RECONHECER AS FUNCIONALIDADES DE FRAMEWORKS E BIBLIOTECAS PARA GERENCIAMENTO DE BANCO DE DADOS

CONCEITOS

Conectores para banco de dados

Quando estamos trabalhando com banco de dados, precisamos pesquisar por bibliotecas que possibilitem a conexão da aplicação com o banco de dados que iremos utilizar.

Na área de banco de dados, essas bibliotecas se chamam conectores ou adaptadores.

Figura: 1.

Como estamos trabalhando com Python, devemos procurar por conectores que atendam a PEP 249.

Essa PEP especifica um conjunto de padrões que os conectores devem seguir, a fim de garantir um melhor entendimento dos módulos e maior portabilidade entre bancos de dados.

Como exemplo de padrões definidos pela DB-API 2.0, podemos citar:

- **Nomes de métodos:**
  - close, commit e cursor
- **Nomes de Classes:**
  - Connection e Cursor
- **Tipos de exceção:**
  - IntegrityError e InternalError

Como exemplo de conectores que implementam a DB-API 2.0, temos:

- **psycopg2:** Conector mais utilizado para o banco de dados PostgreSQL.
- **mysqlclient, PyMySQL e mysql-connector-python:** Conectores para o banco de dados MySQL.
- **sqlite3:** Adaptador padrão do Python para o banco de dados SQLite.

Comentário

Apenas o conector para SQLite está disponível nativamente no Python 3.7. Todos os demais precisam ser instalados.

Como dito na introdução, neste tema, utilizaremos o banco de dados SQLite para demonstrar o desenvolvimento de aplicações Python para banco de dados.

Apesar do nome Lite, o SQLite é um banco de dados completo, que permite a criação de tabelas, relacionamentos, índices, gatilhos e visões.

O SQLite também tem suporte a subconsultas, transações, junções, pesquisa em texto (full text search), restrições de chave estrangeira, entre outras funcionalidades.

Porém, por não ter um servidor para seu gerenciamento, o SQLite não provê um acesso direto pela rede. As soluções existentes para acesso remoto são “caras” e ineficientes.

Além disso, o SQLite não tem suporte à autenticação, com usuários e permissões definidas. Estamos falando aqui sobre usuários do banco de dados, que têm permissão para criar tabela, inserir registros etc.

Observe que, apesar do SQLite não ter suporte à autenticação, podemos e devemos implementar nosso próprio controle de acesso para nossa aplicação.

PRINCIPAIS MÉTODOS DOS CONECTORES EM PYTHON

Antes de começarmos a trabalhar diretamente com banco de dados, precisamos conhecer algumas classes e métodos disponibilizados pelos conetores e previstos na PEP 249.

Esses métodos são a base para qualquer aplicação que necessite de acesso a banco de dados e estão descritos a seguir:

CONNECT

- Função global do conector para criar uma conexão com o banco de dados.
- Retorna um objeto do tipo Connection.

CONNECTION

- Classe utilizada para gerenciar todas as operações no banco de dados.
- Principais métodos:

- commit: Confirma todas as operações pendentes;
- rollback: Desfaz todas as operações pendentes;
- cursor: Retorna um objeto do tipo Cursor; e
- close: Encerra a conexão com o banco de dados.

CURSOR

- Classe utilizada para enviar os comandos ao banco de dados.
- Principais métodos:
  - execute: Prepara e executa a operação passada como parâmetro;
  - fetchone: Retorna a próxima linha encontrada por uma consulta; e
  - fetchall: Retorna todas as linhas encontradas por uma consulta.

A utilização desses métodos segue basicamente o mesmo fluxo de trabalho para todas as aplicações que utilizam banco de dados. A seguir, vamos descrever esse fluxo:

Criar uma conexão com o banco de dados utilizando a função connect.

Utilizar a conexão para criar um cursor, que será utilizado para enviar comandos ao banco de dados.

Utilizar o cursor para enviar comandos ao banco de dados, por exemplo:

a. Criar tabelas.
b. Inserir linhas.
c. Selecionar linhas.

Efetivar as mudanças no banco de dados utilizando o método commit da conexão.

Fechar o cursor e a conexão

Observe os scripts a seguir na Figura 2, na qual temos esse fluxo de trabalho descrito na forma de código Python.

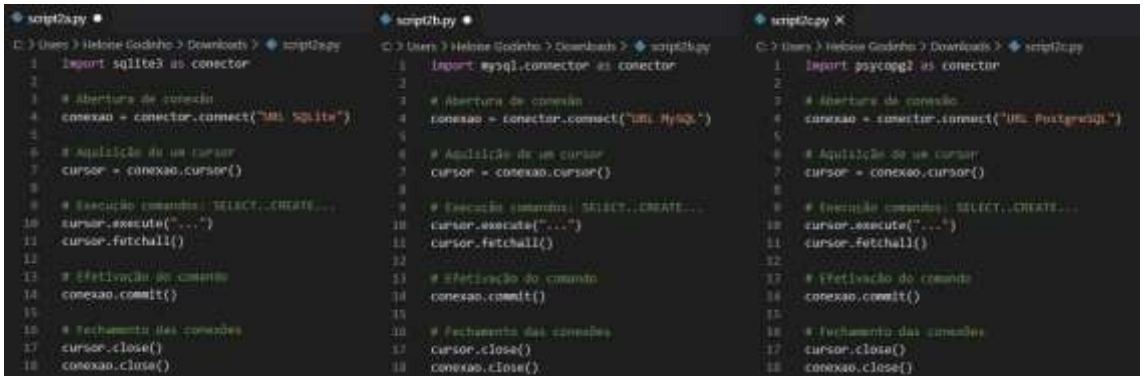

Figura: 2

Neste momento, vamos focar apenas no script1, mais à esquerda da figura.

Na primeira linha, importamos o módulo sqlite3, conector para o banco de dados SQLite disponível nativamente no Python 3.7. Atribuímos um alias (apelido) a esse import chamado conector. Veremos ao final por que criamos esse alias.

Na linha 4, abrimos uma conexão com o banco de dados utilizando uma URL. O retorno dessa conexão é um objeto da classe Connection, que foi atribuído à variável conexao.

Na linha 7, utilizamos o método cursor da classe Connection para criar um objeto do tipo Cursor, que foi atribuído à variável cursor.

Na linha 10, utilizamos o método execute, da classe Cursor. Este método permite enviar comandos SQL para o banco de dados. Entre os comandos, podemos citar: SELECT, INSERT e CREATE.

Na linha 11, usamos o método fetchall, da classe Cursor, que retorna os resultados de uma consulta.

Na linha 14, utilizamos o método commit, da classe Connection, para efetivar todos os comandos enviados anteriormente. Se desejássemos desfazer os comandos, poderíamos utilizar o método rollback.

Nas linhas 17 e 18, fechamos o cursor e a conexão, respectivamente.

A estrutura apresentada irá se repetir ao longo deste tema, onde, basicamente, preencheremos o parâmetro do método execute com o comando SQL pertinente.

Atenção

Agora observe os scripts 2 e 3 da Figura 2. e veja que ambas as bibliotecas***mysql.connector*** e ***psycopg2*** contêm os mesmos métodos e funções da biblioteca ***sqlite3.***

Como estamos utilizando o ***alias conector*** no import, para trocar de banco de dados, bastaria apenas alterar o import da primeira linha. Isso se deve ao fato de que todas elas seguem a especificação DB-API 2.0.

Cada biblioteca tem suas particularidades, que podem adereçar alguma funcionalidade específica do banco de dados referenciado, mas todas elas implementam, da mesma maneira, o básico para se trabalhar com banco de dados.

Isso é de grande utilidade, pois podemos alterar o banco de dados a qualquer momento, sem a necessidade de realizar muitas alterações no código-fonte.

Mais adiante, mostraremos algumas situações onde as implementações podem diferir entre os conectores.

PRINCIPAIS EXCEÇÕES DOS CONECTORES EM PYTHON

Além dos métodos, a DB-API 2.0 prevê algumas exceções que podem ser lançadas pelos conectores.

Vamos listar e explicar algumas dessas exceções:

Error

Classe base para as exceções. É a mais abrangente.

IntegrityError

Exceção para tratar erros relacionados à integridade do banco de dados, como falha na checagem de chave estrangeira e falha na checagem de valores únicos. É uma subclasse de DatabaseError.

OperationalError

Exceção para tratar erros relacionados a operações no banco de dados, mas que não estão sob controle do programador, como desconexão inesperada. É uma subclasse de DatabaseError.

DatabaseError

Exceção para tratar erros relacionados ao banco de dados. Também é uma exceção abrangente. É uma subclasse de Error.

ProgrammingError

Exceção para tratar erros relacionados à programação, como sintaxe incorreta do comando SQL, tabela não encontrada etc. É uma subclasse de DatabaseError.

NotSupportedError

Exceção para tratar erros relacionados a operações não suportadas pelo banco de dados, como chamar o método rollback em um banco de dados que não suporta transações. É uma subclasse de DatabaseError.

TIPOS DE DADOS

Cada dado armazenado em um banco de dados contém um tipo, como inteiro, texto, ponto flutuante, entre outros.

Os tipos suportados pelos bancos de dados não são padronizados e, por isso, é necessário verificar na sua documentação quais tipos são disponibilizados.

O SQLite trata os dados armazenados de um modo um pouco diferente de outros bancos, nos quais temos um número limitado de tipos.

No SQLite, cada valor armazenado no banco é de uma das classes a seguir:

NULL	Para valores nulos.
INTEGER	Para valores que são números inteiros, com sinal.
REAL	Para valores que são números de ponto flutuante.
TEXT	Para valores que são texto (string).
BLOB	Para armazenar valores exatamente como foram inseridos, ex. bytes.

Apesar de parecer um número limitado de classes, quando comparado com outros bancos de dados, o SQLite suporta o conceito de **afinidades de tipo** para as colunas.

No SQLite, quando definimos uma coluna durante a criação de uma tabela, ao invés de especificar um tipo estático, dizemos qual a afinidade dela. Isso nos permite armazenar diferentes tipos de dados em uma mesma coluna. Observe as afinidades disponíveis a seguir:

TEXT	Coluna para armazenar dados das classes NULL, TEXT e BLOB.
NUMERIC	Coluna para armazenar dados de qualquer uma das cinco classes.
INTEGER	Similar ao NUMERIC, diferenciando apenas no processo de conversão de valores.
REAL	Similar ao NUMERIC, porém os valores inteiros são forçados a serem representados como ponto flutuante.
NONE	Coluna sem preferência de armazenamento, não é realizada nenhuma conversão de valores.

A afinidade também permite ao motor do SQLite fazer um mapeamento entre tipos não suportados e tipos suportados. Por exemplo, o tipo *VARCHAR(n)*, disponível no *MySQL* e *PostgreSQL*, é convertido para *TEXT* no *SQLite*.

Esse mapeamento nos permite definir atributos no *CREATE TABLE* com outros tipos, não suportados pelo *SQLite*. Esses tipos são convertidos para tipos conhecidos utilizando afinidade.

A tabela de afinidades do *SQLite* está descrita a seguir:

Tipo no CREATE TABLE	Afinidade
INT INTEGER TINYINT SMALLINT MEDIUMINT BIGINT UNSIGNED BIG INT INT2 INT8	INTEGER
CHARACTER(20) VARCHAR(255) VARYING CHARACTER(255) NCHAR(55) NATIVE CHARACTER(70) NVARCHAR(100) TEXT CLOB	TEXT
BLOB	BLOB
REAL DOUBLE DOUBLE PRECISION FLOAT	REAL
NUMERIC DECIMAL(10,5) BOOLEAN DATE DATETIME	NUMERIC

A partir desta tabela, mesmo utilizando o SQLite para desenvolvimento, podemos definir os atributos de nossas tabelas com os tipos que utilizaremos em ambiente de produção.

Fontes das Informações:

Casos de dengues (datasus) e Estimativas da População (IBGE)