Principais Comandos SQL e Exemplos

José Alfredo F. Costa – 2024 (Agosto)

Principais Comandos SQL:

1. SELECT - Recupera dados de um banco de dados Tradução: SELECIONAR Exemplo:

```
SELECT nome, idade FROM clientes WHERE idade > 18;
```

Este comando seleciona o nome e a idade de todos os clientes maiores de 18 anos.

2. INSERT - Adiciona novos registros a uma tabela Tradução: INSERIR Exemplo:

```
INSERT INTO produtos (nome, preco) VALUES ('Laptop', 1200.00);
```

Este comando insere um novo produto chamado 'Laptop' com preço de 1200.00 na tabela de produtos.

3. UPDATE - Modifica registros existentes em uma tabela Tradução: ATUALIZAR Exemplo:

```
UPDATE funcionarios SET salario = salario * 1.1 WHERE departamento =
'TI';
```

Este comando aumenta em 10% o salário de todos os funcionários do departamento de TI.

4. DELETE - Remove registros de uma tabela Tradução: EXCLUIR Exemplo:

```
DELETE FROM pedidos WHERE data < '2023-01-01';
```

Este comando exclui todos os pedidos feitos antes de 1º de janeiro de 2023.

5. CREATE DATABASE - Cria um novo banco de dados Tradução: CRIAR BANCO DE DADOS Exemplo:

```
CREATE DATABASE minha_loja;
```

Este comando cria um novo banco de dados chamado "minha_loja".

6. CREATE TABLE - Cria uma nova tabela Tradução: CRIAR TABELA Exemplo:

```
CREATE TABLE clientes (
```

```
    id INT PRIMARY KEY,
    nome VARCHAR(100),
    email VARCHAR(100)
);
```

Este comando cria uma nova tabela chamada "clientes" com colunas para id, nome e email.

7. DROP DATABASE - Exclui um banco de dados inteiro e seu conteúdo Tradução: EXCLUIR BANCO DE DADOS Exemplo:

```
DROP DATABASE banco_antigo;
```

Este comando exclui completamente o banco de dados chamado "banco_antigo".

8. DROP TABLE - Exclui uma tabela Tradução: EXCLUIR TABELA Exemplo:

```
DROP TABLE produtos_obsoletos;
```

Este comando remove a tabela "produtos_obsoletos" do banco de dados.

9. ALTER TABLE - Modifica a estrutura de uma tabela existente Tradução: ALTERAR TABELA Exemplo:

```
ALTER TABLE clientes ADD COLUMN telefone VARCHAR(20);
```

Este comando adiciona uma nova coluna chamada "telefone" à tabela "clientes".

10. TRUNCATE TABLE - Remove todos os registros de uma tabela Tradução: TRUNCAR TABELA Exemplo:

```
TRUNCATE TABLE logs_temporarios;
```

Este comando remove todos os dados da tabela "logs_temporarios", mas mantém a estrutura da tabela.

11. CREATE INDEX - Cria um índice em uma coluna da tabela para melhorar o desempenho das consultas Tradução: CRIAR ÍNDICE Exemplo:

```
CREATE INDEX idx_sobrenome ON clientes(sobrenome);
```

Este comando cria um índice na coluna "sobrenome" da tabela "clientes" para acelerar buscas por sobrenome.

12. DROP INDEX - Remove um índice de uma tabela Tradução: EXCLUIR ÍNDICE Exemplo:

```
DROP INDEX idx_sobrenome ON clientes;
```

Este comando remove o índice "idx_sobrenome" da tabela "clientes".

13. INNER JOIN - Recupera registros que têm valores correspondentes em ambas as tabelas Tradução: JUNÇÃO INTERNA Exemplo:

```
SELECT pedidos.id, clientes.nome
FROM pedidos
INNER JOIN clientes ON pedidos.cliente_id = clientes.id;
```

Este comando retorna os IDs dos pedidos e os nomes dos clientes correspondentes.

14. LEFT JOIN - Recupera todos os registros da tabela da esquerda e os registros correspondentes da tabela da direita Tradução: JUNÇÃO À ESQUERDA Exemplo:

```
SELECT clientes.nome, pedidos.id
FROM clientes
LEFT JOIN pedidos ON clientes.id = pedidos.cliente_id;
```

Este comando retorna todos os clientes, mesmo aqueles que não fizeram pedidos.

15. RIGHT JOIN - Recupera todos os registros da tabela da direita e os registros correspondentes da tabela da esquerda Tradução: JUNÇÃO À DIREITA Exemplo:

```
SELECT funcionarios.nome, departamentos.nome
FROM funcionarios
RIGHT JOIN departamentos ON funcionarios.depto_id = departamentos.id;
```

Este comando retorna todos os departamentos, mesmo aqueles sem funcionários.

16. FULL OUTER JOIN - Recupera todos os registros quando há uma correspondência em qualquer uma das tabelas Tradução: JUNÇÃO EXTERNA COMPLETA Exemplo:

```
SELECT *
FROM clientes
FULL OUTER JOIN pedidos ON clientes.id = pedidos.cliente_id;
```

Este comando retorna todos os clientes e todos os pedidos, mostrando correspondências onde existirem.

17. UNION - Combina os conjuntos de resultados de duas ou mais instruções SELECT Tradução: UNIÃO Exemplo:

```
SELECT nome FROM clientes
UNION
SELECT nome FROM funcionarios;
```

Este comando retorna uma lista única com nomes de clientes e funcionários, sem duplicatas.

18. UNION ALL - Combina os conjuntos de resultados de duas ou mais instruções SELECT, incluindo duplicatas Tradução: UNIÃO COMPLETA Exemplo:

```
SELECT cidade FROM clientes
UNION ALL
SELECT cidade FROM fornecedores;
```

Este comando retorna todas as cidades de clientes e fornecedores, incluindo duplicatas.

19. DISTINCT - Recupera valores únicos de uma coluna Tradução: DISTINTO Exemplo:

```
SELECT DISTINCT categoria FROM produtos;
```

Este comando retorna uma lista de categorias únicas de produtos.

20. WHERE - Filtra registros com base em condições especificadas Tradução: ONDE Exemplo:

```
SELECT nome, salario FROM funcionarios WHERE salario > 5000;
```

Este comando seleciona nomes e salários de funcionários que ganham mais de 5000.

Estes comandos são fundamentais para manipulação e consulta de dados em SQL.

21. ORDER BY - Ordena o conjunto de resultados com base em colunas especificadas Tradução: ORDENAR POR Exemplo:

```
SELECT nome, salario FROM funcionarios ORDER BY salario DESC;
```

Este comando retorna os nomes e salários dos funcionários, ordenados do maior para o menor salário.

22. GROUP BY - Agrupa o conjunto de resultados por uma ou mais colunas Tradução: AGRUPAR POR Exemplo:

```
SELECT departamento, COUNT(*) AS total_funcionarios
FROM funcionarios
GROUP BY departamento;
```

Este comando conta quantos funcionários há em cada departamento.

23. HAVING - Filtra grupos baseados em condições especificadas Tradução: TENDO Exemplo:

```
SELECT departamento, AVG(salario) AS media_salarial
FROM funcionarios
GROUP BY departamento
HAVING AVG(salario) > 5000;
```

Este comando mostra departamentos cuja média salarial é superior a 5000.

24. COUNT - Retorna o número de linhas em um conjunto de resultados Tradução: CONTAR Exemplo:

```
SELECT COUNT(*) AS total_produtos FROM produtos;
```

Este comando conta o número total de produtos na tabela.

25. SUM - Calcula a soma dos valores em uma coluna Tradução: SOMA Exemplo:

```
SELECT SUM(valor) AS receita_total FROM vendas;
```

Este comando calcula a receita total somando todos os valores da coluna 'valor' na tabela 'vendas'.

26. AVG - Calcula a média dos valores em uma coluna Tradução: MÉDIA Exemplo:

```
SELECT AVG(preco) AS preco_medio FROM produtos;
```

Este comando calcula o preço médio dos produtos.

27. MIN - Recupera o valor mínimo de uma coluna Tradução: MÍNIMO Exemplo:

```
SELECT MIN(idade) AS idade_minima FROM clientes;
```

Este comando encontra a idade do cliente mais jovem.

28. MAX - Recupera o valor máximo de uma coluna Tradução: MÁXIMO Exemplo:

```
SELECT MAX(salario) AS maior_salario FROM funcionarios;
```

Este comando encontra o maior salário entre os funcionários.

29. BETWEEN - Recupera valores dentro de um intervalo especificado Tradução: ENTRE Exemplo:

```
SELECT nome, preco FROM produtos WHERE preco BETWEEN 100 AND 500;
```

Este comando seleciona produtos com preço entre 100 e 500.

30. LIKE - Pesquisa por um padrão específico em uma coluna Tradução: COMO Exemplo:

```
SELECT nome FROM clientes WHERE nome LIKE 'Jo%';
```

Este comando encontra clientes cujos nomes começam com "Jo".

31. IN - Verifica se um valor está em uma lista especificada Tradução: EM Exemplo:

```sql
SELECT * FROM produtos WHERE categoria IN ('Eletrônicos',
'Informática');
```

Este comando seleciona produtos das categorias 'Eletrônicos' ou 'Informática'.

32. NOT - Nega uma condição Tradução: NÃO Exemplo:

```sql
SELECT * FROM clientes WHERE NOT cidade = 'São Paulo';
```

Este comando seleciona todos os clientes que não são de São Paulo.

33. IS NULL - Verifica valores nulos Tradução: É NULO Exemplo:

```sql
SELECT nome FROM clientes WHERE telefone IS NULL;
```

Este comando encontra clientes sem número de telefone registrado.

34. IS NOT NULL - Verifica valores não nulos Tradução: NÃO É NULO Exemplo:

```sql
SELECT nome FROM funcionarios WHERE data_demissao IS NOT NULL;
```

Este comando encontra funcionários que já foram demitidos.

35. CASE - Realiza lógica condicional em consultas SQL Tradução: CASO Exemplo:

```sql
SELECT nome,
       CASE
           WHEN salario < 3000 THEN 'Baixo'
           WHEN salario BETWEEN 3000 AND 5000 THEN 'Médio'
           ELSE 'Alto'
       END AS faixa_salarial
FROM funcionarios;
```

Este comando categoriza os funcionários em faixas salariais.

36. COALESCE - Retorna o primeiro valor não nulo em uma lista Tradução: COALESCÊNCIA Exemplo:

```sql
SELECT nome, COALESCE(telefone, email, 'Não informado') AS contato
FROM clientes;
```

Este comando retorna o telefone se disponível, senão o email, e se ambos forem nulos, retorna 'Não informado'.

37. EXISTS - Verifica a existência de linhas em uma subconsulta Tradução: EXISTE Exemplo:

```sql
SELECT nome
FROM clientes c
WHERE EXISTS (SELECT 1 FROM pedidos p WHERE p.cliente_id = c.id);
```

Este comando encontra clientes que fizeram pelo menos um pedido.

38. ANY/SOME - Compara um valor a um conjunto de valores retornados por uma subconsulta Tradução: QUALQUER/ALGUM Exemplo:

```
SELECT nome
FROM produtos
WHERE preco > ANY (SELECT preco FROM produtos WHERE categoria =
'Luxo');
```

Este comando encontra produtos mais caros que pelo menos um produto da categoria 'Luxo'.

39. ALL - Compara um valor a todos os valores retornados por uma subconsulta Tradução: TODOS Exemplo:

```
SELECT nome
FROM funcionarios
WHERE salario > ALL (SELECT AVG(salario) FROM funcionarios GROUP BY
departamento);
```

Este comando encontra funcionários com salário acima da média de todos os departamentos.

40. JOIN - Combina linhas de duas ou mais tabelas com base em uma coluna relacionada Tradução: JUNÇÃO Exemplo:

```
SELECT pedidos.id, clientes.nome
FROM pedidos
JOIN clientes ON pedidos.cliente_id = clientes.id;
```

Este comando combina informações de pedidos com os nomes dos clientes correspondentes.

41. PRIMARY KEY - Define uma coluna ou conjunto de colunas como a chave primária de uma tabela Tradução: CHAVE PRIMÁRIA Exemplo:

```
CREATE TABLE produtos (
    id INT PRIMARY KEY,
    nome VARCHAR(100),
    preco DECIMAL(10,2)
);
```

Este comando cria uma tabela 'produtos' com 'id' como chave primária.

42. FOREIGN KEY - Estabelece uma relação entre duas tabelas Tradução: CHAVE ESTRANGEIRA Exemplo:

sql

```
CREATE TABLE pedidos (
```

```
    id INT PRIMARY KEY,
    cliente_id INT,
    FOREIGN KEY (cliente_id) REFERENCES clientes(id)
);
```

Este comando cria uma tabela 'pedidos' com uma chave estrangeira referenciando a tabela 'clientes'.

43. CONSTRAINT - Impõe regras sobre os dados em uma tabela Tradução: RESTRIÇÃO Exemplo:

```
ALTER TABLE produtos
ADD CONSTRAINT check_preco CHECK (preco > 0);
```

Este comando adiciona uma restrição para garantir que o preço dos produtos seja sempre positivo.

44. INDEX - Melhora o desempenho de consultas SELECT criando índices em colunas Tradução: ÍNDICE (Já explicado anteriormente no comando CREATE INDEX)
45. TRANSACTION - Agrupa múltiplas operações SQL em uma única unidade de trabalho Tradução: TRANSAÇÃO Exemplo:

```
BEGIN TRANSACTION;
UPDATE contas SET saldo = saldo - 100 WHERE id = 1;
UPDATE contas SET saldo = saldo + 100 WHERE id = 2;
COMMIT;
```

Este comando inicia uma transação para transferir 100 unidades de saldo entre duas contas.

46. COMMIT - Salva as alterações feitas durante uma transação Tradução: CONFIRMAR (Exemplo incluído na transação acima)
47. ROLLBACK - Reverte as alterações feitas durante uma transação Tradução: REVERTER Exemplo:

```
BEGIN TRANSACTION;
UPDATE produtos SET preco = preco * 1.1;
ROLLBACK;
```

Este comando inicia uma transação para aumentar os preços em 10%, mas depois cancela a operação.

48. SAVEPOINT - Marca um ponto dentro de uma transação para o qual você pode reverter posteriormente Tradução: PONTO DE SALVAMENTO Exemplo:

```
BEGIN TRANSACTION;
SAVEPOINT antes_aumento;
UPDATE produtos SET preco = preco * 1.1;
ROLLBACK TO antes_aumento;
COMMIT;
```

Este comando cria um ponto de salvamento antes de aumentar os preços, permitindo reverter apenas essa parte da transação se necessário.

49. GRANT - Fornece privilégios a objetos de banco de dados Tradução: CONCEDER Exemplo:

```
GRANT SELECT, INSERT ON tabela_clientes TO usuario_app;
```

Este comando concede permissões de SELECT e INSERT na tabela de clientes ao usuário 'usuario_app'.

50. REVOKE - Remove privilégios de objetos de banco de dados Tradução: REVOGAR Exemplo:

```
REVOKE INSERT ON tabela_clientes FROM usuario_app;
```

Este comando remove a permissão de INSERT na tabela de clientes do usuário 'usuario_app'.

Estes comandos cobrem uma ampla gama de operações em bancos de dados SQL, desde manipulação básica de dados até gerenciamento de transações e controle de acesso.

Filtrando os comandos mais básicos (com exemplo em base livros):

Estes comandos são considerados básicos porque formam o núcleo das operações mais comuns em bancos de dados SQL, permitindo realizar consultas, modificar dados e estruturar informações.

1. SELECT - Seleciona dados do banco de dados Exemplo:

```
SELECT titulo FROM livros;
```

2. FROM - Especifica a tabela da qual estamos extraindo dados Exemplo:

```
SELECT titulo FROM livros;
```

3. WHERE - Filtra a consulta para corresponder a uma condição Exemplo:

```
SELECT titulo FROM livros WHERE ano_publicacao > 2000;
```

4. AS - Renomeia uma coluna ou tabela com um alias Exemplo:

```
SELECT titulo AS nome_do_livro FROM livros;
```

5. JOIN - Combina linhas de duas ou mais tabelas Exemplo:

```sql
SELECT livros.titulo, autores.nome
FROM livros
JOIN autores ON livros.autor_id = autores.id;
```

6. AND - Combina condições de consulta, todas devem ser atendidas Exemplo:

```sql
SELECT titulo FROM livros WHERE ano_publicacao > 2000 AND genero =
'Ficção';
```

7. OR - Combina condições de consulta, pelo menos uma deve ser atendida Exemplo:

```sql
SELECT titulo FROM livros WHERE genero = 'Ficção' OR genero = 'Não-
ficção';
```

8. LIMIT - Limita o número de linhas retornadas Exemplo:

```sql
SELECT titulo FROM livros LIMIT 10;
```

9. IN - Especifica múltiplos valores ao usar WHERE Exemplo:

```sql
SELECT titulo FROM livros WHERE genero IN ('Ficção', 'Fantasia', 'Sci-
Fi');
```

10. CASE - Retorna um valor baseado em uma condição especificada Exemplo:

```sql
SELECT titulo,
       CASE
           WHEN ano_publicacao < 2000 THEN 'Século 20'
           ELSE 'Século 21'
       END AS era
FROM livros;
```

11. IS NULL - Retorna apenas linhas com um valor NULL Exemplo:

```sql
SELECT titulo FROM livros WHERE data_devolucao IS NULL;
```

12. LIKE - Busca por padrões em uma coluna Exemplo:

```sql
SELECT titulo FROM livros WHERE titulo LIKE '%Harry Potter%';
```

13. COMMIT - Escreve a transação no banco de dados Exemplo:

```sql
BEGIN;
```

```
INSERT INTO livros (titulo, autor) VALUES ('1984', 'George Orwell');
COMMIT;
```

14. ROLLBACK - Desfaz um bloco de transação Exemplo:

```
BEGIN;
DELETE FROM livros;
ROLLBACK;
```

15. ALTER TABLE - Adiciona/Remove colunas de uma tabela Exemplo:

```
ALTER TABLE livros ADD COLUMN isbn VARCHAR(13);
```

16. UPDATE - Atualiza dados da tabela Exemplo:

```
UPDATE livros SET copias_disponiveis = copias_disponiveis - 1 WHERE id
= 1;
```

17. CREATE - Cria TABELA, BANCO DE DADOS, ÍNDICE ou VISÃO Exemplo:

```
CREATE TABLE livros (
    id INT PRIMARY KEY,
    titulo VARCHAR(100),
    autor VARCHAR(100)
);
```

18. DELETE - Exclui linhas de uma tabela Exemplo:

```
DELETE FROM livros WHERE id = 1;
```

19. INSERT - Adiciona uma única linha à tabela Exemplo:

```
INSERT INTO livros (titulo, autor) VALUES ('O Senhor dos Anéis',
'J.R.R. Tolkien');
```

20. DROP - Exclui TABELA, BANCO DE DADOS ou ÍNDICE Exemplo:

```
DROP TABLE livros;
```

21. GROUP BY - Agrupa dados em conjuntos lógicos Exemplo:

```
SELECT genero, COUNT(*) as total FROM livros GROUP BY genero;
```

22. ORDER BY - Define a ordem do resultado. Use DESC para ordem reversa Exemplo:

```sql
SELECT titulo FROM livros ORDER BY ano_publicacao DESC;
```

23. HAVING - Semelhante ao WHERE, mas filtra grupos Exemplo:

```sql
SELECT genero, COUNT(*) as total
FROM livros
GROUP BY genero
HAVING COUNT(*) > 10;
```

24. COUNT - Conta o número de linhas Exemplo:

```sql
SELECT COUNT(*) as total_livros FROM livros;
```

25. SUM - Retorna a soma de uma coluna Exemplo:

```sql
SELECT SUM(preco) as receita_total FROM livros;
```

26. AVG - Retorna a média de uma coluna Exemplo:

```sql
SELECT AVG(preco) as preco_medio FROM livros;
```

27. MIN - Retorna o valor mínimo de uma coluna Exemplo:

```sql
SELECT MIN(ano_publicacao) as livro_mais_antigo FROM livros;
```

28. MAX - Retorna o valor máximo de uma coluna Exemplo:

```sql
SELECT MAX(preco) as livro_mais_caro FROM livros;
```

Exemplo completo usando Python e SQL para um sistema de cadastro e edição de um banco de dados de livros:

```python
import sqlite3

# Conectar ao banco de dados (cria se não existir)
conn = sqlite3.connect('biblioteca.db')
cursor = conn.cursor()

# Criar tabela de livros
```

```
cursor.execute('''
CREATE TABLE IF NOT EXISTS livros (
    id INTEGER PRIMARY KEY,
    titulo TEXT NOT NULL,
    autor TEXT NOT NULL,
    ano_publicacao INTEGER,
    genero TEXT,
    preco REAL
)
''')

# Função para adicionar um livro
def adicionar_livro(titulo, autor, ano_publicacao, genero, preco):
    cursor.execute('''
    INSERT INTO livros (titulo, autor, ano_publicacao, genero, preco)
    VALUES (?, ?, ?, ?, ?)
    ''', (titulo, autor, ano_publicacao, genero, preco))
    conn.commit()
    print("Livro adicionado com sucesso!")

# Função para editar um livro
def editar_livro(id, titulo, autor, ano_publicacao, genero, preco):
    cursor.execute('''
    UPDATE livros
    SET titulo = ?, autor = ?, ano_publicacao = ?, genero = ?, preco =
?
    WHERE id = ?
    ''', (titulo, autor, ano_publicacao, genero, preco, id))
    conn.commit()
    print("Livro atualizado com sucesso!")

# Função para listar todos os livros
def listar_livros():
    cursor.execute('SELECT * FROM livros')
    livros = cursor.fetchall()
    for livro in livros:
        print(f"ID: {livro[0]}, Título: {livro[1]}, Autor: {livro[2]},
Ano: {livro[3]}, Gênero: {livro[4]}, Preço: {livro[5]}")

# Exemplo de uso
adicionar_livro("1984", "George Orwell", 1949, "Ficção Distópica",
29.90)
adicionar_livro("O Hobbit", "J.R.R. Tolkien", 1937, "Fantasia", 35.50)

print("\nLista de livros após adição:")
listar_livros()

editar_livro(1, "1984", "George Orwell", 1949, "Ficção Distópica",
32.90)

print("\nLista de livros após edição:")
listar_livros()
```

```
# Fechar a conexão
conn.close()
```

Este exemplo demonstra como usar Python com SQLite para criar um banco de dados simples de livros, adicionar livros, editar informações e listar todos os livros. Ele utiliza vários dos comandos SQL básicos que discutimos, como CREATE TABLE, INSERT, UPDATE, e SELECT.

Mesmo exemplo usando MySQL:

As principais diferenças ao usar MySQL são:

- Uso da biblioteca `mysql-connector-python` em vez de `sqlite3`.
- Necessidade de especificar informações de conexão (host, usuário, senha).
- Sintaxe ligeiramente diferente para alguns comandos SQL.

Aqui está o código completo usando MySQL:

```
import mysql.connector

# Conectar ao banco de dados
conn = mysql.connector.connect(
    host="localhost",
    user="seu_usuario",
    password="sua_senha",
    database="biblioteca"
)
cursor = conn.cursor()

# Criar tabela de livros
cursor.execute('''
CREATE TABLE IF NOT EXISTS livros (
    id INT AUTO_INCREMENT PRIMARY KEY,
    titulo VARCHAR(255) NOT NULL,
    autor VARCHAR(255) NOT NULL,
    ano_publicacao INT,
    genero VARCHAR(100),
    preco DECIMAL(10, 2)
)
''')

# Função para adicionar um livro
def adicionar_livro(titulo, autor, ano_publicacao, genero, preco):
    sql = '''
    INSERT INTO livros (titulo, autor, ano_publicacao, genero, preco)
    VALUES (%s, %s, %s, %s, %s)
    '''
    valores = (titulo, autor, ano_publicacao, genero, preco)
```

```
    cursor.execute(sql, valores)
    conn.commit()
    print("Livro adicionado com sucesso!")

# Função para editar um livro
def editar_livro(id, titulo, autor, ano_publicacao, genero, preco):
    sql = '''
    UPDATE livros
    SET titulo = %s, autor = %s, ano_publicacao = %s, genero = %s,
preco = %s
    WHERE id = %s
    '''
    valores = (titulo, autor, ano_publicacao, genero, preco, id)
    cursor.execute(sql, valores)
    conn.commit()
    print("Livro atualizado com sucesso!")

# Função para listar todos os livros
def listar_livros():
    cursor.execute('SELECT * FROM livros')
    livros = cursor.fetchall()
    for livro in livros:
        print(f"ID: {livro[0]}, Título: {livro[1]}, Autor: {livro[2]},
Ano: {livro[3]}, Gênero: {livro[4]}, Preço: {livro[5]}")

# Exemplo de uso
adicionar_livro("1984", "George Orwell", 1949, "Ficção Distópica",
29.90)
adicionar_livro("O Hobbit", "J.R.R. Tolkien", 1937, "Fantasia", 35.50)

print("\nLista de livros após adição:")
listar_livros()

editar_livro(1, "1984", "George Orwell", 1949, "Ficção Distópica",
32.90)

print("\nLista de livros após edição:")
listar_livros()

# Fechar a conexão
cursor.close()
conn.close()
```

2. Exemplo usando SQLAlchemy (ORM) com MySQL:

SQLAlchemy é um ORM (Object-Relational Mapping) que oferece uma camada de abstração sobre o banco de dados, permitindo trabalhar com objetos Python em vez de SQL direto.

```
from sqlalchemy import create_engine, Column, Integer, String, Float
from sqlalchemy.ext.declarative import declarative_base
```

```python
from sqlalchemy.orm import sessionmaker

# Configuração do banco de dados
engine =
create_engine('mysql+mysqlconnector://seu_usuario:sua_senha@localhost/
biblioteca')
Base = declarative_base()
Session = sessionmaker(bind=engine)

# Definição do modelo
class Livro(Base):
    __tablename__ = 'livros'

    id = Column(Integer, primary_key=True)
    titulo = Column(String(255), nullable=False)
    autor = Column(String(255), nullable=False)
    ano_publicacao = Column(Integer)
    genero = Column(String(100))
    preco = Column(Float)

    def __repr__(self):
        return f"<Livro(titulo='{self.titulo}',
autor='{self.autor}')>"

# Criar tabelas
Base.metadata.create_all(engine)

# Função para adicionar um livro
def adicionar_livro(titulo, autor, ano_publicacao, genero, preco):
    session = Session()
    novo_livro = Livro(titulo=titulo, autor=autor,
ano_publicacao=ano_publicacao, genero=genero, preco=preco)
    session.add(novo_livro)
    session.commit()
    print("Livro adicionado com sucesso!")
    session.close()

# Função para editar um livro
def editar_livro(id, titulo, autor, ano_publicacao, genero, preco):
    session = Session()
    livro = session.query(Livro).filter_by(id=id).first()
    if livro:
        livro.titulo = titulo
        livro.autor = autor
        livro.ano_publicacao = ano_publicacao
        livro.genero = genero
        livro.preco = preco
        session.commit()
        print("Livro atualizado com sucesso!")
    else:
        print("Livro não encontrado.")
    session.close()
```

```python
# Função para listar todos os livros
def listar_livros():
    session = Session()
    livros = session.query(Livro).all()
    for livro in livros:
        print(f"ID: {livro.id}, Título: {livro.titulo}, Autor:
{livro.autor}, "
              f"Ano: {livro.ano_publicacao}, Gênero: {livro.genero},
Preço: {livro.preco}")
    session.close()

# Exemplo de uso
adicionar_livro("1984", "George Orwell", 1949, "Ficção Distópica",
29.90)
adicionar_livro("O Hobbit", "J.R.R. Tolkien", 1937, "Fantasia", 35.50)

print("\nLista de livros após adição:")
listar_livros()

editar_livro(1, "1984", "George Orwell", 1949, "Ficção Distópica",
32.90)

print("\nLista de livros após edição:")
listar_livros()
```

Algumas informações adicionas (documentação) para ajudar no entendimento:

Sistema de Gerenciamento de Biblioteca

Este é um sistema simples de gerenciamento de biblioteca usando Python e SQLAlchemy com MySQL.

Configuração Inicial

1. Instale as dependências necessárias:

```
pip install sqlalchemy mysql-connector-python
```

2. Certifique-se de ter um servidor MySQL rodando e crie um banco de dados chamado "biblioteca".
3. Modifique a linha de conexão do banco de dados no código para refletir suas credenciais:

```
engine =
create_engine('mysql+mysqlconnector://seu_usuario:sua_senha@loca
lhost/biblioteca')
```

Estrutura do Código

Importações e Configuração

- Importamos as classes necessárias do SQLAlchemy.
- Configuramos a conexão com o banco de dados usando `create_engine`.
- Criamos uma classe base `Base` usando `declarative_base()`.
- Configuramos uma `Session` para interagir com o banco de dados.

Definição do Modelo

- A classe `Livro` representa a tabela 'livros' no banco de dados.
- Cada atributo da classe corresponde a uma coluna na tabela.

Funções Principais

1. `adicionar_livro`: Adiciona um novo livro ao banco de dados.
2. `editar_livro`: Atualiza as informações de um livro existente.
3. `listar_livros`: Exibe todos os livros no banco de dados.

Como Usar

1. Para adicionar um livro:

```
adicionar_livro("Título", "Autor", ano, "Gênero", preco)
```

2. Para editar um livro:

```
editar_livro(id, "Novo Título", "Novo Autor", novo_ano, "Novo Gênero", novo_preco)
```

3. Para listar todos os livros:

```
listar_livros()
```

Conceitos Importantes

- **ORM (Object-Relational Mapping)**: Permite trabalhar com banco de dados usando objetos Python em vez de SQL direto.
- **Modelo**: A classe `Livro` define a estrutura da tabela no banco de dados.
- **Sessão**: Usada para realizar operações no banco de dados, como adicionar, atualizar ou consultar dados.

Dicas para Iniciantes

1. Entenda a estrutura básica do código antes de fazer modificações.
2. Experimente adicionar, editar e listar livros para ver como o sistema funciona.
3. Tente adicionar novas funcionalidades, como buscar livros por título ou autor.

4. Lembre-se de sempre fechar as sessões após o uso para liberar recursos.

Este sistema fornece uma base sólida para entender como trabalhar com bancos de dados em Python usando um ORM. À medida que você se familiariza com esses conceitos, pode expandir o sistema com mais funcionalidades e uma interface de usuário mais elaborada.