Senac - Desenvolver Banco de Dados - UC10 - 2024_A6-2/2

SQL - Data Manipulation Language (DML)

Manipulação de Dados (DML - Data Manipulation Language) com SQLite e Python

A Data Manipulation Language (DML) no SQL é usada para inserir, atualizar, excluir e consultar dados nas tabelas de um banco de dados. Ela manipula diretamente o conteúdo sem alterar a estrutura.

1. Declarações DML:

1.1 Consultas e Filtragem

No SQL, consultas são usadas para extrair dados de um banco, enquanto filtragem permite refinar os resultados com base em condições específicas.
A instrução mais comum para consultas é o SELECT. Para obter apenas os dados relevantes, utilizamos cláusulas como WHERE; ORDER BY; GROUP BY; e operadores como IN, BETWEEN, LIKE, e NULL.

1.1.1 SELECT

- **Descrição**:
    - O SELECT é um comando SQL utilizado para **consultar dados** de uma ou mais tabelas no banco de dados. Ele permite filtrar, ordenar e formatar os resultados conforme necessário.
    - No SQL, o comando `SELECT` é usado para **consultar dados** de uma ou mais tabelas.

Exemplo em Python:

```
cursor.execute('SELECT nome, curso FROM alunos')
resultados = cursor.fetchall()

for aluno in resultados:
    print(aluno)
```

1.1.2 WHERE

- **Descrição**:
  - A cláusula WHERE no SQL é usada para **filtrar registros** com base em uma condição específica, retornando apenas os registros que atendem ao critério estabelecido.
  - A cláusula `WHERE` filtra registros com base em uma **condição** específica.

Exemplo em Python:

```
cursor.execute('SELECT * FROM alunos WHERE idade > 20')
print(cursor.fetchall())
```

1.1.3 ORDER BY

- **Descrição**:
  - A cláusula ORDER BY no SQL é usada para **ordenar os resultados** de uma consulta em ordem **crescente** (ASC) ou **decrescente** (DESC) com base em uma ou mais colunas.
  - O `ORDER BY` organiza os resultados em **ordem crescente (ASC)** ou **decrescente (DESC)**.

Exemplo em Python:

```
cursor.execute('SELECT * FROM alunos ORDER BY idade DESC')
print(cursor.fetchall())
```

1.1.4 GROUP BY

- **Descrição**:
  - A cláusula GROUP BY no SQL é usada para **agrupar linhas** que compartilham valores em colunas específicas. Isso permite realizar operações agregadas (como SUM, COUNT, AVG) para cada grupo gerado.
  - A cláusula `GROUP BY` agrupa registros com base em uma ou mais colunas.

Exemplo em Python:

```
cursor.execute('SELECT curso, COUNT(*) FROM alunos GROUP BY curso')
print(cursor.fetchall())
```

1.1.5 HAVING

- **Descrição**:
    - A cláusula HAVING no SQL é usada para **filtrar grupos** gerados por uma consulta com GROUP BY. Diferente de WHERE, que filtra linhas antes da agregação, HAVING aplica filtros **após a agregação**.
    - O `HAVING` filtra grupos de registros após o uso do `GROUP BY`

Exemplo em Python:

```
cursor.execute('SELECT curso, COUNT(*) FROM alunos GROUP BY curso HAVING COUNT(*) > 2')
print(cursor.fetchall())
```

1.1.6 IN

- **Descrição**:
    - A cláusula IN no SQL é usada para **verificar se um valor pertence a uma lista de valores específicos**. Ela simplifica consultas que envolvem múltiplos critérios, funcionando como uma alternativa mais clara ao uso repetido de OR.
    - O operador `IN` verifica se um valor está dentro de uma lista.

Exemplo em Python:

```
cursor.execute('SELECT * FROM alunos WHERE nome IN ("Gustavo", "Henrik")')
print(cursor.fetchall())
```

1.1.7 BETWEEN

- **Descrição**:
    - A cláusula BETWEEN no SQL é usada para **filtrar valores dentro de um intervalo**, incluindo os limites especificados. Ela funciona para dados numéricos, datas e até texto, dependendo da ordenação.
    - O `BETWEEN` filtra registros entre dois valores.

Exemplo em Python:

```
cursor.execute('SELECT * FROM alunos WHERE idade BETWEEN 18 AND 25')
print(cursor.fetchall())
```

1.1.8 LIKE

- **Descrição**:
  - A cláusula LIKE no SQL é usada para filtrar registros que correspondem a um padrão específico em colunas de texto. Ela permite buscas mais flexíveis utilizando coringas.
  - Busca padrões em strings.
  - É útil quando você não conhece o valor exato do dado.
- **Curingas:**
  - `%`: Representa qualquer sequência de caracteres.
    - Exemplo: `'A%'` encontra valores que começam com **"A"**.
  - `_`: Representa **um único caractere**.
    - Exemplo: `'C_t'` encontra **"Cat"**, **"Cut"**.

Exemplo em Python:

```
cursor.execute("SELECT * FROM alunos WHERE nome LIKE 'G%'")
print(cursor.fetchall())
# Encontra alunos cujos nomes começam com "G".

cursor.execute("SELECT * FROM produtos WHERE codigo LIKE 'A_1'")
print(cursor.fetchall())
# Encontra códigos como "A01", "A21", etc.
```

1.1.9 NULL Values

- **Descrição**:
  - No SQL, valores NULL indicam a ausência de valor em uma coluna. Isso é diferente de valores como 0 ou uma string vazia, pois significa que o valor é desconhecido ou não aplicável.
  - Verifica valores nulos.

Exemplo em Python:

```
cursor.execute('SELECT * FROM alunos WHERE email IS NULL')
print(cursor.fetchall())
```

1.1.10 Operadores (AND, OR, NOT)

- **Descrição**:
  - No SQL, os operadores lógicos AND, OR e NOT são usados para combinar condições na cláusula WHERE para realizar filtros mais precisos.
  - **AND**: Todas as condições devem ser verdadeiras.
  - **OR**: Pelo menos uma condição deve ser verdadeira.
  - **NOT**: Inverte a condição.

Exemplo em Python:

```
cursor.execute('SELECT * FROM alunos WHERE idade > 18 AND curso = "Informática"')
print(cursor.fetchall())

cursor.execute('SELECT * FROM alunos WHERE cidade = "Vitória" OR cidade = "Cariacica"')
print(cursor.fetchall())

cursor.execute('SELECT * FROM alunos WHERE idade > NOT cidade = "Vitória"')
print(cursor.fetchall())
```

1.2 Inserção e Atualização de Dados

Em SQL, inserção adiciona novos registros e atualização modifica dados existentes. Essas operações mantêm o banco de dados atualizado e relevante, permitindo também remover registros e copiar dados entre tabelas. Combiná-las com WHERE ajuda a evitar modificações indesejadas e garantir a integridade dos dados. As instruções de inserção e atualização são:

1.2.1 INSERT INTO

- **Descrição**:
  - O comando INSERT INTO no SQL é usado para inserir novos registros em uma tabela. Ele pode especificar todas ou apenas algumas colunas.
  - Insere novos registros.

Exemplo em Python:

```
cursor.execute('INSERT INTO alunos (nome, idade, curso) VALUES (?, ?, ?)', ('Filipe', 18, 'Informática'))
conn.commit()
```

1.2.2 UPDATE

- **Descrição**:
  - O comando **UPDATE** no SQL é usado para **modificar registros existentes** em uma tabela. Ele permite alterar os valores de uma ou mais colunas, com a possibilidade de aplicar filtros usando a cláusula **WHERE**.
  - Atualiza registros existentes.

Exemplo em Python:

```
cursor.execute('UPDATE alunos SET idade = 19 WHERE nome = "Thales"')
conn.commit()
```

1.2.3 DELETE

- **Descrição**:
  - O comando **`DELETE`** no SQL é utilizado para **remover registros** de uma tabela. Pode excluir registros específicos com a cláusula **`WHERE`**, ou todos os registros se o `WHERE` for omitido.
  - Remove registros.

Exemplo em Python:

```
cursor.execute('DELETE FROM alunos WHERE nome = "Paulo Henrique"')
conn.commit()
```

1.2.4 INSERT INTO SELECT

- **Descrição**:
  - O comando **`INSERT INTO SELECT`** no SQL é usado para **copiar dados de uma tabela para outra**. Ele permite inserir registros em uma tabela de destino com base em uma consulta realizada em outra tabela.
  - Copia dados de uma tabela para outra.

Exemplo em Python:

```
cursor.execute('''INSERT INTO ex_alunos (nome, idade) SELECT nome, idade
                  FROM alunos WHERE curso = "Informatica"''')
conn.commit()
```

1.3 Funções

As funções em SQL são utilizadas para realizar operações em dados durante consultas, como agregar valores, manipular strings, lidar com valores nulos e trabalhar com datas e horas.

1.3.1. Funções de Agregação

As funções de agregação são usadas para realizar cálculos sobre um conjunto de registros e retornar um único valor.

1.3.1.1. MIN() e MAX()

- **Descrição**:
    - As funções `MIN()` e `MAX()` no SQL são usadas para encontrar, respectivamente, o **menor** e o **maior valor** em uma coluna.

Exemplo em Python:

```
cursor.execute('SELECT MIN(idade), MAX(idade) FROM alunos')
print(cursor.fetchall())
```

1.3.1.2. COUNT()

- **Descrição**:
    - A função `COUNT()` no SQL é usada para **contar o número de registros** em uma tabela ou o número de valores não nulos em uma coluna específica.

Exemplo em Python:

```
cursor.execute('SELECT COUNT(*) FROM alunos')
print(cursor.fetchone())
```

1.3.1.3. SUM()

- **Descrição**:
    - A função `SUM()` no SQL é usada para **somar valores numéricos** de uma coluna específica.

Exemplo em Python:

```
cursor.execute('SELECT SUM(idade) FROM alunos')
print(cursor.fetchone())
```

1.3.1.4. AVG()

- **Descrição**:
    - A função `AVG()` no SQL é usada para **calcular a média aritmética** dos valores em uma coluna numérica.

Exemplo em Python:

```
cursor.execute('SELECT AVG(idade) FROM alunos')
print(cursor.fetchone())
```

1.3.2. Funções de Manipulação de Strings

Essas funções ajudam a transformar e manipular textos.

1.3.2.1. UPPER()

- **Descrição**

  Converte a string inteira para **maiúsculas**.

Exemplo em Python:

```
cursor.executemany('INSERT INTO alunos (nome) VALUES (?)',
                   [('Gustavo',), ('Thales',), ('Nycollas',)])

cursor.execute('SELECT UPPER(nome) FROM alunos')
print(cursor.fetchall())
```

Saída: [('GUSTAVO',), ('THALES',), ('NYCOLLAS',)]

1.3.2.2. LOWER()

- **Descrição**

  Converte uma string inteira para **minúsculas**.

Exemplo em Python:

```
cursor.executemany('INSERT INTO alunos (nome) VALUES (?)',
                   [('Henrik',), ('Paulo Henrique',), ('Thales',)])

cursor.execute('SELECT LOWER(nome) FROM alunos')
print(cursor.fetchall())
```

Saída: [('henrik',), ('paulo henrique',), ('thales',)]

1.3.2.3. LENGTH()

- **Descrição**

  Retorna o **tamanho de uma string** (número de caracteres).

Exemplo em Python:

```
cursor.execute('INSERT INTO alunos (nome) VALUES ("Thales")')

cursor.execute('SELECT LENGTH(nome) FROM alunos')
print(cursor.fetchone())
```

Saída: (6,)

1.3.3. Funções de Data e Hora

SQLite oferece suporte para trabalhar com **datas e horas** como strings ou timestamps.

1.3.3.1. date()

- **Descrição**

  Retorna a data atual no formato `YYYY-MM-DD`.

Exemplo em Python:

```
cursor.execute('SELECT date("now")')
print(cursor.fetchone())
```

Saída: ('2024-10-21',)

1.3.3.2. datetime()

- **Descrição**

  Retorna a data e hora atual no formato `YYYY-MM-DD HH:MM:SS`.

Exemplo em Python:

```
cursor.execute('SELECT datetime("now")')
print(cursor.fetchone())
```

Saída: ('2024-10-21 15:30:00',)

1.3.3.3. strftime()

- **Descrição**

  Formata a data/hora em um **formato personalizado**.

Exemplo em Python:

```
cursor.execute('SELECT strftime("%d/%m/%Y %H:%M:%S", "now")')
print(cursor.fetchone())
```

Saída: ('21/10/2024 15:30:00',)

1.3.4. Funções de Valores Nulos

Essas funções tratam **registros com valores nulos**, retornando alternativas quando o valor esperado é **NULL**

1.3.4.1. IFNULL(valor, substituto)

- **Descrição**

  Retorna o **valor** se ele não for nulo; caso contrário, retorna o **substituto** fornecido.

Exemplo em Python:

```
cursor.execute('INSERT INTO alunos (nome, email) VALUES ("Filipe", NULL)')

cursor.execute('SELECT nome, IFNULL(email, "Sem email") FROM alunos')
print(cursor.fetchall())  # Saída: [('Filipe', 'Sem email')]
```

1.3.4.2. COALESCE(valor1, valor2, ...)

- **Descrição**

  Retorna o **primeiro valor não nulo** da lista.

Exemplo em Python:

```
cursor.execute('INSERT INTO alunos (nome, email1, email2) VALUES ("Paulo Henrique", NULL, "ph@example.com")')

cursor.execute('SELECT nome, COALESCE(email1, email2, "Sem email") FROM alunos')
print(cursor.fetchall())  # Saída: [('Paulo Henrique', 'ph@example.com')]
```

1.3.5. Função Condicional: CASE

- **Descrição**:
    - A expressão **`CASE`** no SQL é usada para criar instruções condicionais em consultas, semelhante a uma estrutura `if-else`. Ela permite retornar valores diferentes com base em condições especificadas..

Exemplo em Python:

```
cursor.execute('''
SELECT nome,
       CASE
           WHEN email IS NULL THEN 'Email não fornecido'
           ELSE email
       END AS status_email
FROM alunos
''')
print(cursor.fetchall())
```

Resumo das Funções em SQLite

Categoria	Exemplos
Agregação	`COUNT()`, `SUM()`, `AVG()`, `MIN()`, `MAX()`
Manipulação de Strings	`UPPER()`, `LOWER()`, `LENGTH()`
Datas e Horas	`date()`, `datetime()`, `strftime()`
Valores Nulos	`IFNULL()`, `COALESCE()`
Condicional	`CASE`

2. Joins e Combinações

Em SQL, joins e combinações são usados para recuperar dados de múltiplas tabelas, estabelecendo relacionamentos entre elas. Joins combinam registros com base em colunas comuns, possibilitando consultas mais complexas e detalhadas. As combinações também incluem operações como UNION, que mesclam resultados de diferentes consultas em um único conjunto, garantindo uma visão completa dos dados.

2.1 INNER JOIN

- **Descrição**:
  - O `INNER JOIN` no SQL é usado para **combinar registros** de duas ou mais tabelas, retornando apenas os registros que possuem correspondências em ambas. Ele cria uma junção com base em uma **coluna comum** entre as tabelas.
  - Combina registros que existem em ambas as tabelas

Exemplo em Python:

```
cursor.execute('''
SELECT alunos.nome, cursos.nome
FROM alunos
INNER JOIN cursos ON alunos.curso_id = cursos.id
''')
print(cursor.fetchall())
```

2.2 LEFT JOIN

- **Descrição**:
  - O `LEFT JOIN` no SQL retorna **todos os registros da tabela à esquerda** (primeira tabela) e **os registros correspondentes da tabela à direita** (segunda tabela). Se não houver correspondência, os campos da tabela direita serão preenchidos com **NULL**.
  - Inclui todos os registros da tabela da **esquerda**, mesmo que não haja correspondência na tabela da direita.

Exemplo em Python:

```
cursor.execute('''
SELECT alunos.nome, cursos.nome
FROM alunos
LEFT JOIN cursos ON alunos.curso_id = cursos.id
''')
print(cursor.fetchall())
```

2.3 CROSS JOIN

- **Descrição**:
  - O `CROSS JOIN` no SQL é usado para **combinar todas as linhas** de duas tabelas, formando o **produto cartesiano**. O resultado terá o número total de registros equivalente ao **produto do número de linhas** das duas tabelas.
  - Combina todos os registros de ambas as tabelas (produto cartesiano).

Exemplo em Python:

```
cursor.execute('''
SELECT alunos.nome, cursos.nome
FROM alunos
CROSS JOIN cursos
''')
print(cursor.fetchall())
```

2.4 UNION e UNION ALL

- **Descrição**:
  - No SQL, `UNION` e `UNION ALL` combinam os resultados de duas ou mais consultas.
    - `UNION`: Retorna apenas registros **únicos** (sem duplicatas). Eexecuta uma remoção de duplicatas, o que pode ser mais.
    - `UNION ALL`: Retorna **todos os registros**, incluindo duplicatas. É mais rápido, pois não verifica duplicatas
  - Combina conjuntos de resultados.

Exemplo em Python:

```
cursor.execute('SELECT nome FROM alunos UNION SELECT nome FROM professores')
print(cursor.fetchall())
```

Nota: Right Join e Full Join

- **Right Join** e **Full Join** não são suportados diretamente no SQLite, mas podem ser simulados usando **LEFT JOIN** e **UNION**.

Simulando Full Join:

```
cursor.execute('''
SELECT alunos.nome FROM alunos
LEFT JOIN cursos ON alunos.curso_id = cursos.id
UNION
SELECT cursos.nome FROM cursos
LEFT JOIN alunos ON alunos.curso_id = cursos.id
''')
print(cursor.fetchall())
```