



INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

AULA 12 - SQL III (CRUD)

O QUE IREMOS APRENDER

- 01** RESUMO DA AULA PASSADA
- 02** CONTEXTUALIZAÇÃO DA AULA DE HOJE
- 03** CONSULTAS SQL
- 04** RELACIONAMENTOS SQL
- 05** MÃOS NO CÓDIGO
- 06** SQLITE: CONCEITOS E COMANDOS BÁSICOS

Resumo da aula passada

A manipulação de dados e comandos SQL são fundamentais para trabalhar com bancos de dados e realizar tarefas como inserção, atualização, consulta e exclusão de informações.

Manipulação de Dados:

- Inserção (INSERT): Adicionar novos registros a uma tabela.
- Atualização (UPDATE): Modificar dados existentes na tabela.
- Consulta (SELECT): Recuperar dados da tabela.
- Exclusão (DELETE): Remover registros da tabela.

Resumo da aula passada

Comandos SQL:

- CREATE TABLE: Criar uma nova tabela.
- ALTER TABLE: Modificar a estrutura de uma tabela existente.
- DROP TABLE: Remover uma tabela e seus dados.
- JOIN: Combinar dados de várias tabelas relacionadas.
- GROUP BY: Agrupar dados com funções de agregação.
- ORDER BY: Ordenar resultados.
- WHERE: Filtrar dados com base em condições.



Resumo da aula passada

DISTINCT: Retornar valores únicos.

LIMIT: Limitar o número de resultados em uma consulta.

Esses são conceitos básicos de manipulação de dados e comandos SQL, que são amplamente utilizados na gestão de bancos de dados relacionais, como MySQL, PostgreSQL, entre outros. SQL é uma linguagem poderosa para lidar com informações armazenadas em bancos de dados.

Contextualização da aula de hoje

RELACIONAMENTOS:

- Em SQL, um relacionamento se refere à forma como as tabelas em um banco de dados estão associadas ou conectadas. Os relacionamentos são fundamentais para organizar dados e permitir que você consulte informações de maneira eficaz através de várias tabelas.

CONSULTAS:

- Já as consultas são comandos que você utiliza para interagir com um banco de dados relacional e recuperar, atualizar, inserir ou excluir informações nele. Essas consultas permitem que você acesse dados de uma ou mais tabelas em um banco de dados e realizem diversas operações. Vimos alguns exemplos de consultas na aula passada e hoje daremos continuidade ao assunto.

Consultas SQL - WHERE

A cláusula WHERE em SQL é como uma "peneira" que você usa em suas consultas para filtrar os registros que atendem a determinadas condições. Você especifica essas condições após o WHERE e somente os registros que satisfazem essas condições serão incluídos nos resultados da consulta. Isso permite que você busque apenas os dados que são relevantes para o que você precisa, em vez de retornar todos os registros da tabela.



```
1 SELECT nome, cidade FROM clientes WHERE cidade = "São Paulo";
```

Consultas SQL - AS

A palavra-chave **AS** é usada em consultas SQL para renomear colunas, tabelas ou expressões temporárias com um alias (apelido) mais amigável ou descritivo. Ela é frequentemente utilizada para tornar os resultados da consulta mais legíveis ou para referenciar colunas calculadas por meio de expressões.



```
1 SELECT nome_cliente as Nome, email AS Endereco_de_Email FROM clientes
```

Consultas SQL - GROUP BY

A cláusula **GROUP BY** é usada em consultas SQL para agrupar linhas de dados com base em valores em uma ou mais colunas e aplicar funções de agregação a esses grupos. Ela é comumente usada em conjunto com funções de agregação, como SUM, COUNT, MAX, MIN, entre outras.



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

Consultas SQL - ORDER BY

A cláusula `ORDER BY` é usada em consultas SQL para classificar os resultados de uma consulta em ordem ascendente (ASC) ou descendente (DESC) com base em uma ou mais colunas. Essa cláusula é frequentemente usada para organizar os resultados da consulta de maneira significativa.



```
1 SELECT nome, data, valor FROM transacoes ORDER BY data ASC, valor DESC;
```

Consultas SQL - INNER JOIN

A cláusula `INNER JOIN` é usada em consultas SQL para combinar dados de duas ou mais tabelas com base em colunas relacionadas. Ela retorna apenas os registros que têm correspondências nas tabelas sendo unidas. O `INNER JOIN` é uma das operações de junção mais comuns e é usado para recuperar dados relacionados de várias tabelas.



```
1 SELECT clientes.nome, pedidos.numero_pedido, pedidos.data_pedido FROM clientes
2 INNER JOIN pedidos ON clientes.cliente_id = pedidos.cliente_id;
```

Consultas SQL - LEFT JOIN

A cláusula `LEFT JOIN`, é usada em consultas SQL para combinar dados de duas ou mais tabelas com base em colunas relacionadas, mas com uma característica importante: ela retorna todos os registros da tabela da esquerda (ou a primeira tabela na cláusula `LEFT JOIN`) e os registros correspondentes da tabela da direita (ou a segunda tabela na cláusula `LEFT JOIN`). Se não houver correspondência na tabela da direita, as colunas da tabela da direita serão preenchidas com valores nulos.



```
1 SELECT departamentos.nome AS Departamento, funcionarios.nome AS Funcionario
2 FROM departamentos
3 LEFT JOIN funcionarios ON departamentos.departamento_id = funcionarios.departamento_id;
```

Consultas SQL - LEFT JOIN

- Departamentos e funcionarios são os nomes das tabelas.
- Departamentos.departamento_id e funcionarios.departamento_id são as colunas que estão sendo usadas para relacionar as duas tabelas.
- SELECT especifica as colunas que você deseja recuperar dos resultados da consulta.



```
1 SELECT departamentos.nome AS Departamento, funcionarios.nome AS Funcionario
2 FROM departamentos
3 LEFT JOIN funcionarios ON departamentos.departamento_id = funcionarios.departamento_id;
```

Consultas SQL - RIGHT JOIN

A cláusula RIGHT JOIN, é usada em consultas SQL para combinar dados de duas ou mais tabelas com base em colunas relacionadas.

No entanto, o RIGHT JOIN opera de forma oposta ao LEFT JOIN. Enquanto o LEFT JOIN retorna todos os registros da tabela da esquerda (ou a primeira tabela na cláusula LEFT JOIN) e os registros correspondentes da tabela da direita, o RIGHT JOIN retorna todos os registros da tabela da direita e os registros correspondentes da tabela da esquerda.

Consultas SQL - RIGHT JOIN

- departamentos e funcionarios são os nomes das tabelas.
- departamentos.departamento_id e funcionarios.departamento_id são as colunas que estão sendo usadas para relacionar as duas tabelas.
- SELECT especifica as colunas que você deseja recuperar dos resultados da consulta.



```
1 SELECT departamentos.nome AS Departamento, funcionarios.nome AS Funcionario
2 FROM departamentos
3 RIGHT JOIN funcionarios ON departamentos.departamento_id = funcionarios.departamento_id;
```

Consultas SQL - FULL OUTER JOIN

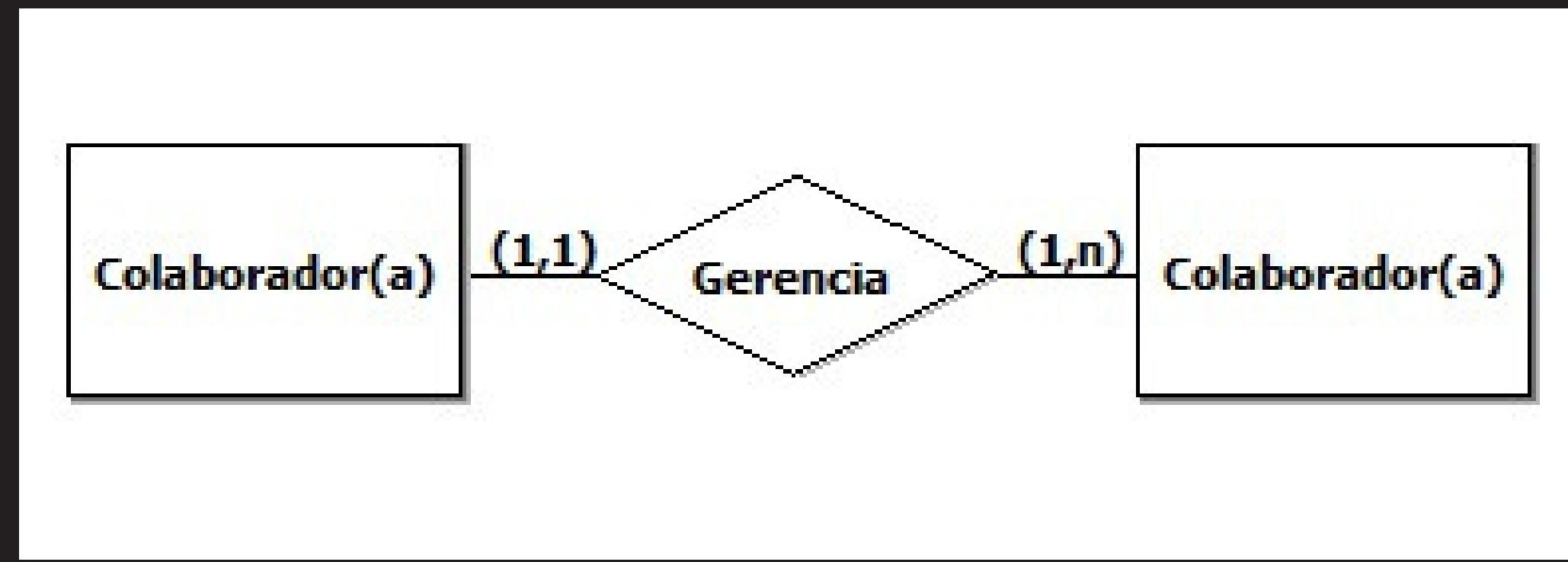
A cláusula **FULL OUTER JOIN** é usada em consultas SQL para combinar dados de duas ou mais tabelas com base em colunas relacionadas. Ela retorna todos os registros de ambas as tabelas, independentemente de haver correspondências. Se não houver correspondência em uma tabela, as colunas dessa tabela serão preenchidas com valores nulos.



```
1 SELECT departamentos.nome AS Departamento, funcionarios.nome AS Funcionario FROM departamentos
2 FULL OUTER JOIN funcionarios ON departamentos.departamento_id = funcionarios.departamento_id;
```

Relacionamentos SQL

Em bancos de dados SQL, os relacionamentos são usados para estabelecer conexões entre tabelas por meio de chaves. Os relacionamentos são fundamentais para a organização de dados e permitem que você consulte e recupere informações de várias tabelas.



Relacionamentos SQL - UM PARA UM

Um relacionamento um-para-um (1:1) em bancos de dados significa que um registro em uma tabela está diretamente relacionado a um único registro em outra tabela e vice-versa. Cada registro na primeira tabela tem um correspondente único na segunda tabela. Isso é usado quando você deseja dividir informações relacionadas em tabelas separadas para manter a integridade dos dados e reduzir a duplicação de informações.

Relacionamentos SQL - UM PARA UM

Um exemplo prático de um relacionamento um-para-um é a relação entre a tabela "Pessoas" e a tabela "Documentos de Identidade":

Tabela "Pessoas":

- pessoa_id (Chave Primária)
- nome
- data_nascimento

Tabela "Documentos de Identidade":

- documento_id (Chave Primária)
- numero_documento
- tipo_documento
- pessoa_id (Chave Estrangeira, relacionando-se à tabela "Pessoas")

Neste cenário:

- Cada pessoa pode ter apenas um documento de identidade.
- Cada registro na tabela "Pessoas" está diretamente relacionado a um único registro na tabela "Documentos de Identidade" por meio da chave estrangeira "pessoa_id".

Relacionamentos SQL - UM PARA MUITOS

Um relacionamento um-para-muitos (1:N) em bancos de dados SQL significa que um registro em uma tabela está relacionado a vários registros em outra tabela, enquanto cada registro na segunda tabela está associado a apenas um registro na primeira tabela. Isso é frequentemente usado quando você tem uma entidade principal que pode ter múltiplas entidades secundárias relacionadas a ela.

Vamos ver um exemplo prático de um relacionamento um-para-muitos entre as tabelas "Departamentos" e "Funcionários":

Relacionamentos SQL - UM PARA MUITOS

Vamos ver um exemplo prático de um relacionamento um-para-muitos entre as tabelas "Departamentos" e "Funcionários":

Tabela "Departamentos":

- departamento_id (Chave Primária)
- nome_departamento
- localizacao

Tabela "Funcionários":

- funcionario_id (Chave Primária)
- nome_funcionario
- salario
- departamento_id (Chave Estrangeira, relacionando-se à tabela "Departamentos")

Neste cenário:

- Cada departamento pode ter vários funcionários.
- Cada registro na tabela "Funcionários" está associado a um único departamento por meio da chave estrangeira "departamento_id".

Relacionamentos SQL - MUITOS PARA MUITOS

Um relacionamento muitos-para-muitos (N:M) em bancos de dados SQL ocorre quando múltiplos registros em uma tabela podem estar relacionados a múltiplos registros em outra tabela. Para representar esse tipo de relacionamento, você normalmente usa uma tabela intermediária (também conhecida como tabela de junção) que associa registros de ambas as tabelas. O relacionamento muitos- para-muitos é comumente usado para modelar associações complexas entre entidades em um banco de dados.

Relacionamentos SQL - UM PARA UM

Vamos ver um exemplo prático de um relacionamento muitos-para-muitos entre as tabelas "Alunos" e "Cursos":

Tabela "Alunos":

- aluno_id (Chave Primária)
- nome_aluno

Tabela "Cursos":

- curso_id (Chave Primária)
- nome_curso

Tabela de Junção "Matrículas":

- matricula_id (Chave Primária)
- aluno_id (Chave Estrangeira
relacionando-se à tabela "Alunos")
- curso_id (Chave Estrangeira
relacionando-se à tabela "Cursos")

Neste cenário:

- Cada aluno pode estar matriculado em vários cursos.
- Cada curso pode ter vários alunos matriculados.

Introdução

Olá, aventureiros do mundo dos dados! Até agora, vocês embarcaram em uma jornada incrível, explorando os mistérios dos bancos de dados através do **Workbench** e do **SQLite Studio**.

Foi uma experiência empolgante ver como as informações podem ser manipuladas e organizadas com apenas alguns cliques e comandos, não é mesmo? Mas agora, preparem-se para elevar essa aventura a um novo nível!

Vocês vão aprender a utilizar o poder do Python para interagir com os dados de uma forma totalmente nova e dinâmica. Com alguns comandos simples e claros, vocês terão o controle do banco de dados na ponta dos dedos, diretamente do seu script Python.

Então, estão prontos para descobrir como transformar linhas de código em poderosas ferramentas de manipulação de dados?

Vamos juntos nesta nova jornada de aprendizado e descobrir alguns comandos.



SQLITE: Conceitos e comandos básicos

Comando: `sqlite3.connect()`

- **Uso:** Estabelece uma conexão com um banco de dados SQLite. Se o arquivo não existir, ele será criado.
- **Detalhe:** É o primeiro passo para trabalhar com SQLite em Python, Garante que você tenha uma conexão com seu banco de dados.
- **Exemplo:**



```
1 import sqlite3
2
3 conn = sqlite3.connect('meu_banco.db')
4
5 # Aqui, 'meu_banco.db' é o nome do arquivo do banco de dados.
6 # Estabelece uma conexão com um banco de dados SQLite.
7 # Se o arquivo não existir, ele será criado.
```

SQLITE: Conceitos e comandos básicos

Comando: `conn.cursor()`

- **Uso:** Cria um objeto cursor, que permite executar comandos SQL no banco de dados.
- **Detalhe:** O cursor é essencial para a execução de todas as operações SQL, como consultas, inserções, atualizações e exclusões.
- **Exemplo:**



```
1 import sqlite3
2
3 cursor = conn.cursor()
4
5 # Após criar o cursor, você pode usá-lo para executar comandos SQL.
```

SQLITE: Conceitos e comandos básicos

Comando: cursor.execute()

- Uso: Executa um único comando SQL.
- Detalhe: Pode ser usado para criar tabelas, inserir dados, ou executar qualquer outro comando SQL.
- Exemplo:



```
1 import sqlite3  
2  
3 cursor.execute("CREATE TABLE usuarios (id INTEGER, nome TEXT)")
```

SQLITE: Conceitos e comandos básicos

Comando: cursor.fetchall()

- Uso: O fetchall recupera todas as linhas de uma consulta SQL
- Detalhe: Usado após executar uma consulta SELECT para obter os dados.
- Exemplo:



```
1 import sqlite3  
2  
3 cursor.execute("SELECT * FROM usuarios");  
4 rows = cursor.fetchall()
```

SQLITE: Conceitos e comandos básicos

Comando: conn.commit()

- **Uso:** Salva as alterações feitas no banco de dados.
- **Detalhe:** Essencial após comandos INSERT, UPDATE, DELETE para garantir que as alterações sejam aplicadas.
- **Exemplo:**

```
● ● ●  
1 conn = sqlite3.connect('exemplo.db') # Estabelecer conexão com o banco de dados  
2  
3 cursor = conn.cursor() # Criar um cursor  
4  
5 cursor.execute("CREATE TABLE exemplo (id INTEGER PRIMARY KEY, nome TEXT)") # Executar um comando SQL para criar uma tabela  
6  
7 cursor.execute("INSERT INTO exemplo (nome) VALUES ('Teste')") # Inserir dados na tabela  
8  
9 conn.commit() # Salvar as alterações feitas no banco de dados
```

SQLITE: Conceitos e comandos básicos

Comando: `conn.close()`

- Uso: Fecha a conexão com o banco de dados.
- Detalhe: Importante para liberar recursos. Deve ser feito após terminar as operações com o banco de dados.
- Exemplo:

```
● ● ●  
1 import sqlite3  
2  
3 conn = sqlite3.connect('exemplo.db') # Estabelecer conexão com o banco de dados  
4  
5 cursor = conn.cursor() # Criar um cursor  
6  
7 cursor.execute("CREATE TABLE exemplo (id INTEGER PRIMARY KEY, nome TEXT)") # Executar um comando SQL para criar uma tabela  
8  
9 cursor.execute("INSERT INTO exemplo (nome) VALUES ('Teste')") # Inserir dados na tabela  
10  
11 conn.commit() # Salvar as alterações feitas no banco de dados  
12  
13 conn.close() # Encerrando a conexão
```

ATIVIDADE PRÁTICA 1

Suponha que você tenha uma tabela "clientes" e uma tabela "pedidos". Escreva uma consulta SQL que retorne o nome do cliente e o número do pedido para todos os clientes, incluindo aqueles que não fizeram nenhum pedido. Utilizando as mesmas tabelas de "clientes" e "pedidos", escreva uma consulta SQL que retorne o nome do cliente e o número do pedido para todos os pedidos, incluindo os pedidos que não estão associados a nenhum cliente.

ATIVIDADE PRÁTICA 2

Suponha que você tenha uma tabela "vendedores" e uma tabela "vendas". Escreva uma consulta SQL que retorne o nome do vendedor e o valor da venda para todas as vendas e todos os vendedores, incluindo os vendedores que não fizeram nenhuma venda e as vendas não associadas a nenhum vendedor.

ATIVIDADE PRÁTICA 3

Suponha que você tenha tabelas "Pessoas" e "Documentos de Identidade" com um relacionamento um-para-um. Escreva uma consulta para recuperar o nome de cada pessoa e o número do documento de identidade, se estiverem disponíveis.

ATIVIDADE PRÁTICA 4

Suponha que você tenha tabelas "Autores" e "Livros" com um relacionamento um-para-muitos. Escreva uma consulta que retorne o nome de cada autor e os títulos dos livros que eles escreveram.

ATIVIDADE PRÁTICA 5

Dado um cenário com tabelas "Músicos" e "Bandas" com um relacionamento muitos-para-muitos, escreva uma consulta que liste o nome de cada músico e as bandas em que eles tocam.

DESAFIO PRÁTICO

Banco de dados de uma biblioteca

DESAFIO FINAL

Suponha que você esteja gerenciando um banco de dados para uma biblioteca. O banco de dados contém as seguintes tabelas:

Tabela "Livros" com as seguintes colunas:

- livro_id (Chave Primária)
- título
- autor_id (Chave Estrangeira)
- relacionando-se à tabela "Autores")
- ano_publicação
- gênero

DESAFIO PRÁTICO

Sistema de gerenciamento de vendas

Tabela "Autores" com as seguintes colunas:

- autor_id (Chave Primária)
- nome_autor

Tabela "Empréstimos" com as seguintes colunas:

- emprestimo_id (Chave Primária)
- livro_id (Chave Estrangeira relacionando-se à tabela "Livros")
- data_emprestimo
- data_devolução

DESAFIO PRÁTICO

Sistema de gerenciamento de vendas

Seu desafio é escrever uma consulta SQL que retorna o nome de cada autor, o título do livro emprestado e a data de empréstimo. No entanto, você precisa considerar apenas os autores cujos livros foram emprestados. Além disso, a consulta deve listar os autores em ordem alfabética e os livros em ordem de data de empréstimo crescente.

Material Complementar

- Exploração: Não tenha medo de explorar e testar diferentes códigos. A experimentação é uma grande aliada da aprendizagem.
- Perguntas: Faça perguntas, seja curioso! Entender o "porquê" das coisas ajuda a consolidar o conhecimento.
- Revisão: Revise o que aprendeu, tente explicar para si mesmo ou para outras pessoas. Ensinar é uma ótima forma de aprender.

Prática: A prática leva à perfeição. Quanto mais exercícios fizer, mais fácil será lembrar e entender os conceitos.



SE LIGA NO CONTEÚDO DA PRÓXIMA AULA!

AULA 13 DE PYTHON:
PROJETO CRUD



INFINITY SCHOOL
VISUAL ART CREATIVE CENTER

PROJETO

Gerenciador de Estoque:

- Crie um sistema para o gerenciamento que os Atributos:
Conexão com o banco de dados

Métodos:

- Métodos para adicionar, atualizar e visualizar produtos no estoque.
- Métodos para registrar vendas e atualizar o estoque de produtos após uma venda.

Métodos para gerar relatórios sobre o estoque.



INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

AULA 12 - SQL III (CRUD)