UniSENAI

Módulo 3: Manipulação de Dados (DML - Data Manipulation Language)

Módulo 3: Manipulação de Dados (DML - Data Manipulation Language)

Inserção de Dados

A manipulação de dados é um dos pilares fundamentais do SQL (Structured Query Language). Entre as principais operações de DML está a inserção de dados, realizada por meio do comando `INSERT INTO`. Este comando permite adicionar novos registros a uma tabela existente no banco de dados**.**

Comando `INSERT INTO`

O comando `INSERT INTO` é usado para inserir registros em uma tabela. Ele requer que você forneça os valores que deseja adicionar em cada coluna da tabela, correspondendo à ordem definida durante a criação da tabela.

A sintaxe básica do comando é:

```
INSERT INTO nome_da_tabela (coluna1, coluna2, coluna3, ...)
VALUES (valor1, valor2, valor3, ...);
```

- **`nome_da_tabela`**: Nome da tabela onde os dados serão inseridos.
- **`coluna1, coluna2, coluna3, ...`**: Colunas que receberão os valores.
- **`valor1, valor2, valor3, ...`**: Valores a serem adicionados às colunas respectivas.

Exemplo Prático

Imagine uma tabela chamada `clientes` com a seguinte estrutura:

id_cliente	nome	email	cidade
INTEGER	TEXT	TEXT	TEXT

Para inserir um novo cliente, use o comando:

```
INSERT INTO clientes (id_cliente, nome, email, cidade)
VALUES (1, 'João Silva', 'joao.silva@email.com', 'São Paulo');
```

Após a execução, a tabela `clientes` ficará assim:

id_cliente	nome	email	cidade

1	João Silva	joao.silva@email.com	São Paulo

Inserção de Vários Registros Simultaneamente

O SQL permite inserir múltiplos registros em uma única instrução `INSERT INTO`, economizando tempo e melhorando o desempenho em comparação com várias instruções separadas.

A sintaxe é:

```
INSERT INTO nome_da_tabela (coluna1, coluna2, coluna3, ...)
VALUES
    (valor1, valor2, valor3, ...),
    (valor4, valor5, valor6, ...),
    ...;
```

Exemplo Prático

Adicionando múltiplos clientes à tabela `clientes`:

```
INSERT INTO clientes (id_cliente, nome, email, cidade)
VALUES
    (2, 'Maria Souza', 'maria.souza@email.com', 'Rio de Janeiro'),
    (3, 'Carlos Pereira', 'carlos.pereira@email.com', 'Belo Horizonte');
```

Resultado:

id_cliente	nome	email	cidade
1	João Silva	joao.silva@email.com	São Paulo
2	Maria Souza	maria.souza@email.com	Rio de Janeiro
3	Carlos Pereira	carlos.pereira@email.com	Belo Horizonte

Inserção com Valores Padrão (`DEFAULT`)

Em algumas situações, nem todas as colunas de uma tabela precisam receber valores explicitamente. Para esses casos, valores padrão podem ser definidos durante a criação da tabela ou configurados dinamicamente com o uso da palavra-chave `DEFAULT`.

A sintaxe é:

```
INSERT INTO nome_da_tabela (coluna1, coluna2)
VALUES (valor1, DEFAULT);
```

Exemplo Prático

Se a tabela `clientes` possui a coluna `cidade` configurada com um valor padrão como `'Não Informado'`, você pode omitir este valor ao inserir dados:

```
INSERT INTO clientes (id_cliente, nome, email)
VALUES (4, 'Ana Oliveira', 'ana.oliveira@email.com');
```

Resultado:

id_cliente	nome	email	cidade
4	Ana Oliveira	ana.oliveira@email.com	Não Informado

Aplicabilidade Prática

- **Sistemas de Vendas**: Inserir novos pedidos e clientes em tabelas relacionadas.
- **Cadastro de Usuários**: Inserir dados de novos usuários em um sistema.
- **Gerenciamento de Estoque**: Adicionar novos produtos a uma base de dados.

Esses exemplos mostram como o comando `INSERT INTO` pode ser usado para manipular registros de maneira eficiente e adaptável às necessidades do sistema.

Consulta de Dados

Uma das operações mais importantes em SQL é a **consulta de dados**. Utilizando o comando `SELECT`, é possível recuperar informações armazenadas em tabelas, combinando diferentes cláusulas e operadores para obter resultados específicos e bem organizados.

Comando `SELECT` e suas Cláusulas

O comando `SELECT` é utilizado para consultar dados de uma ou mais tabelas no banco de dados. Ele é altamente flexível, permitindo a aplicação de filtros, ordenações, limitações e agrupamentos.

Sintaxe Geral do `SELECT`

```
SELECT colunas
FROM nome_da_tabela
[WHERE condição]
[ORDER BY coluna [ASC | DESC]]
[LIMIT número_registros]
[DISTINCT]
[GROUP BY coluna]
[HAVING condição];
```

Cláusula WHERE: Filtragem de Dados

A cláusula `WHERE` filtra os registros retornados pela consulta, exibindo apenas aqueles que atendem a uma condição específica.

Exemplo Prático

Imagine a tabela `clientes`:

id_cliente	Nome	E-mail	Cidade
1	João Silva	joao@email.com	São Paulo
2	Maria Souza	maria@email.com	Rio de Janeiro
3	Carlos Lima	carlos@email.com	São Paulo

Consulta: Filtrar clientes que residem em São Paulo.

```
SELECT *
FROM clientes
WHERE cidade = 'São Paulo';
```

Resultado:

id_cliente	Nome	E-mail	Cidade
1	João Silva	joao@email.com	São Paulo
3	Carlos Lima	carlos@email.com	São Paulo

Cláusula ORDER BY: Ordenação dos Resultados

A cláusula `ORDER BY` organiza os registros retornados em ordem crescente (`ASC`) ou decrescente (`DESC`).

Exemplo Prático

Consulta: Ordenar clientes por nome em ordem alfabética.

```
SELECT *
FROM clientes
ORDER BY nome ASC;
```

Resultado:

id_cliente	Nome	E-mail	Cidade
3	Carlos Lima	carlos@email.com	São Paulo
1	João Silva	joao@email.com	São Paulo
2	Maria Souza	maria@email.com	Rio de Janeiro

Cláusula LIMIT: Limitação do Número de Registros

A cláusula `LIMIT` restringe o número de registros retornados.

Exemplo Prático

Consulta: Retornar apenas dois registros.

```
SELECT *
FROM clientes
LIMIT 2;
```

Resultado:

id_cliente	Nome	E-mail	Cidade
1	João Silva	joao@email.com	São Paulo
2	Maria Souza	maria@email.com	Rio de Janeiro

Cláusula DISTINCT: Eliminação de Valores Duplicados

A cláusula `DISTINCT` elimina valores duplicados em uma coluna ou combinação de colunas.

Exemplo Prático
Consulta: Listar cidades únicas.

```
SELECT DISTINCT cidade
FROM clientes;
```

Resultado:

Cidade
São Paulo
Joinville

Cláusula GROUP BY: Agrupamento de Dados
A cláusula `GROUP BY` agrupa registros com base em uma ou mais colunas, frequentemente usada com funções agregadas (como `COUNT`, `SUM`, `AVG`).
Exemplo Prático
Consulta: Contar o número de clientes por cidade.

```
SELECT cidade, COUNT(*) AS total_clientes
FROM clientes
GROUP BY cidade;
```

Resultado:

Cidade	total_clientes
São Paulo	2
Joinville	1

Cláusula HAVING: Filtragem de Grupos

A cláusula `HAVING` é usada para filtrar resultados de agrupamentos criados com `GROUP BY`.

Exemplo Prático

Consulta: Retornar cidades com mais de um cliente.

```
SELECT cidade, COUNT(*) AS total_clientes
FROM clientes
GROUP BY cidade
HAVING COUNT(*) > 1;
```

Resultado:

Cidade	total_clientes
São Paulo	2

Operadores de Comparação e Lógicos

Os operadores permitem realizar comparações e combinar condições.

Operadores de Comparação

- **=: Igualdade**

```
SELECT * FROM clientes WHERE cidade = 'São Paulo';
```

- **> e <**: Maior ou menor que

```
SELECT * FROM clientes WHERE id_cliente > 1;
```

- **BETWEEN**: Faixa de valores

```
SELECT * FROM clientes WHERE id_cliente BETWEEN 1 AND 2;
```

- **IN**: Valores em uma lista

```
SELECT * FROM clientes WHERE cidade IN ('São Paulo', 'Rio de Janeiro');
```

- **LIKE**: Pesquisa por padrão (usando % como coringa)

```
SELECT * FROM clientes WHERE nome LIKE 'M%';
```

Operadores Lógicos

- **AND**: Combina condições (todas devem ser verdadeiras)

```
SELECT * FROM clientes WHERE cidade = 'São Paulo' AND id_cliente > 1;
```

- **OR**: Combina condições (uma deve ser verdadeira)

```
SELECT * FROM clientes WHERE cidade = 'São Paulo' OR cidade = 'Rio de Janeiro';
```

- **NOT**: Exclui resultados que atendem à condição

```
SELECT * FROM clientes WHERE NOT cidade = 'Rio de Janeiro';
```

Aplicabilidade Prática

- **Análise de Dados**: Identificar tendências e realizar agrupamentos úteis, como vendas por região ou períodos de pico.
- **Filtragem de Informações**: Encontrar registros específicos, como clientes de uma cidade ou faixa de idade.
- **Relatórios Dinâmicos**: Criar relatórios organizados e úteis para decisões estratégicas.

Essa abordagem consolidada e prática do `SELECT` e suas cláusulas é fundamental para consultas eficientes e análise de dados robusta.

3. Atualização de Dados

A **atualização de dados** é fundamental para manter as informações armazenadas no banco de dados consistentes e atualizadas. O comando SQL utilizado para modificar os dados existentes é o **`UPDATE`**. Ele permite alterar um ou mais campos em uma tabela com base em condições específicas ou atualizar todos os registros.

Comando `UPDATE`

O comando `UPDATE` é usado para alterar os valores de uma ou mais colunas em registros existentes.

Sintaxe Geral

```
UPDATE nome_da_tabela
SET coluna1 = valor1, coluna2 = valor2, ...
[WHERE condição];
```

`nome_da_tabela`: Nome da tabela onde os dados serão atualizados.
`SET`: Define os campos que serão alterados e os valores que eles receberão.
`WHERE`: (Opcional) Filtra os registros a serem atualizados. Sem o `WHERE`, **todos os registros da tabela serão atualizados.**

Atualização de Dados com Condições (`WHERE`)

Para evitar alterações indesejadas, o uso da cláusula `WHERE` é essencial. Ela permite especificar quais registros devem ser atualizados com base em uma ou mais condições.

Exemplo Prático

Imagine a tabela `clientes`:

id_cliente	Nome	E-mail	Cidade
1	João Silva	joao@email.com	São Paulo
2	Maria Souza	maria@email.com	Rio de Janeiro
3	Carlos Lima	carlos@email.com	São Paulo

Consulta: Atualizar o email de João Silva.

```
UPDATE clientes
SET email = 'joao.silva@email.com'
WHERE id_cliente = 1;
```

Resultado:

id_cliente	Nome	E-mail	Cidade
1	João Silva	joao.silva@email.com	São Paulo
2	Maria Souza	maria@email.com	Rio de Janeiro
3	Carlos Lima	carlos@email.com	São Paulo

Atualização de Vários Registros Simultaneamente

O `UPDATE` pode alterar vários registros ao mesmo tempo, dependendo da condição especificada.

Exemplo Prático

Consulta: Atualizar a cidade de todos os clientes que estão em São Paulo para "Campinas".

```
UPDATE clientes
SET cidade = 'Campinas'
WHERE cidade = 'São Paulo';
```

Resultado:

id_cliente	Nome	E-mail	Cidade
1	João Silva	joao.silva@email.com	Campinas

2	Maria Souza	maria@email.com	Rio de Janeiro
3	Carlos Lima	carlos@email.com	Campinas

4. Exclusão de Dados

A exclusão de dados é outro aspecto essencial na manipulação de informações em bancos de dados. Utilizamos o comando `DELETE` para remover registros específicos. Em alguns casos, o comando `TRUNCATE` pode ser usado para excluir todos os registros de uma tabela de forma mais eficiente.

Comando `DELETE`

O comando `DELETE` remove registros de uma tabela com base em condições específicas. Sem a cláusula `WHERE`, todos os registros da tabela serão excluídos.

Sintaxe Geral

```
DELETE FROM nome_da_tabela
[WHERE condição];
```

- **`nome_da_tabela`: Nome da tabela de onde os registros serão excluídos.**
- **`WHERE`: (Opcional) Define os critérios para exclusão de registros.**

Exclusão Condicionada (`WHERE`)

A exclusão condicionada permite que apenas os registros que atendem a uma condição sejam removidos.

Exemplo Prático

Consulta: Remover o cliente Maria Souza.

```
DELETE FROM clientes
WHERE id_cliente = 2;
```

Resultado:

id_cliente	Nome	E-mail	Cidade
1	João Silva	joao.silva@email.com	Campinas

3	Carlos Lima	carlos@email.com	Campinas

Diferença entre `DELETE` e `TRUNCATE`

Aspecto	DELETE	TRUNCATE
Finalidade	Remove registros específicos.	Remove todos os registros da tabela.
Cláusula WHERE	Suporta o uso da cláusula WHERE.	Não suporta condições (WHERE).
Velocidade	Mais lento devido ao registro de transações.	Mais rápido, pois não registra transações.
Restauração	Sujeito a ROLLBACK (se dentro de transação).	Não pode ser restaurado via ROLLBACK.
Estrutura	Mantém a estrutura e os índices da tabela.	Mantém a estrutura, mas redefine contadores de auto incremento.

Exemplo Prático

Comando `DELETE`: Remove todos os clientes de Campinas.

```
DELETE FROM clientes
WHERE cidade = 'Campinas';
```

Comando `TRUNCATE`: Remove todos os registros da tabela `clientes`.

```
TRUNCATE TABLE clientes;
```

Aplicabilidade Prática

- Atualização de Dados: Corrigir erros em registros, alterar informações desatualizadas ou ajustar múltiplos registros em massa.
  Exemplo: Atualizar preços de produtos ou status de pedidos.
- Exclusão de Dados: Remover registros antigos, inválidos ou dados que não são mais necessários.
  Exemplo: Deletar usuários inativos ou limpar logs antigos de sistema.

Com o domínio das operações de atualização e exclusão, é possível manter a integridade e a relevância das informações armazenadas no banco de dados.

Vamos Praticar

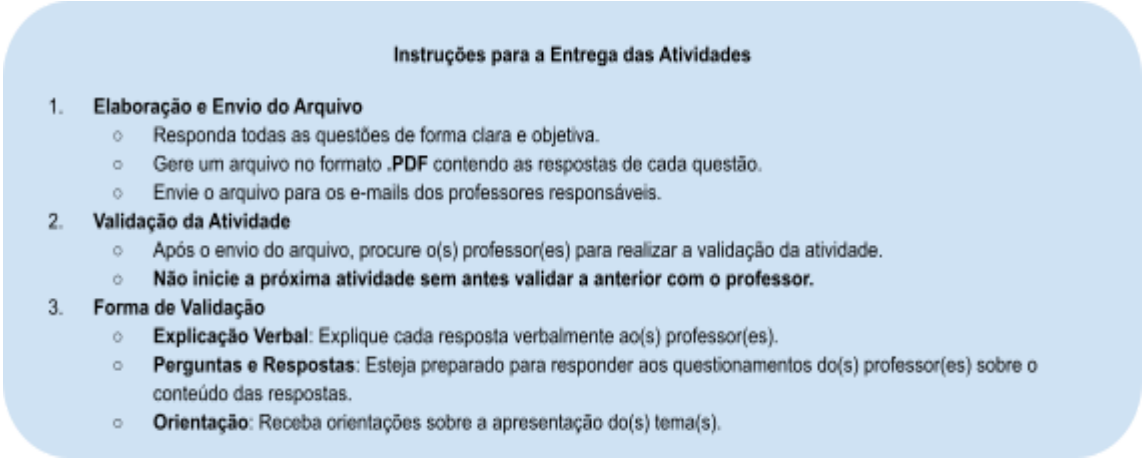
Questões Práticas

1. **Inserção de Dados**
   - Crie uma tabela chamada `funcionarios` com as colunas: `id_funcionario` (INT, chave primária), `nome` (VARCHAR), `email` (VARCHAR) e `cargo` (VARCHAR).
   - Insira três registros nesta tabela usando o comando `INSERT INTO`.
2. **Inserção de Múltiplos Registros**
   - Na tabela `funcionarios`, insira de uma só vez cinco registros adicionais com dados fictícios.
3. **Uso de Valores Padrão**
   - Adicione uma coluna `cidade` à tabela `funcionarios` com o valor padrão "Não Informado".
   - Insira um registro omitindo o valor da coluna `cidade` e verifique o resultado.
4. **Consulta com Filtros (WHERE)**
   - Crie uma consulta para selecionar todos os funcionários cujo cargo seja "Analista".
5. **Consulta com Ordenação (ORDER BY)**

    - Liste todos os funcionários ordenados por nome em ordem alfabética.
6. **Limitação de Resultados (LIMIT)**
    - Exiba os três primeiros registros da tabela `funcionarios`.
7. **Uso de DISTINCT**
    - Liste todas as cidades únicas presentes na tabela `funcionarios`.
8. **Agrupamento de Dados (GROUP BY)**
    - Agrupe os funcionários por cargo e conte quantos funcionários existem em cada cargo.
9. **Atualização de Dados**
    - Atualize o cargo de todos os funcionários com o nome "João" para "Coordenador".
10. **Exclusão de Dados**
    - Remova todos os funcionários que tenham "Analista" como cargo.

Questões Teóricas

1. Explique a diferença entre `INSERT INTO` e `UPDATE`. Em que situações cada um deve ser usado?
2. Qual a função do comando `DELETE` e como ele se diferencia do comando `TRUNCATE`?
3. Por que é importante usar a cláusula `WHERE` em comandos como `UPDATE` e `DELETE`?
4. Descreva a utilidade da cláusula `ORDER BY` e como ela pode ser aplicada em consultas.
5. O que significa a palavra-chave `DEFAULT` no contexto de inserção de dados? Dê um exemplo de uso.
6. Qual a finalidade da cláusula `DISTINCT` e em que situações ela é útil?
7. Explique como a cláusula `GROUP BY` funciona e cite um exemplo prático de sua aplicação.
8. Qual é o impacto de não usar índices no banco de dados durante consultas que utilizam filtros (`WHERE`)?
9. Cite dois operadores de comparação e dois operadores lógicos em SQL, explicando como funcionam.
10. Qual a diferença entre `HAVING` e `WHERE`? Por que o `HAVING` é usado em conjunto com o `GROUP BY`?



Instruções para a Entrega das Atividades

1. **Elaboração e Envio do Arquivo**
    - Responda todas as questões de forma clara e objetiva.
    - Gere um arquivo no formato **.PDF** contendo as respostas de cada questão.
    - Envie o arquivo para os e-mails dos professores responsáveis.
2. **Validação da Atividade**
    - Após o envio do arquivo, procure o(s) professor(es) para realizar a validação da atividade.
    - **Não inicie a próxima atividade sem antes validar a anterior com o professor.**
3. **Forma de Validação**
    - **Explicação Verbal**: Explique cada resposta verbalmente ao(s) professor(es).
    - **Perguntas e Respostas**: Esteja preparado para responder aos questionamentos do(s) professor(es) sobre o conteúdo das respostas.
    - **Orientação**: Receba orientações sobre a apresentação do(s) tema(s).