CENTRO PAULA SOUZA

ETEC PROFESSOR CAMARGO ARANHA

Técnico em Desenvolvimento de Sistemas

Ana Paula de Souza Rodrigues

CONEXÃO DE BANCO DE DADOS COM PYTHON

Desenvolvimento de Sistemas

São Paulo

2025

Sumário

Desenvolvimento

Conectar Python a um banco de dados Oracle é uma tarefa não tão complexa, mas que amplia as conexões e estrutura projetos de muito potencial. O processo envolve uma biblioteca Python específica e, tradicionalmente, bibliotecas de clientes da própria Oracle.

Para que o Python "converse" com um Oracle Database, você precisa de duas coisas principais:

1. **A Biblioteca Python (Driver):** é o pacote que você instala no seu ambiente Python (via `pip`). Ele fornece os comandos Python (como `.connect()`, `.execute()`, `.fetchall()`) que traduzem suas instruções para a linguagem que o Oracle entende.
    - **Biblioteca Recomendada:** `oracledb` (anteriormente conhecida como `cx_Oracle`).
2. **O Oracle Client (Bibliotecas do Cliente):** O Oracle é um sistema complexo. A biblioteca Python geralmente não se conecta *diretamente* a ele pela rede. Ela atua como um intermediário, passando os comandos para um conjunto de bibliotecas da própria Oracle (chamado de *Oracle Instant Client*), que lida com a comunicação de rede (protocolo OCI) de forma otimizada.

A biblioteca moderna `python-oracledb` agora possui dois modos de operação, o que simplifica muito o processo:

- **Modo "Thin" (Fino):**
    - **Não requer o Oracle Instant Client.**
    - A própria biblioteca `oracledb` implementa o protocolo de rede em Python puro.
    - **Vantagem:** Instalação e configuração *extremamente* fáceis. Basta um `pip install oracledb`.
    - **Desvantagem:** Pode não ter todos os recursos avançados de "enterprise" (embora seja perfeitamente adequado para a maioria dos casos de uso).

- **Modo "Thick" (Espesso):**
    - **Requer o Oracle Instant Client.**
    - Este é o modo tradicional, que usa as bibliotecas C otimizadas da Oracle.
    - **Vantagem:** Performance máxima e suporte a todos os recursos avançados do Oracle (como drivers de terceiros, autenticação Kerberos complexa, etc.).
    - **Desvantagem:** Exige o download e configuração do Instant Client no seu sistema operacional (configurando variáveis de ambiente como `PATH` no Windows ou `LD_LIBRARY_PATH` no Linux).

Uma indicação é que se comece sempre pelo **Modo Thin**. É o padrão e o mais simples. Você só precisa mudar para o Modo "Thick" se especificamente precisar de um recurso que o Modo Thin não oferece.

Instruções para uma conexão utilizando Modo Thin

Este é o método mais fácil e recomendado para começar.

Passo 1: Instalar a Biblioteca `oracledb`

- Abra seu terminal ou prompt de comando e execute:

```
Bash

pip install oracledb
```

Passo 2: Obter os Dados de Conexão

Você precisará das seguintes informações do seu DBA (Administrador de Banco de Dados) ou do seu ambiente Oracle:

- **Usuário (User):** O nome de usuário do schema.
- **Senha (Password):** A senha deste usuário.

- **DSN (Data Source Name):** Esta é a "coordenada" do banco. O formato mais fácil é o **Easy Connect String**, que se parece com:
  - `hostname:port/service_name`
  - *Exemplo para um BD local (XE):* `localhost:1521/XEPDB1`
  - *Exemplo para um BD na nuvem:* `sua-instancia.oraclecloud.com:1522/servico_de_banco.oraclecloud.com`

Passo 3: Escrever o Código Python

O processo no código é sempre:

1. Importar a biblioteca.
2. Usar `oracledb.connect()` com suas credenciais.
3. Criar um "cursor". O cursor é o objeto que realmente executa os comandos SQL.
4. Executar comandos (`.execute()`).
5. Buscar resultados (`.fetchone()`, `.fetchall()`).
6. **Importante:** Chamar `.commit()` se você fez alterações (INSERT, UPDATE, DELETE).
7. Fechar a conexão (ou, idealmente, usar blocos `with` para gerenciamento automático).

Exemplo Prático Completo (Modo Thin)

Este exemplo se conecta, cria uma tabela (se não existir), insere dados, faz um commit e, em seguida, consulta os dados.

```
Python

import oracledb
```

```
import sys # Para tratar exceções de forma limpa


# --- 1. Configure suas Credenciais ---
# (Substitua pelos seus dados reais)
# Para segurança, é melhor usar variáveis de ambiente,
# mas vamos usar variáveis simples para este exemplo.

DB_USER = "seu_usuario"
DB_PASS = "sua_senha"

# Formato Easy Connect: "hostname:port/service_name"
# Exemplo comum para Oracle XE (Express Edition) local: "localhost:1521/XEPDB1"
# Exemplo para Oracle Cloud ATP: "sua_instancia_adb.oraclecloud.com:1522/seu_servico_g.adb.oraclecloud.com"
DB_DSN = "localhost:1521/XEPDB1"

# --- 2. Bloco Principal de Conexão e Operação ---
try:
    print("Tentando conectar ao Oracle (Modo Thin)...")

    # O uso de 'with' garante que a conexão e o cursor
    # sejam fechados automaticamente, mesmo se ocorrer um erro.
```

```
    with oracledb.connect(user=DB_USER, password=DB_PASS, dsn=DB_DSN) as
connection:


        print("Conexão bem-sucedida!")
        print("Versão do Oracle Database:", connection.version)
        print("---")


        with connection.cursor() as cursor:


            # --- 3. Criar uma Tabela (DDL) ---
            print("Tentando criar a tabela 'python_funcionarios'...")
            try:
                cursor.execute("""
                    CREATE TABLE python_funcionarios (
                        id NUMBER(5) PRIMARY KEY,
                        nome VARCHAR2(100) NOT NULL,
                        cargo VARCHAR2(50)
                    )
                """)
                print("Tabela criada com sucesso.")
            except oracledb.DatabaseError as e:
                    # O erro ORA-00955 significa "name is already used by an existing
object"
                # (A tabela já existe, o que não é um problema)
```

```
    if "ORA-00955" in str(e.args[0]):
        print("A tabela 'python_funcionarios' já existe. Continuando.")
    else:
        # Se for outro erro, levanta a exceção
        raise

# --- 4. Inserir Dados (DML) ---
print("Inserindo dados...")

# Limpar dados de execuções anteriores (para este exemplo)
cursor.execute("DELETE FROM python_funcionarios")

# Usar 'bind variables' (:1, :2) é a prática segura
# para evitar SQL Injection.
dados_para_inserir = [
    (101, 'Ana Silva', 'Desenvolvedora'),
    (102, 'Bruno Costa', 'Analista de Dados'),
    (103, 'Carla Mendes', 'Gerente de Projetos')
]

# .executemany() é eficiente para inserir várias linhas
cursor.executemany("""
    INSERT INTO python_funcionarios (id, nome, cargo)
```

```
            VALUES (:1, :2, :3)
        """, dados_para_inserir)

        print(f"{cursor.rowcount} linhas inseridas.")

        # --- 5. COMMIT (ESSENCIAL!) ---
        # Sem commit, as inserções (DML) não são salvas permanentemente.
        print("Executando COMMIT...")
        connection.commit()

        # --- 6. Consultar Dados (Query) ---
        print("\nConsultando dados da tabela:")
            cursor.execute("SELECT id, nome, cargo FROM python_funcionarios
ORDER BY nome")

        # .fetchall() busca todas as linhas retornadas pela consulta
        for linha in cursor.fetchall():
            # Cada 'linha' é uma tupla
            print(f"  ID: {linha[0]}, Nome: {linha[1]}, Cargo: {linha[2]}")

        print("\n---")
        print("Operação concluída com sucesso.")

except oracledb.Error as e:
```

```
    # Captura erros específicos do Oracle (Ex: senha errada, BD offline)
    print(f"Erro de banco de dados Oracle: {e}")
    err_obj, = e.args
    print(f"  Código do Erro: {err_obj.code}")
    print(f"  Mensagem: {err_obj.message}")
    sys.exit(1) # Sai do script com código de erro

except Exception as e:
    # Captura qualquer outro erro inesperado
    print(f"Um erro inesperado ocorreu: {e}")
    sys.exit(1)
```

Conclusão

Na execução desta pesquisa foi identificado que a comunicação do banco de dados da Oracle com o Python requer, assim como a maioria das conexões, passos que não se pode pular, como as instalações e definições da biblioteca utilizada.

Para uma conexão íntegra e precisa é necessário acompanhamento de documentações oficiais que tem muito a explicar e acrescentar ao que será executado, além dos padrões temos opções que podemos recorrer e cada uma delas segue suas peculiaridades e refletem no resultado final do nosso código.

Referências

https://python-oracledb.readthedocs.io/

https://www.oracle.com/database/technologies/instant-client/downloads.html

https://docs.oracle.com/en/database/oracle/oracle-database/19/netag/configuring-naming-methods.html#GUID-8C26E955-6670-4E15-A18B-3C8A42763321

https://gemini.google.com