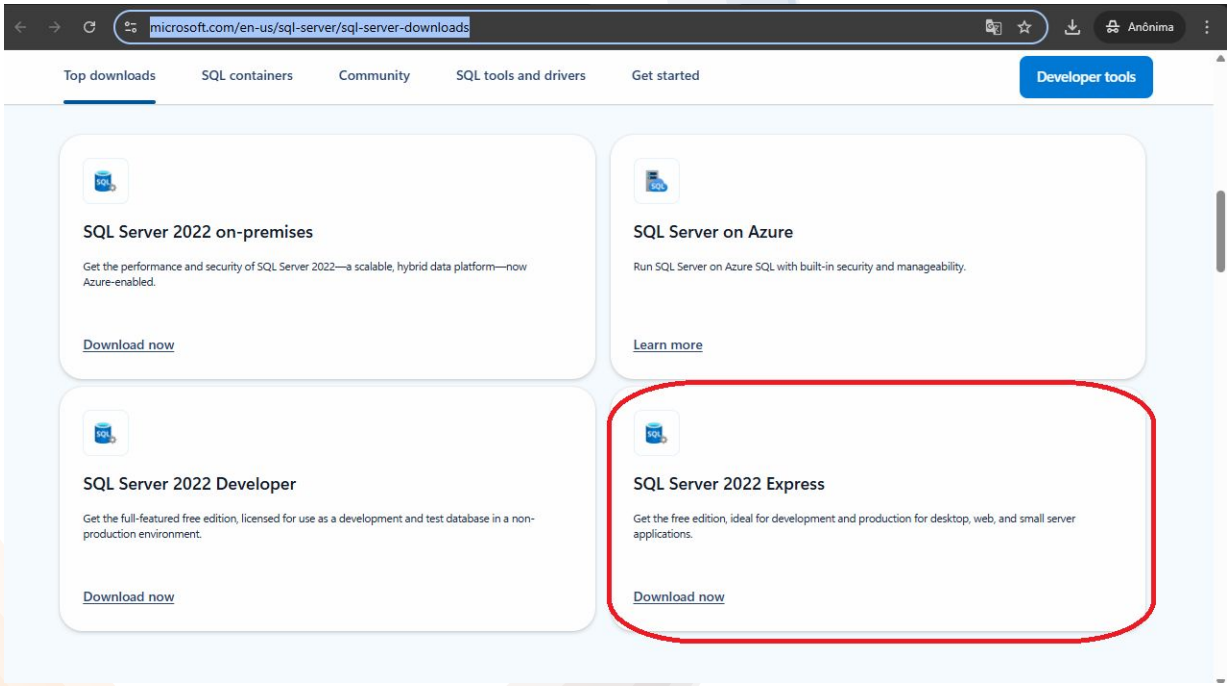
# SQL Server + Python

- **pyODBC:** módulo Python que permite a conexão com bancos de dados por meio do ODBC (Open Database Connectivity). Com ele é possível estabelecer uma conexão com o SQL Server, executar consultas SQL e manipular os resultados no Python.
- **pymssql:** oferece conectividade direta com o SQL Server por meio de uma interface simples por meio da qual é possível se conectar a um banco de dados e executar consultas SQL, além de oferecer suporte para transações e manipulação Python. de dados.
- **SQLAlchemy:** biblioteca de mapeamento objeto-relacional (ORM) em Python que permite interagir com diversos bancos de dados, incluindo o SQL Server. Fornece uma camada de abstração que facilita a comunicação com o banco de dados, permitindo trabalhar com objetos Python ao invés de escrever diretamente consultas SQL.

# SQL Server + Python (Prática)

Utilizaremos a ferramenta SQL Server Management Studio para criar o banco de dados, mas, antes é necessário realizar a instalação do SQL Server. Para isso, acesse a página de downloads da Microsoft e localize o link (https://www.microsoft.com/en-us/sql-server/sql-server-downloads) para baixá-lo, conforme a figura a seguir:

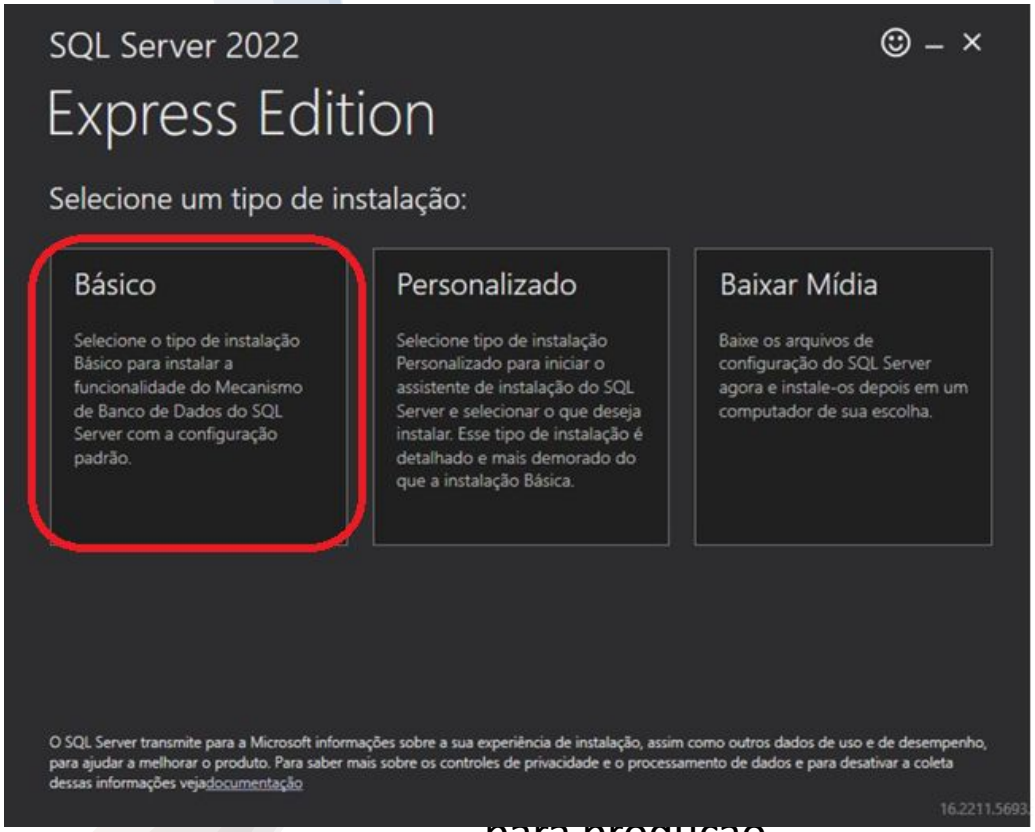
.




* O Express é uma versão gratuita que pode ser usada tanto para desenvolvimento quanto para produção.

# SQL Server + Python (Prática)

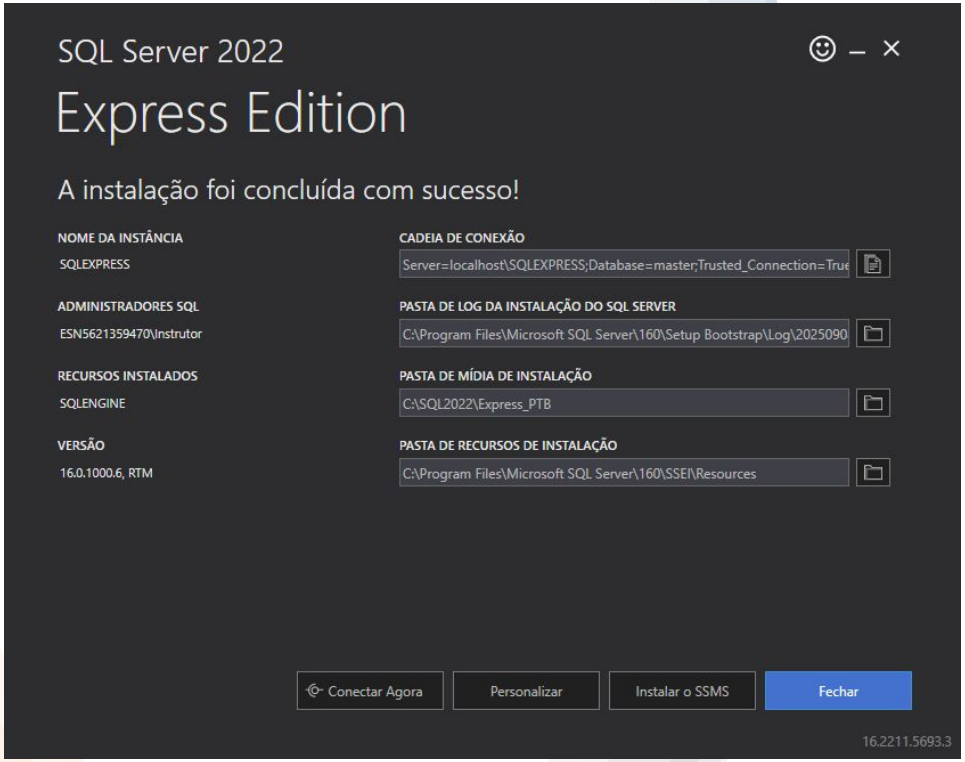
Após finalizado o download, escolha a opção ‘Básico’, aceite os Termos e clique em ‘Instalar’.




para produção.

# SQL Server + Python (Prática)

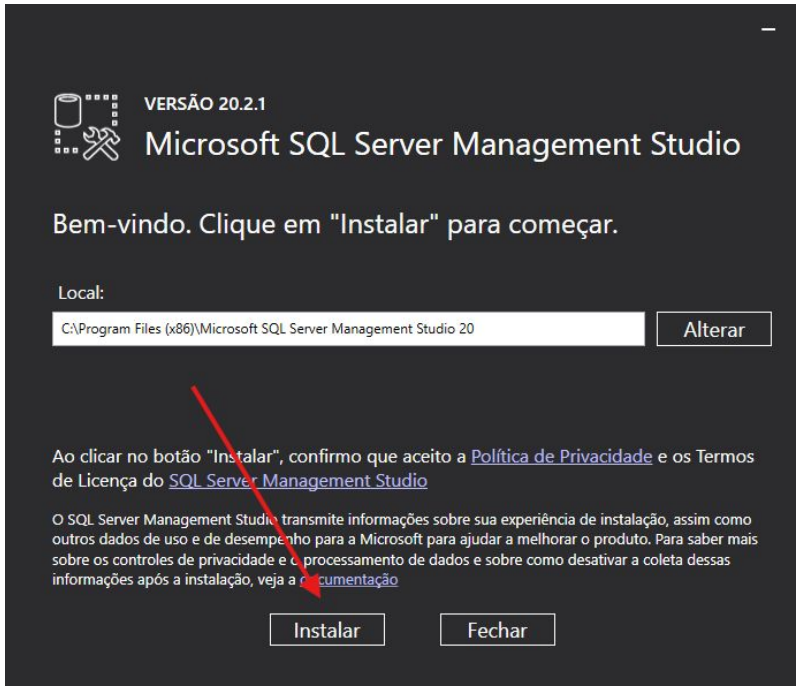
No fim da instalação o SQL SERVER e instale o SQL Server Management Studio (SSMS), utilizando o link:



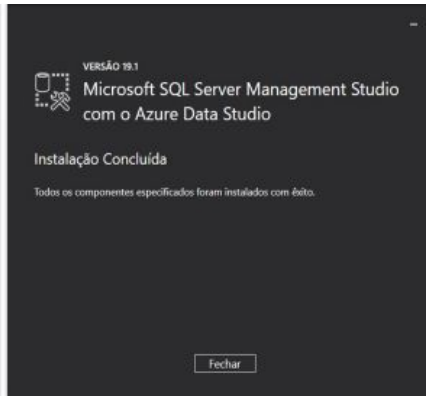

https://bit.ly/46jYTTc

# SQL Server + Python (Prática)

•No fim da instalação do SSMS, clique em ‘Fechar’:

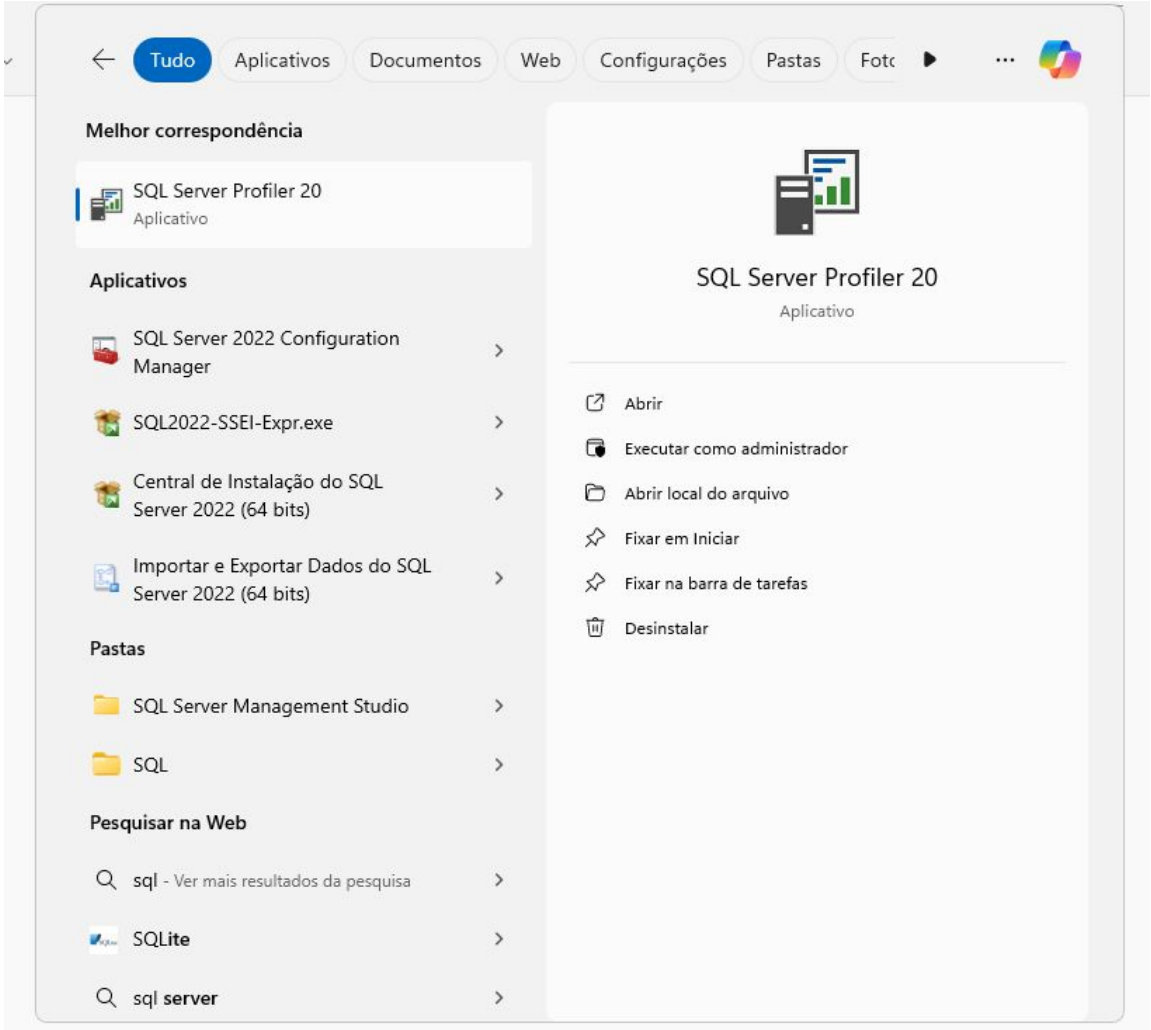






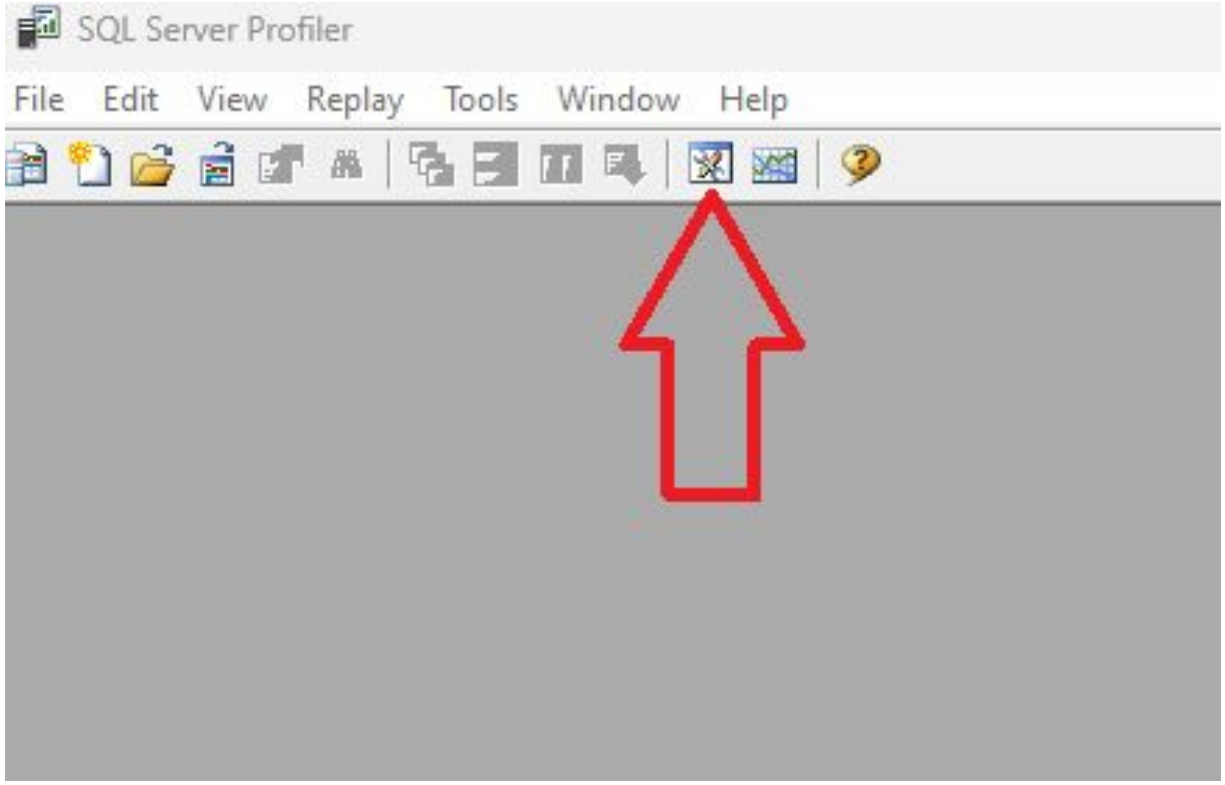
# SQL Server + Python (Prática)

- Pesquise SQL SEVER PROFILE 20 na barra de pesquisa do Windows e inicie este aplicativo:



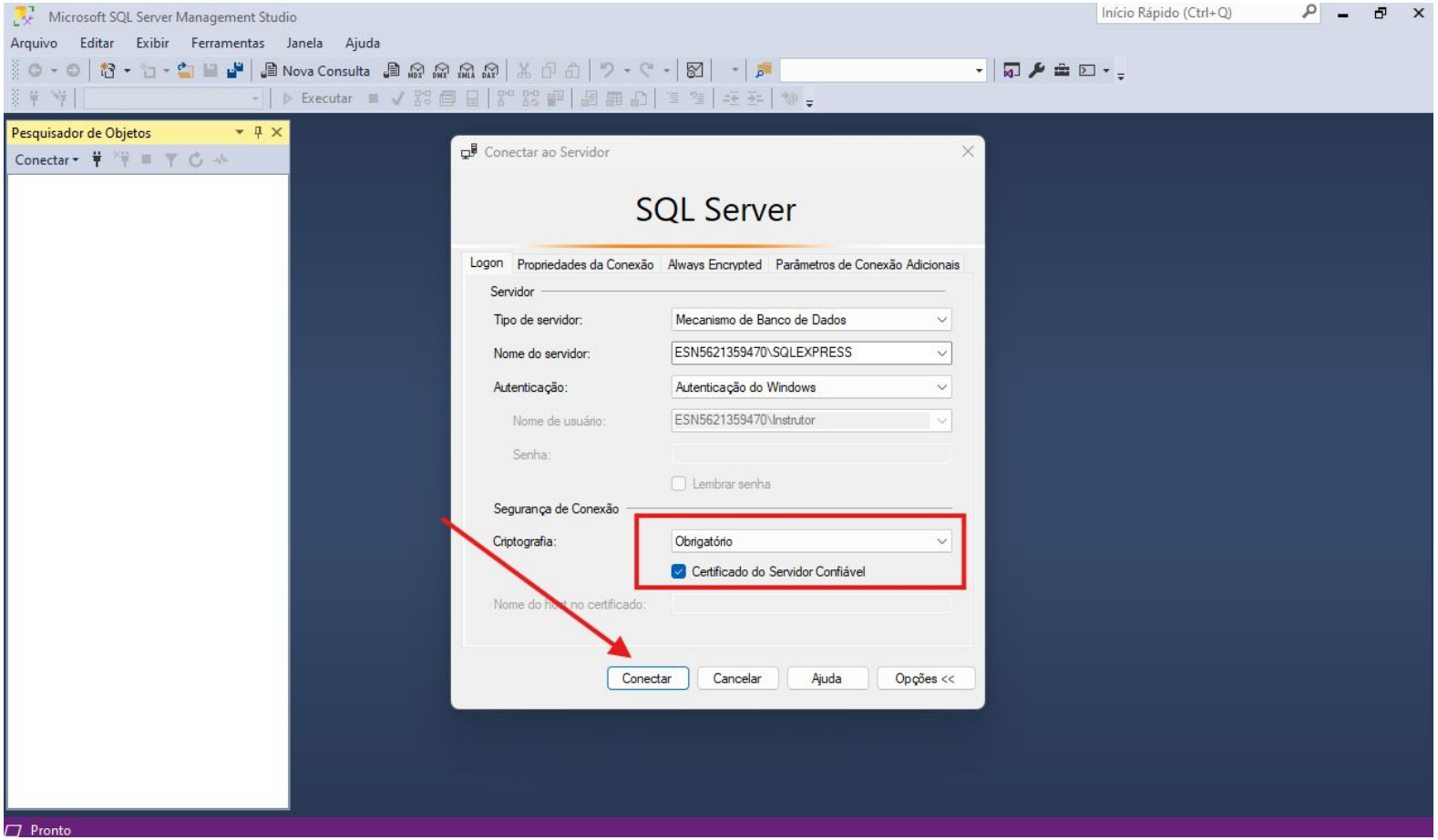
# SQL Server + Python (Prática)

•Com o SQL SEVER PROFILE 20 aberto, inicie o SSMS clicando no seguinte ícone:



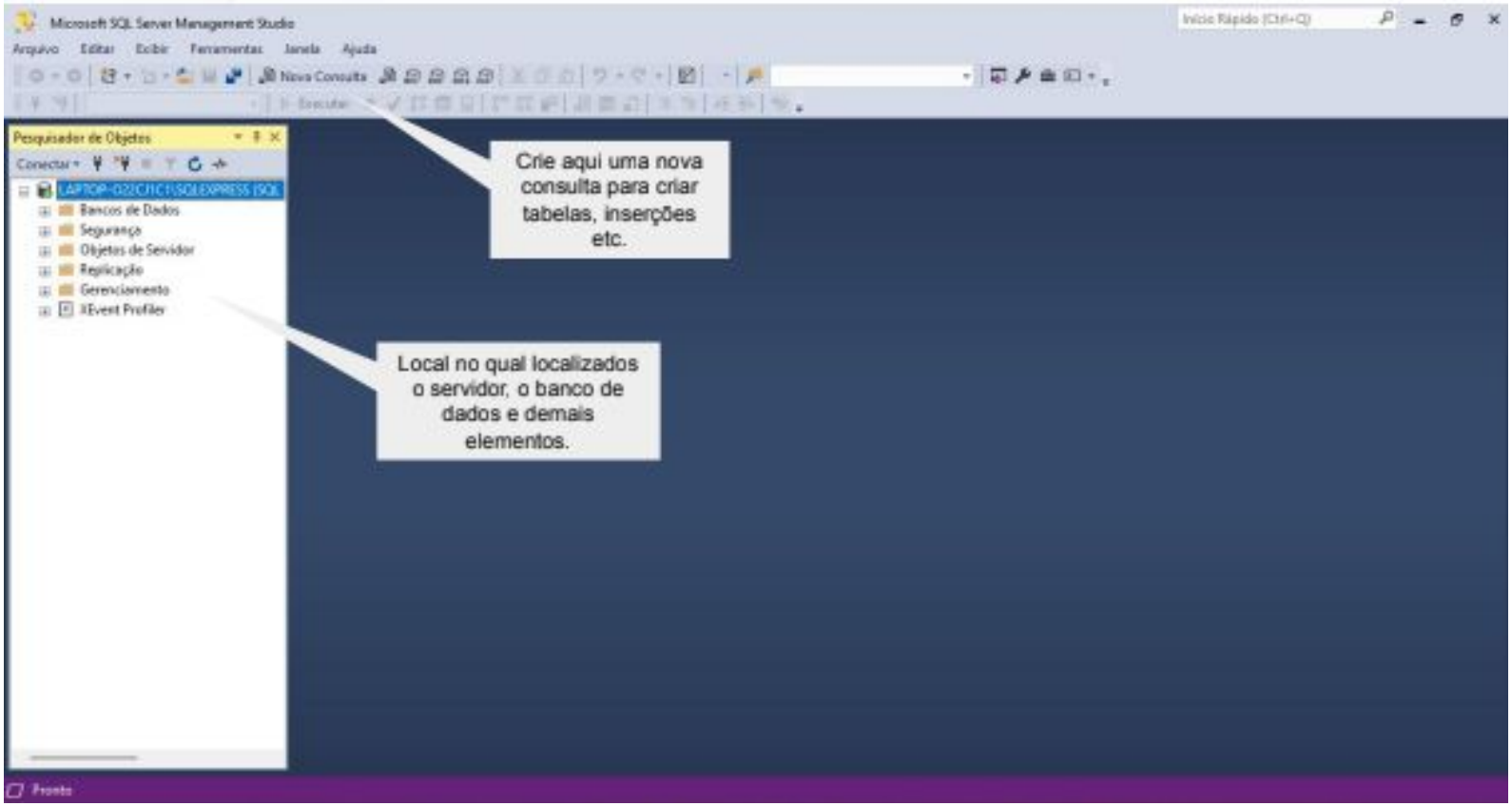
# SQL Server + Python (Prática)

- Conecte o SSMS ao SERVER, ativando a Criptografia como 'Obrigatório' e ativando 'Certificado do Servidor Confiável':



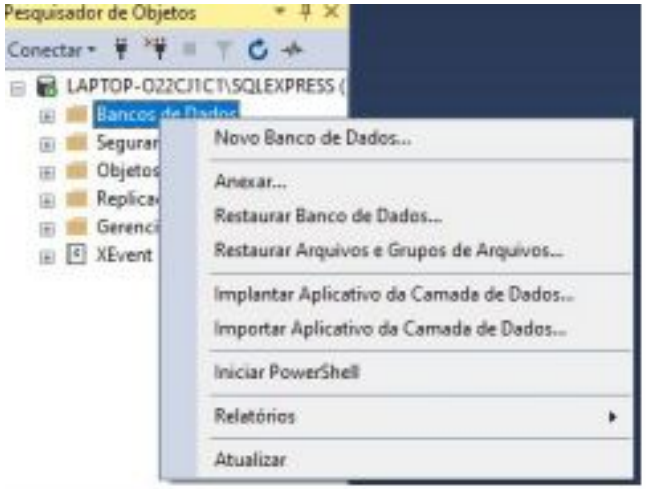
# SQL Server + Python (Prática)

- A partir deste momento, você terá acesso à página inicial do SSMS e poderá iniciar a criação da base de dados.

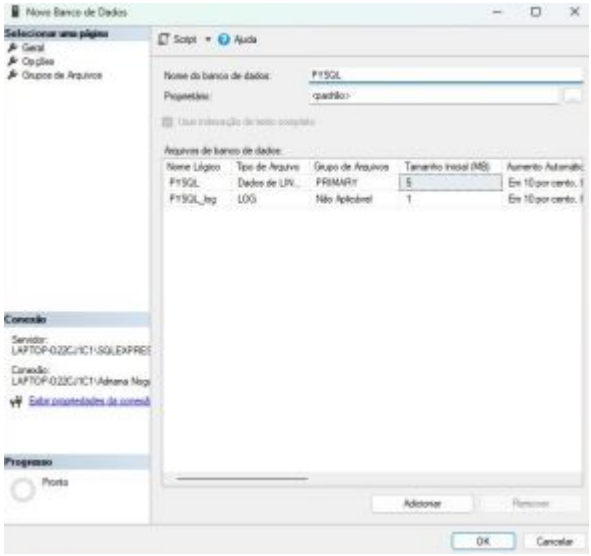


# SQL Server + Python (Prática)

•No pesquisador de objetos, localize a pasta Banco de Dados e clique sobre ela com o botão direto. Aparecerá o menu conforme a imagem a seguir:

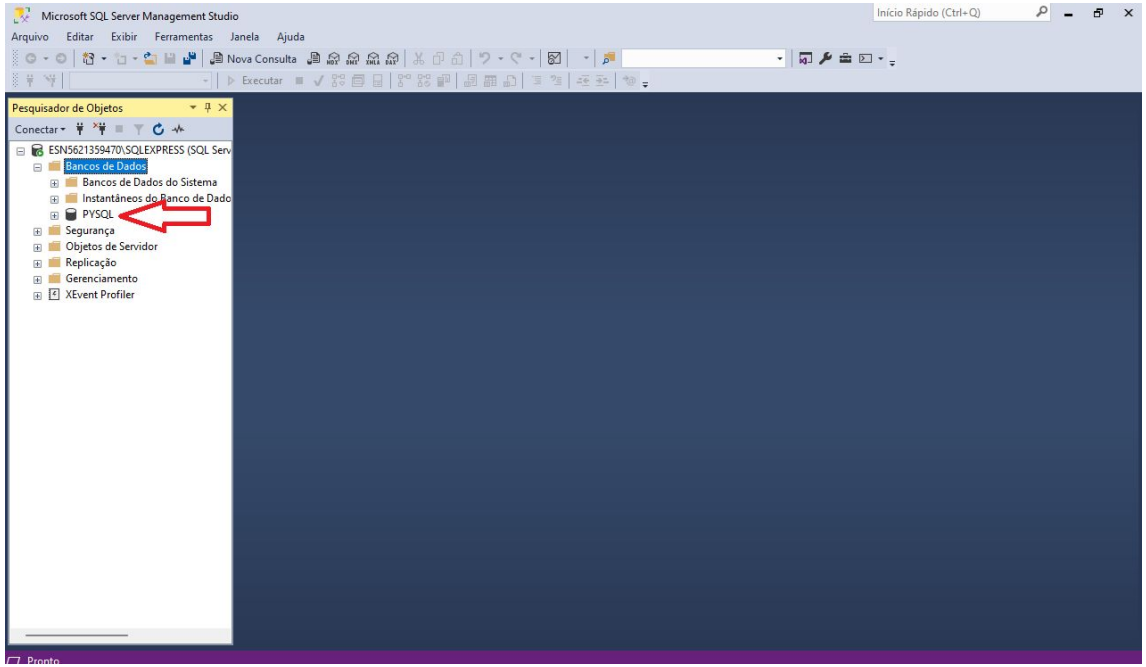



•Clique na opção Novo Banco de Dados. **Para esta atividade nomearemos o banco de dados como PYSQL**. Insira o nome no campo Nome do banco de dados e, em seguida, clique em OK.



# SQL Server + Python (Prática)

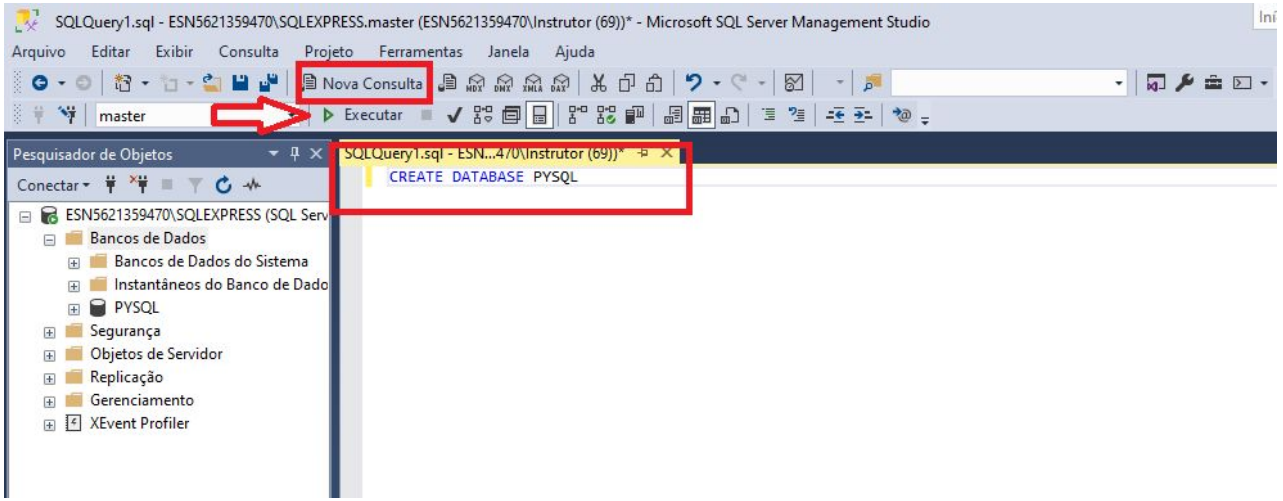
- Verifique dentro da pasta Banco de Dados se o seu banco foi criado. Pode ser necessário atualizar e/ou expandir para visualizá-lo.



# SQL Server + Python (Prática)

•**SOMENTE CURIOSIDADE:** Outra opção para a criação do banco de dados seria clicar no botão e digitar o comando a seguir :

CREATE DATABASE PYSQL



# SQL Server + Python (Prática)

- **Criação da Tabela**: Iremos criar a tabela Clientes_PF para armazenar os dados dos clientes. Nesta tabela estarão os campos: ID_Cliente_PF, Nome, Telefone e Sexo. Use a consulta a seguir e ao final clique no botão “Executar”para que seja realizada a ação solicitada.

```
USE PYSQL
CREATE TABLE CLIENTES_PF(
  ID_Cliente_PF varchar(10),
  Nome varchar(50),
  Telefone varchar(15),
  Sexo char(1)
);
```

# SQL Server + Python (Prática)

- **Inserção de Dados**: Para a inserção dos dados criaremos uma nova consulta.

O comando utilizado é o INSERT INTO, que permite a inserção de uma linha por vez. Sendo assim é possível usar os comandos abaixo numa única consulta e executá-la. Não esqueça de executá-la.

```
USE PYSQL

INSERT INTO
CLIENTES_PF(ID_Cliente_PF, Nome,
Telefone,
Sexo) VALUES (01, 'Ana',
'(19)9999-9999', 'F')

INSERT INTO CLIENTES_PF(
ID_Cliente_PF , Nome, Telefone,
Sexo) VALUES (02, 'Carlos',
'(48)3030-3030', 'M')

INSERT INTO CLIENTES_PF(
ID_Cliente_PF , Nome, Telefone,
Sexo) VALUES (03, 'Andrade',
'(58)3090-3999', 'M')
```

# SQL Server + Python (Prática)

- **Verificação dos Dados das tabelas:** Para ter certeza que os dados foram inseridos, devemos listar o conteúdo da tabela usando o comando SELECT. O retorno apresentado deve ser a seguinte tabela:

```
USE PYSQL
SELECT * FROM CLIENTES_PF
```

Resultados | Mensagens

| | ID_Cliente_PF | Nome | Telefone | Sexo |
|---|---|---|---|---|
| 1 | 1 | Ana | (19)9999-9999 | F |
| 2 | 2 | Carlos | (48)3030-3030 | M |
| 3 | 3 | Andrade | (58)3090-3999 | M |

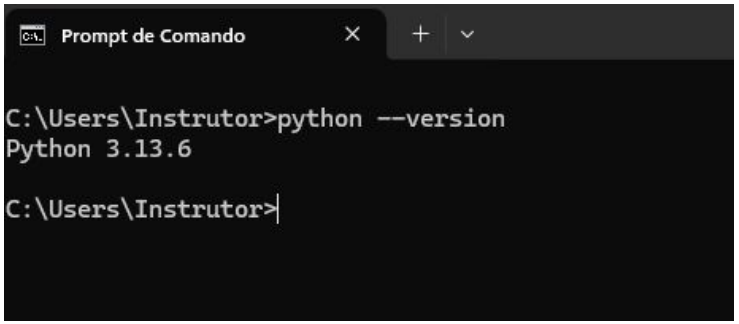
# SQL Server + Python (Prática)

Agora, precisamos verificar a instalação do Python via Prompt de Comando do Windows para o utilizarmos para a anonimização dos dados do BD:

1. **Abra o menu Iniciar:** Clique no ícone do Windows no canto inferior esquerdo da tela, ou pressione a tecla Windows.
2. **Abra o Prompt de Comando:** Digite cmd na caixa de pesquisa e selecione o aplicativo "Prompt de Comando" nos resultados.
3. **Digite o comando:** No Prompt de Comando, digite python --version e pressione Enter.
   O que esperar ao executar o comando
   - **Se o Python estiver instalado:** O prompt exibirá a versão do Python instalada, como Python 3.x.x.
   - **Se o Python não estiver instalado:** Você poderá ver uma mensagem de erro, um aviso de que o comando não foi reconhecido, ou ser sugerido a pesquisar por Python na Microsoft Store.

Caso não haja o Python instalado, acesse a página de downloads do site oficial e baixe a última versão liberada:

https://www.python.org/downloads/windows/

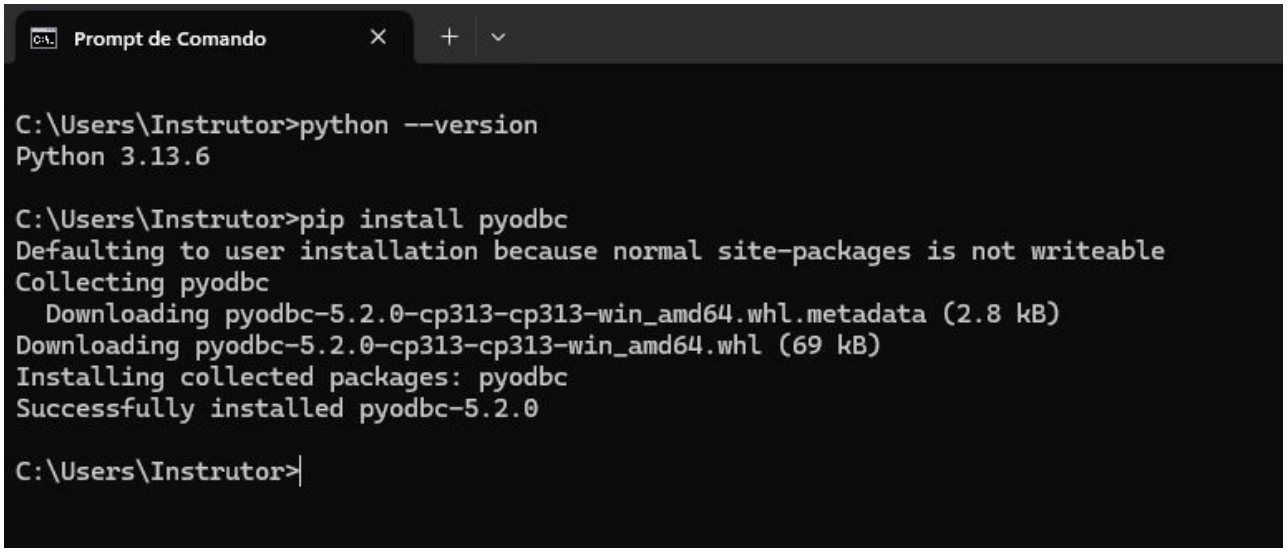


# SQL Server + Python (Prática)

**Instalando o pacote pyODBC (para conectar e executar consultas no SQL Server):** abra novamente a janela do Prompt de Comando. Digite o comando:

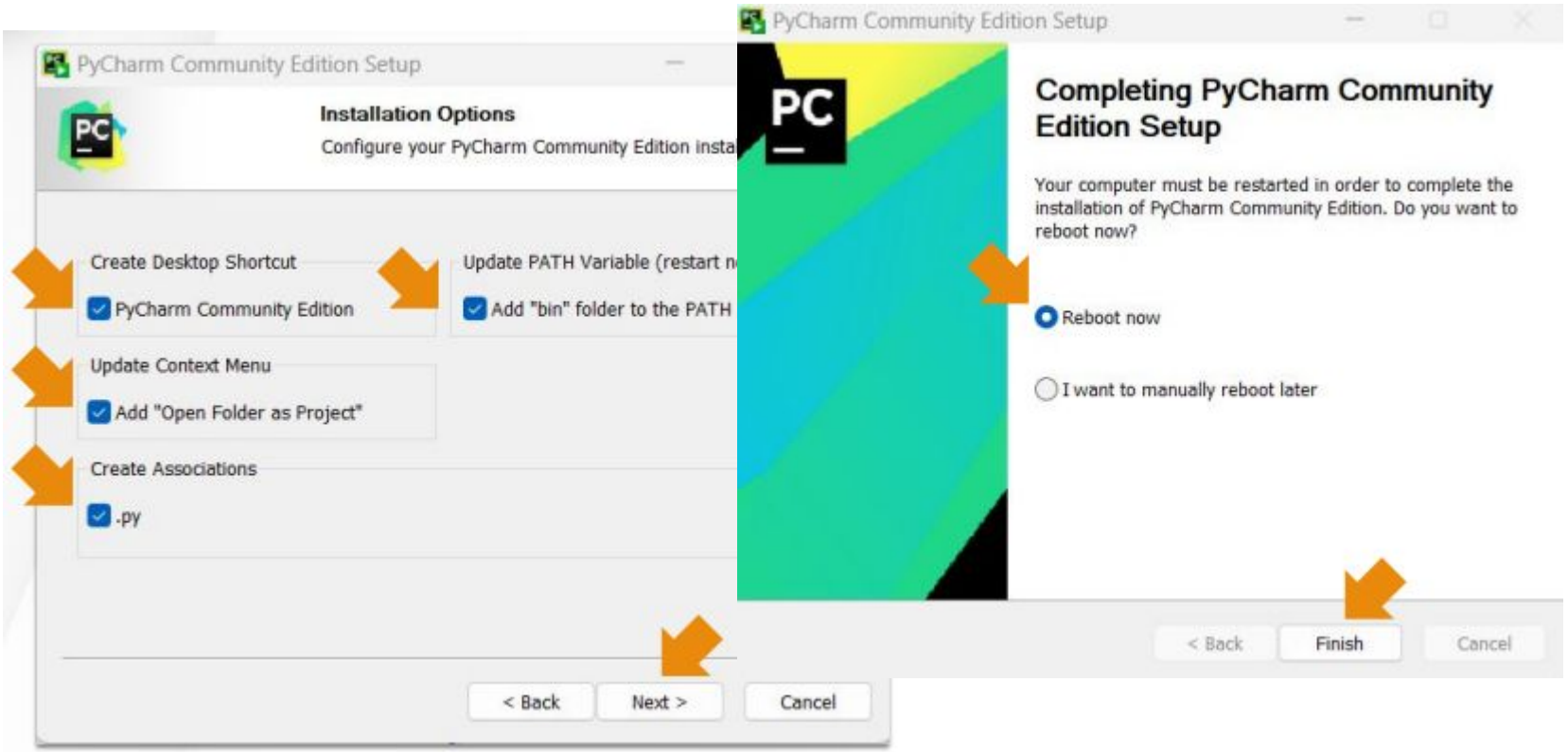
pip install pyodbc

Pressione a tecla Enter para executar. Assim que o pacote for instalado, você será notificado. Após o término esta janela pode ser fechada.



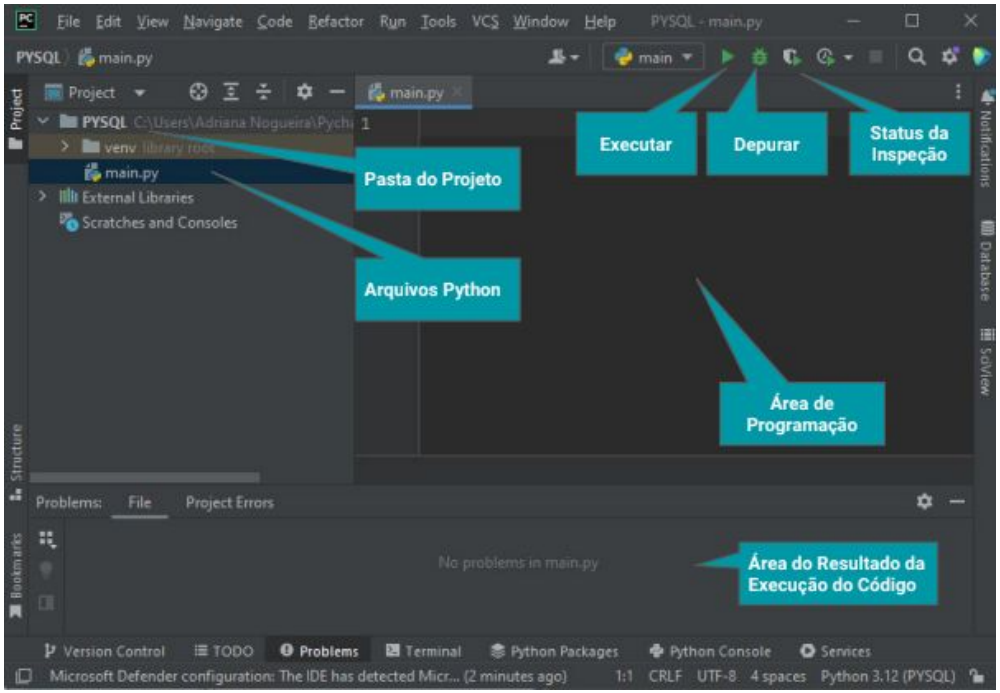
# SQL Server + Python (Prática)

**Pycharm:** O PyCharm é uma poderosa IDE (Integrated Development Environment) desenvolvida pela JetBrains, projetada especificamente para desenvolvimento em Python. Caso seja necessário o downlod, acesse o link abaixo: http://bit.ly/4m5PkwG



# SQL Server + Python (Prática)

**Criando novo Projeto Python:** Ao abrir o PyCharm pela primeira vez, uma janela chamada “New Project” aparecerá e nela, define-se as configurações necessárias para iniciar um novo projeto. É possível criar um novo projeto a qualquer momento, clicando em “File” e, em seguida, “New Project” no menu superior da janela principal do PyCharm. Primeiro, especifica-se o nome do projeto PySQL e o local padrão em que será salvo. Ao clicar em “Create”, o projeto é criado e direcionado para o ambiente de programação, contendo os recursos necessários.



# SQL Server + Python (Prática - Proposta de Solução do Estudo de Caso)

Como estudo de caso criamos um banco de dados com as informações necessárias para contatar um cliente e oferecer serviços de internet banda larga para clientes de uma empresa.

Se você estivesse vendendo um serviço de internet banda larga para seus clientes, quais seriam os dados necessários para o fornecimento do serviço?

Debata com seu grupo quais seriam esses dados. Pense na relevância das informações e na segurança dos dados. Sugerimos que esta tabela seja nomeada como **Contatos**.
Uma segunda tabela deve ser gerada para armazenar os dados dos clientes que se interessarem em fechar o serviço. Sugerimos que esta tabela seja nomeada como **Clientes**.

Após a criação das tabelas, será necessário anonimizar os dados na base. A seguir, você encontrará o código que poderá ser usado para a solução do estudo de caso.

# CONTINUAÇÃO: Script Python para Anonimização de dados em BD Relacional (pyODBC)

**Importante:**

- Certifique-se de substituir os placeholders “seu_driver”, “seu_servidor”, “seu_banco_de_dados”, “seu_usuario” e “sua_senha” pelas informações corretas do seu banco de dados.

- Substitua também “sua_tabela”, “campo”, “campo_anonimizado” e “id” pelos nomes correspondentes da tabela e colunas que deseja anonimizar.

No exemplo apresentado, são fornecidas três funções para diferentes técnicas de anonimização: encrypt_value para criptografia usando o algoritmo SHA256, mask_value para mascaramento substituindo caracteres por asteriscos e generalize_value para generalização substituindo letras por "X" e números por "0".

É possível escolher qual técnica deseja aplicar descomentando a linha correspondente e comentando as outras duas. Além disso, as funções podem ser personalizadas de acordo com a necessidade ou, ainda, implementar outras técnicas de anonimização.

# CONTINUAÇÃO: Script Python para Anonimização de dados em BD Relacional (pyODBC)

Ex: Meu placeholders com os meus dados:

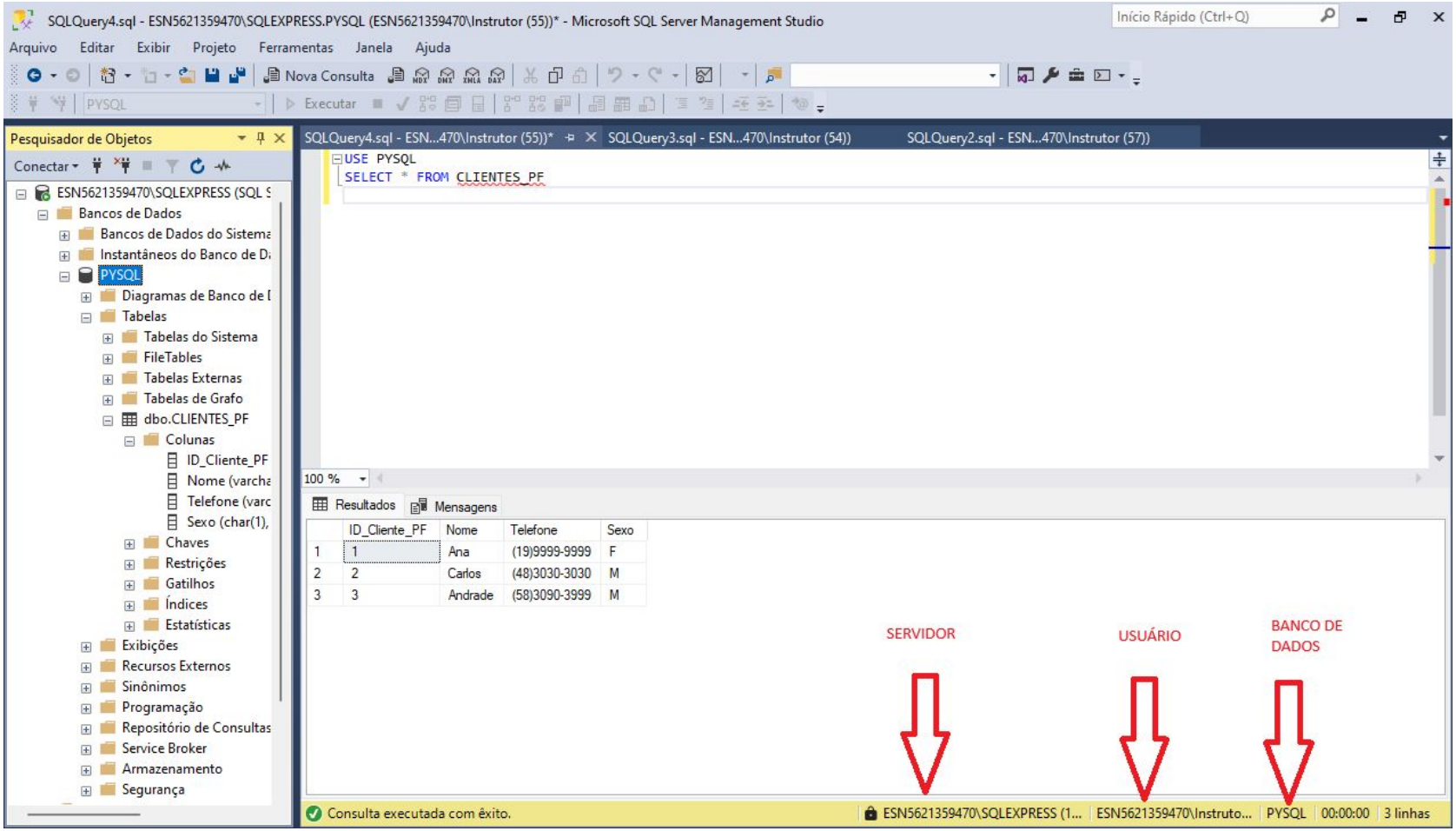
- **Servidor**: ESN5621359470\SQLEXPRESS
- **Banco de dados**: PYSQL
- **Autenticação**: Windows Authentication (não precisa de usuário/senha no pyodbc.connect)
- **Tabela**: dbo.CLIENTES_PF
- **Campo a anonimizar**: Nome
- **Identificador (id)**: todos os registros da tabela

👉 Verifique também a indentação da função *generalize_value*, que estava desalinhada.

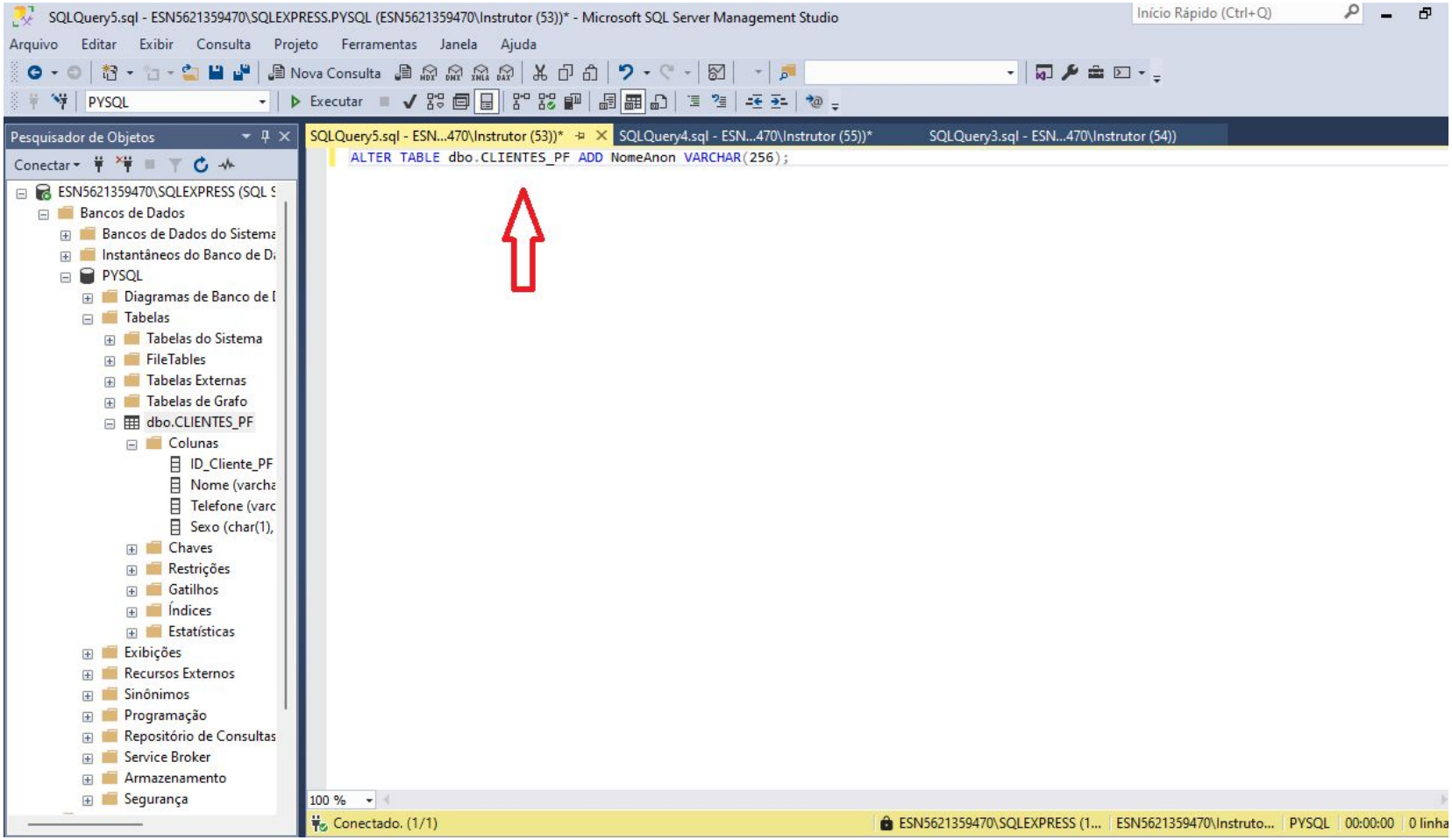
👉 Crie uma coluna nova *NomeAnon* para salvar o valor anonimizado (senão você perderia os originais).

Aqui está o código revisado e funcional:

# CONTINUAÇÃO: Script Python para Anonimização de dados em BD Relacional (pyODBC)



# CONTINUAÇÃO: Script Python para Anonimização de dados em BD Relacional (pyODBC)



# Script Python para Anonimização de dados em BD Relacional (pyODBC)

- Uso de pyODBC para conectar e executar consultas no SQL Server.
- Exemplo mostra aplicação de SHA256 para anonimizar o nome do cliente.

```
import pyodbc
import hashlib
import random
import string


# Função para gerar uma string aleatória
def generate_random_string(length):
    letters = string.ascii_letters
    return ''.join(random.choice(letters) for i in range(length))


# Função para criptografar um valor usando SHA256
def encrypt_value(value):
    hashed_value = hashlib.sha256(value.encode('utf-8')).hexdigest()
    return hashed_value
```

# CONTINUAÇÃO: Script Python para Anonimização de dados em BD Relacional (pyODBC)

```
# Função para mascarar um valor trocando
caracteres por asteriscos
def mask_value(value):
    masked_value = '*' * len(value)
    return masked_value


# Função para generalizar um valor trocando
letras por X e números por 0
def generalize_value(value):
    generalized_value = ''
    for char in value:
        if char.isalpha():
            generalized_value += 'X'
        elif char.isdigit():
            generalized_value += '0'
        else:
            generalized_value += char
    return generalized_value
```

# CONTINUAÇÃO: Script Python para Anonimização de dados em BD Relacional (pyODBC)

```
# Conectar ao banco de dados
def anonimizar_dados():
    conn = pyodbc.connect(
        r'DRIVER={ODBC Driver 17 for SQL
Server};'
        r'SERVER=ESN5621359470\SQLEXPRESS;'
        r'DATABASE=PYSQL;'
        r'Trusted_Connection=yes;'
    )
    cursor = conn.cursor()


    # Selecionar todos os registros da tabela
que deseja anonimizar
    cursor.execute("SELECT ID_Cliente_PF, Nome
FROM dbo.CLIENTES_PF")
    registros = cursor.fetchall()


    # Anonimizar cada registro
    for registro in registros:
        id_cliente = registro[0]
        nome = registro[1]
```

# CONTINUAÇÃO: Script Python para Anonimização de dados em BD Relacional (pyODBC)

```
# Escolha a técnica de anonimização desejada
        campo_anonimizado = encrypt_value(nome)        # Criptografia
        # campo_anonimizado = mask_value(nome)         # Mascaramento
        # campo_anonimizado = generalize_value(nome)   # Generalização

        # Atualizar o registro anonimizado na tabela (em uma nova coluna)
        cursor.execute(
            "UPDATE dbo.CLIENTES_PF SET NomeAnon = ? WHERE ID_Cliente_PF = ?",
            (campo_anonimizado, id_cliente)
        )

    # Confirmar as alterações no banco de dados
    conn.commit()

    # Fechar a conexão com o banco de dados
    conn.close()

# Chamada da função para anonimizar os dados
anonimizar_dados()
```

# VALE ESTE Script Python para Anonimização de dados em BD Relacional (pyODBC)

- Uso de pyODBC para conectar e executar consultas no SQL Server.
- Exemplo mostra aplicação de SHA256 para anonimizar o nome do cliente.

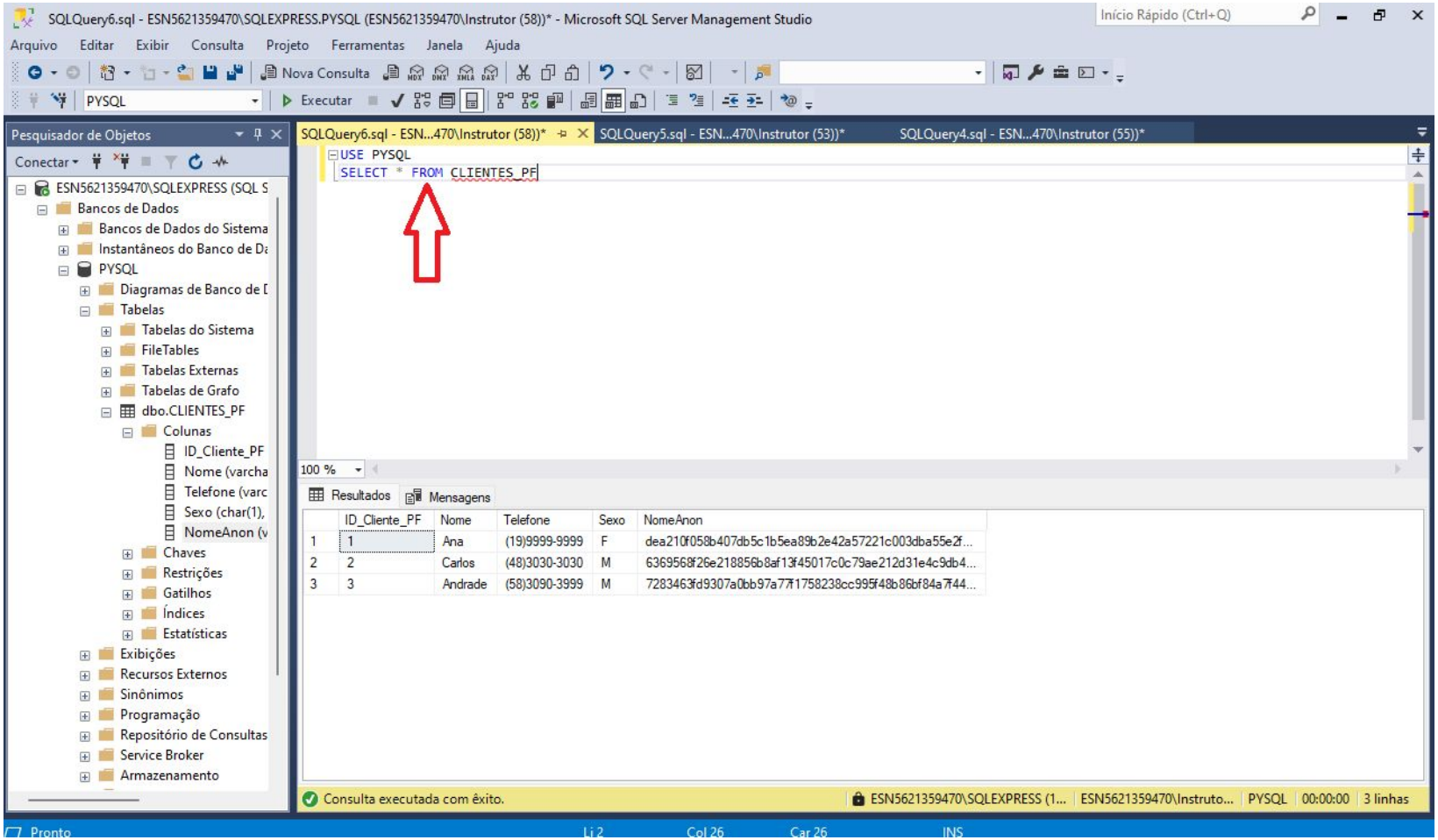
```
import pyodbc
import hashlib
import random
import string

# Função para gerar uma string aleatória
def generate_random_string(length):
    letters = string.ascii_letters
    return ''.join(random.choice(letters) for i in range(length))


# Função para criptografar um valor usando SHA256
def encrypt_value(value):
    hashed_value = hashlib.sha256(value.encode('utf-8')).hexdigest()
    return hashed_value


# Função para mascarar um valor trocando caracteres por asteriscos
def mask_value(value):
    masked_value = '*' * len(value)
    return masked_value
```

# CONTINUAÇÃO: Script Python para Anonimização de dados em BD Relacional (pyODBC)



# CONTINUAÇÃO: Script Python para Anonimização de dados em BD Relacional (pyODBC)

Como você escolheu **Autenticação do Windows**, não precisa de usuário/senha no pyodbc.connect.

Se quiser alternar entre criptografia, máscara ou generalização, basta comentar/descomentar as linhas na parte da escolha da técnica.

# Se você precisa reverter, o que você busca não é anonimização, e sim pseudonimização (ou criptografia reversível)

Nesse caso:

Em vez de usar SHA256, use uma função de criptografia simétrica (AES, Fernet do pacote cryptography).

Guarde a chave secreta em local seguro (ex.: Azure Key Vault, HashiCorp Vault, ou pelo menos em um arquivo protegido).

Assim, você pode criptografar antes de salvar no banco e descriptografar quando precisar consultar.

# Exemplo com cryptography.fernet (reversível)

```
from cryptography.fernet import Fernet
import pyodbc

# Gerar e salvar a chave (faça isso uma única vez e guarde com segurança)
# key = Fernet.generate_key()
# with open("chave.key", "wb") as chave_file:
#     chave_file.write(key)

# Carregar a chave previamente salva
with open("chave.key", "rb") as chave_file:
    key = chave_file.read()

fernet = Fernet(key)

def encrypt_value(value: str) -> str:
    return fernet.encrypt(value.encode()).decode()

def decrypt_value(value: str) -> str:
    return fernet.decrypt(value.encode()).decode()
```

# Exemplo com cryptography.fernet (reversível)

```
# Exemplo de uso:
nome = "Carlos"
nome_cripto = encrypt_value(nome)
print("Criptografado:", nome_cripto)

nome_original = decrypt_value(nome_cripto)
print("Descriptografado:", nome_original)
```

# Exemplo com cryptography.fernet (reversível)

Como integrar com seu banco CLIENTES_PF:

Armazenar criptografado na coluna NomeAnon.

Para consultar depois, ler NomeAnon e aplicar decrypt_value().

# Exemplo com cryptography.fernet (reversível)

```
# Conectar
conn = pyodbc.connect(
    r'DRIVER={ODBC Driver 17 for SQL Server};'
    r'SERVER=ESN5621359470\SQLEXPRESS;'
    r'DATABASE=PYSQL;'
    r'Trusted_Connection=yes;'
)
cursor = conn.cursor()

# Recuperar dado criptografado
cursor.execute("SELECT NomeAnon FROM dbo.CLIENTES_PF WHERE ID_Cliente_PF
= '01'")
registro = cursor.fetchone()

if registro:
    nome_original = decrypt_value(registro[0])
    print("Nome original:", nome_original)
```

# Exemplo com cryptography.fernet (reversível)

Resumo:

Se o dado foi armazenado com hash/mascara/generalização, não há volta.

Se precisa consultar novamente, deve usar criptografia reversível (como o exemplo acima com Fernet).

# Boas práticas de segurança

- Nunca mantenha chaves de criptografia no mesmo local que os dados: use HSM ou cofres (Azure Key Vault, HashiCorp Vault).
- Faça backup antes de operações destrutivas e documente o processo.
- Implemente controle de acesso (princípio do menor privilégio).
- Teste a resistência à reidentificação (privacy risk assessment).

# Atividades práticas (entregáveis)

- 1) Criar tabelas Contatos e Clientes no banco PYSQL.
- 2) Implementar duas técnicas de anonimização (por ex.: máscara + SHA256).
- 3) Gerar relatório CSV/Excel com dados anonimizados.
- 4) Submeter o código e o relatório no Google Classroom.