Aula 08

SQL

Comando Commit

Em um banco de dados SQLite, o comando COMMIT é usado para confirmar as alterações feitas nas tabelas do banco de dados. Quando você faz uma alteração em uma tabela, como uma inserção, atualização ou exclusão, a alteração não é imediatamente gravada no disco, mas é armazenada em uma área temporária chamada transação. O objetivo dessa área é permitir que várias alterações sejam agrupadas e aplicadas de uma só vez, o que pode melhorar o desempenho e garantir a integridade dos dados.

O comando COMMIT é usado para confirmar todas as alterações realizadas na transação e gravar as alterações no disco. Se ocorrer um erro antes do comando COMMIT ser executado, as alterações serão revertidas e o banco de dados retornará ao seu estado anterior.

Aqui está um exemplo de como usar o comando COMMIT ao atualizar um registro na tabela contatos na agenda :

```
C:\Users\augusto.hertzog\Desktop\main.py
# conecta-se ao banco de dados e obtem o cursor
conn = sqlite3.connect('agenda.db')
c = conn.cursor()

# atualiza o telefone do contato com nome 'Alice'
c.execute("UPDATE CONTATO SET telefone = ? WHERE nome = ?", ('555-1234', 'Alice'))

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

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

Observe como chamamos o método commit() após a atualização do registro. Isso garante que a atualização seja confirmada no banco de dados e persista mesmo após o programa ser encerrado.

É importante lembrar que o comando COMMIT deve ser usado com cuidado, pois pode causar perda de dados se usado incorretamente. Em geral, é uma boa prática executar o comando COMMIT apenas depois que todas as alterações na transação forem concluídas e verificadas.

Comando update

Para atualizar um registro na tabela de CONTATO, usamos a cláusula UPDATE em sua consulta SQL:

```
C:\Users\augusto.hertzog\Desktop\main.py
# solicita o ID do contato a ser atualizado
contato_id = int(input("Digite o ID do contato que deseja atualizar: "))

# solicita o novo número de telefone
novo_telefone = input("Digite o novo número de telefone: ")

# executa uma consulta para atualizar o número de telefone do contato especificado pelo usuário
c.execute("UPDATE CONTATO SET telefone = ? WHERE id = ?", (novo_telefone, contato_id))

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

# exibe uma mensagem de confirmação na tela
print("Contato atualizado com sucesso")
```

Neste exemplo, após inserir os contatos na tabela, solicitamos ao usuário que digite o ID do contato que deseja atualizar e o novo número de telefone para este contato.

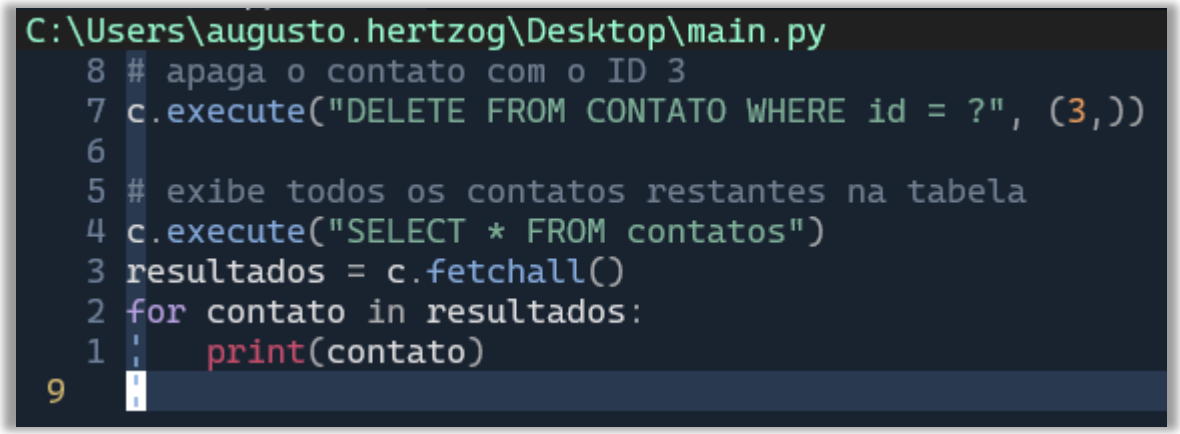
Em seguida, executamos uma consulta SQL UPDATE para atualizar o número de telefone do contato especificado pelo usuário, usando o operador = na cláusula WHERE. Observe que, neste caso, usamos dois sinais de interrogação na chamada de execute() para passar dois parâmetros: o novo número de telefone e o ID do contato.

Finalmente, usamos o método commit() para salvar as alterações no banco de dados e exibimos uma mensagem de confirmação na tela.

Observe que, neste exemplo, apenas o número de telefone do contato é atualizado. Se você quiser permitir que o usuário atualize outros campos, basta modificar a consulta SQL para incluir as colunas e os novos valores correspondentes.

Comando delete

Para apagar um registro da tabela CONTATO no exemplo da agenda, você pode executar uma consulta SQL DELETE que selecione o registro a ser excluído com base em sua chave primária.

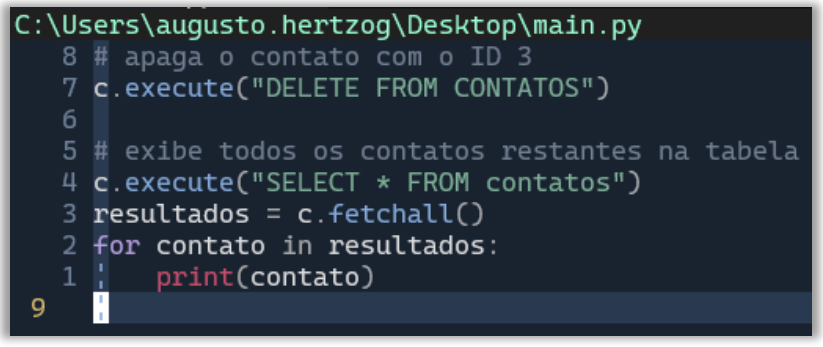


Neste exemplo, executamos uma consulta SQL DELETE que exclui o contato com o ID 3 da tabela CONTATO. Em seguida, executamos uma consulta SQL SELECT para recuperar todos os contatos restantes na tabela e exibimos esses contatos na tela usando um loop for.

Observe que, ao executar a consulta SQL DELETE, usamos um ponto de interrogação (?) como um marcador de posição para o valor do ID do contato a ser excluído. Em seguida, passamos o valor real do ID do contato como um segundo argumento para o método execute() na forma de uma tupla com um único elemento. Isso evita vulnerabilidades de segurança como a injeção de SQL.

Cuidados

Tanto o comando update quanto o comando delete precisam ter MUITO cuidado ao realizar as operações. Isso porque, se não for especificado um filtro com o comando WHERE, todos os registros serão alterados.



O código acima vai apagar TODOS os registros da tabela CONTATO.

Chave Estrangeira e JOIN

Como vimos anteriormente, utilizamos uma Chave Primária (PK) para garantir que os registros na nossa tabela sejam únicos. Além disso, a PK tem outra funcionalidade, que é garantir que os dados entre tabelas sejam íntegros através da Chave Estrangeira (FK).
Usando a agenda, vamos criar duas tabelas, uma CONTATOS que armazena informações básicas de contato (como nome e telefone) e outra tabela ENDERECOS que armazena informações de endereço para cada contato.

Depois, vamos relacionar essas tabelas usando a chave primária de CONTATOS como uma chave estrangeira em ENDERECOS.

Veja a criação das tabelas:

```
C:\Users\augusto.hertzog\Desktop\main.py
 1 import sqlite3
 2
 3 # conecta-se ao banco de dados
 4 conn = sqlite3.connect('agenda.db')
 5
 6 # cria uma tabela para os contatos
 7 c = conn.cursor()
 8 c.execute('''CREATE TABLE CONTATOS
 9              (id INTEGER PRIMARY KEY,
10               nome TEXT,
11               telefone TEXT)''')
12
13 # cria uma tabela para os endereços
14 c.execute('''CREATE TABLE ENDERECOS
15              (id INTEGER PRIMARY KEY,
16               contato_id INTEGER,
17               rua TEXT,
18               cidade TEXT,
19               estado TEXT,
20               cep TEXT,
21               FOREIGN KEY(contato_id) REFERENCES CONTATOS(id))''')
```

Repare que acima temos um campo na tabela ENDERECOS chamado contato_id. Ele é que será usado para criar um relacionamento entre as duas tabelas. Dessa forma, qualquer registro que for inserido na tabela ENDERECO só poderá ser feito se o valor de contato_id existir um registro respectivo na tabela CONTATOS.

Vamos criar agora alguns registros para as novas tabelas:

```
C:\Users\augusto.hertzog\Desktop\main.py
22
23 # insere alguns contatos e endereços na tabela
24 c.execute("INSERT INTO CONTATOS (nome, telefone) VALUES (?, ?)", ('João Silva', '555-1234'))
25 contato_id = c.lastrowid
26 c.execute("INSERT INTO ENDERECOS (contato_id, rua, cidade, estado, cep) VALUES (?, ?, ?, ?, ?)",
   (contato_id, 'Rua A', 'São Paulo', 'SP', '01001-000'))
27
28 c.execute("INSERT INTO CONTATOS (nome, telefone) VALUES (?, ?)", ('Maria Santos', '555-9876'))
29 contato_id = c.lastrowid
30 c.execute("INSERT INTO ENDERECOS (contato_id, rua, cidade, estado, cep) VALUES (?, ?, ?, ?, ?)",
   (contato_id, 'Rua B', 'Rio de Janeiro', 'RJ', '02002-000'))
31
32 c.execute("INSERT INTO CONTATOS (nome, telefone) VALUES (?, ?)", ('Pedro Souza', '555-5555'))
33 contato_id = c.lastrowid
34 c.execute("INSERT INTO ENDERECOS (contato_id, rua, cidade, estado, cep) VALUES (?, ?, ?, ?, ?)",
   (contato_id, 'Rua C', 'Belo Horizonte', 'MG', '03003-000'))
```

E agora, vamos buscar os dados da tabela. Para isso, temos que usar o comando JOIN, onde iremos passar a segunda tabela a ser cruzada com a primeira. Se não especificarmos um critério de comparação entre as duas tabelas, iremos ter um produto cartesiano entre todos os registros.

Colocando um critério de comparação, temos como buscar apenas os registros de um contato que também possuem outros registros em outra tabela. Isso é feito para evitar a redundância de dados.

Veja como ficará a busca e exibição:

```
36 # seleciona todos os contatos com seus respectivos endereços
37 c.execute("SELECT CONTATOS.nome, CONTATOS.telefone, ENDERECOS.rua, ENDERECOS.cidade, ENDERECOS.es
   tado, ENDERECOS.cep FROM CONTATOS JOIN ENDERECOS ON CONTATOS.id = ENDERECOS.contato_id")
38 resultados = c.fetchall()
39
40 # exibe os contatos e endereços na tela
41 for contato in resultados:
42     print(f'Nome: {contato[0]}')
43     print(f'Telefone: {contato[1]}')
44     print(f'Rua: {contato[2]}')
45     print(f'Cidade: {contato[3]}')
46     print(f'Estado: {contato[4]}')
47     print(f'CEP: {contato[5]}')
48     print('')
49
50 # fecha a conexão com o banco de dados
51 conn.close()
```

No comando do JOIN, temos em seguida o comando ON que realiza a comparação dos campos CONTATOS.id com o ENDERECOS.contato_id.

O resultado é uma lista dos contatos e seus respectivos endereços.

Error no SQLite3

O tratamento de erros e exceções é uma parte importante da programação em Python. Quando estamos trabalhando com bancos de dados, é especialmente importante lidar com erros que possam ocorrer ao executar instruções SQL.

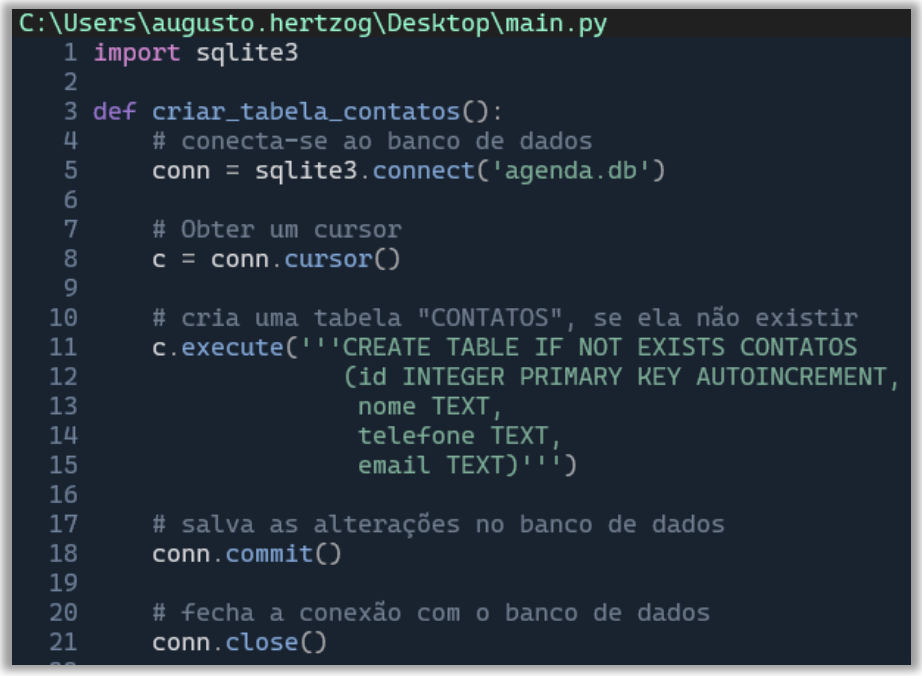
Uma maneira de lidar com erros é usar um bloco try...except em torno de nossas instruções SQL. Isso nos permite capturar exceções e lidar com elas de maneira apropriada. Aqui está um exemplo de como fazer isso com a tabela CONTATOS na agenda:

```
C:\Users\augusto.hertzog\Desktop\main.py
 1 import sqlite3
 2
 3 # conecta-se ao banco de dados e obtém um cursor
 4 conn = sqlite3.connect('agenda.db')
 5 cur = conn.cursor()
 6
 7 # tenta inserir um novo contato e lidar com possíveis erros
 8 try:
 9     # tenta inserir um novo contato na tabela
10     cur.execute("INSERT INTO CONTATOS (id, nome, telefone) VALUES (?, ?, ?)",
   (1, 'Daniel', '555-4321'))
11
12 # capturar a exceção se o registro já existir na tabela
13 except sqlite3.IntegrityError as e:
14     print("Ocorreu um erro:", e)
15
16 # executa este bloco somente se a inserção for bem-sucedida
17 else:
18     print("Novo contato inserido com sucesso!")
19
20 # fecha a conexão com o banco de dados
21 finally:
22     conn.close()
```

Observe como usamos um bloco try...except pontualmente para capturar qualquer exceção que possa ser levantada durante a execução da inserção SQL. Se ocorrer um erro, imprimimos uma mensagem de erro para o usuário. Independentemente de ocorrer um erro ou não, sempre fechamos a conexão com o banco de dados no bloco finally.

Usando Funções

Agora, vamos alterar o código para utilizar funções para nossas conexões :



Repare no comando novo na criação da tabela (IF NOT EXISTS). Ela serve para não precisarmos sempre colocar a criação da tabela em blocos try...except.

Veja a continuação do código abaixo:

```
C:\Users\augusto.hertzog\Desktop\main.py
45 if __name__ == '__main__':
46     # cria a tabela "CONTATOS", se ela não existir
47     criar_tabela_contatos()
48 
49     # insere alguns contatos
50     inserir_contato('Alice', '555-1234', 'alice@example.com')
51     inserir_contato('Bob', '555-5678', 'bob@example.com')
52     inserir_contato('Charlie', '555-9012', 'charlie@example.com')
53 
54     # busca os contatos e imprime os resultados
55     contatos = buscar_contatos()
56     for contato in contatos:
57         print(contato)
```

Repare que o nosso código agora fica muito mais limpo, já que basta chamarmos as funções para realizar as operações que queremos.

Exercícios para Praticar

1. Atualize o campo "Nome" da tabela "Clientes" para "Joana" onde o ID for igual a 1.
2. Atualize o campo "Preço" da tabela "Produtos" para 19.99 onde o ID for igual a 2.
3. Atualize o campo "Cargo" da tabela "Funcionarios" para "Gerente" onde o Salário for maior que 5000.
4. Atualize o campo "Data" da tabela "Eventos" para "2023-12-31" onde o Nome for igual a "Ano Novo".
5. Atualize o campo "Quantidade" da tabela "Produtos" para 0 onde o Nome do produto for "Esgotado".
6. Atualize o campo "Nome" da tabela "Clientes" para "Carlos" onde o Email for igual a "carlos@example.com".
7. Atualize o campo "Salario" da tabela "Funcionarios" para 6000 onde o Cargo for igual a "Analista".
8. Atualize o campo "DataNascimento" da tabela "Pacientes" para "1995-02-20" onde o ID for igual a 1.
9. Atualize o campo "Nome" da tabela "Livros" para "A Revolta dos Anjos" onde o Autor for igual a "Machado de Assis".

10. Delete o registro da tabela "Clientes" onde o ID for igual a 1.
11. Delete todos os registros da tabela "Pedidos" onde o total for menor que 50.
12. Delete os registros da tabela "Produtos" onde o preço for igual a 0.
13. Delete os registros da tabela "Funcionarios" onde o salário for menor que 3000.
14. Delete os registros da tabela "Eventos" onde a data for anterior a "2023-06-01".
15. Delete os registros da tabela "Clientes" onde o email for nulo.
16. Delete os registros da tabela "Livros" onde o autor for igual a "Machado de Assis".
17. Delete os registros da tabela "Pacientes" onde a data de nascimento for posterior a "2000-01-01".
18. Delete os registros da tabela "Veiculos" onde a marca for "Toyota" e o ano for igual a 2022.
19. Delete os registros da tabela "Produtos" onde a quantidade for igual a zero.

Usando a estrutura de tabla CONTATOS e ENDERECOS apresentados anteriormente, resolva os exercícios abaixo:

20. INSERT:
    a. Insira um novo contato na tabela "CONTATOS" com ID 1, nome "João" e telefone "123456789".
    b. Insira um novo endereço na tabela "ENDERECOS" com ID 1, contato_id 1, rua "Rua A", cidade "Cidade A", estado "Estado A" e cep "12345-678".
    c. Insira um novo contato na tabela "CONTATOS" com ID 2, nome "Maria" e telefone "987654321".
    d. Insira um novo endereço na tabela "ENDERECOS" com ID 2, contato_id 2, rua "Rua B", cidade "Cidade B", estado "Estado B" e cep "87654-321".
    e. Insira um novo contato na tabela "CONTATOS" com ID 3, nome "Carlos" e telefone "555555555".
    f. Insira um novo endereço na tabela "ENDERECOS" com ID 3, contato_id 3, rua "Rua C", cidade "Cidade C", estado "Estado C" e cep "33333-333".
21. UPDATE:
    a. Atualize o telefone do contato com ID 1 para "987654321".

    b. Atualize a rua do endereço com ID 1 para "Rua X".
    c. Atualize o nome do contato com ID 2 para "Ana".
    d. Atualize a cidade do endereço com ID 2 para "Cidade Y".
22. DELETE:
    a. Exclua o contato com ID 3 da tabela "CONTATOS".
    b. Exclua o endereço com ID 3 da tabela "ENDERECOS".
23. SELECT com JOIN:
    a. Selecione todos os contatos e seus respectivos endereços.
    b. Selecione o nome do contato e a cidade do endereço para todos os registros.
    c. Selecione o nome do contato, telefone e cep do endereço para todos os registros.
    d. Selecione o nome do contato, telefone e cidade do endereço para contatos que moram em "Cidade A".
    e. Selecione o nome do contato, telefone e estado do endereço para contatos que moram em "Cidade B".
    f. Selecione o nome do contato e o cep do endereço para contatos que moram em "Cidade C".
24. INSERT com JOIN:
    a. Insira um novo contato na tabela "CONTATOS" com nome "Pedro" e telefone "111111111" e um novo endereço relacionado a ele na tabela "ENDERECOS" com rua "Rua D", cidade "Cidade D", estado "Estado D" e cep "44444-444".
25. UPDATE com JOIN:
    a. Atualize o telefone do contato que mora em "Cidade A" para "999999999".
26. DELETE com JOIN:
    a. Exclua o contato e o endereço de todos os registros onde a cidade seja "Cidade B".
27. SELECT com JOIN e condições:
    a. Selecione o nome do contato e o estado do endereço para contatos que moram em "Cidade A" ou "Cidade C".
    b. Selecione o nome do contato e o cep do endereço para contatos que moram em "Cidade A" e "Estado A".
    c. Selecione o nome do contato e o telefone para contatos que não possuem endereço associado.
28. INSERT com valores de outra tabela:
    a. Insira todos os contatos da tabela "CONTATOS" na tabela "CONTATOS_NOVOS" com seus respectivos telefones.
29. UPDATE com base em outra tabela:
    a. Atualize o telefone do contato na tabela "CONTATOS" com base nos dados da tabela "CONTATOS_NOVOS".
30. DELETE com base em outra tabela:
    a. Exclua os contatos da tabela "CONTATOS" com base nos dados da tabela "CONTATOS_NOVOS".
31. SELECT com JOIN em várias tabelas:
    a. Selecione o nome do contato, a rua do endereço e o nome de uma terceira tabela relacionada para todos os registros.
    b. Selecione o nome do contato, a cidade do endereço e a descrição de uma quarta tabela relacionada apenas para contatos com ID maior que 5.
    c. Selecione todos os contatos, seus endereços e todos os campos das tabelas relacionadas.