IPOG

Banco de Dados 2

EDITORA IPOG

IP5p Instituto de Pós-Graduação e Graduação – IPOG

Banco de dados 2. / Autor: Murilo Almeida Pacheco.

240f. :il.

ISBN:

1. Ciência de Dados 2. Inteligência Artificial 3. Aprendizado de Máquina 4. Análise de Dados 5. Python.

CDU: 005

Bibliotecário responsável: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

IPOG	**Sede**
Instituto de Pós Graduação e Graduação	Av. T-1 esquina com Av. T-55 N. 2.390 - Setor Bueno - Goiânia-GO. Telefone (0xx62) 3945-5050

http://www.ipog.edu.br

SUMÁRIO

APRESENTAÇÃO

Seja bem-vindo ou bem-vinda à disciplina de Banco de Dados 2

.

juntos e aproveitar ao máximo. Prepare-se para uma experiência de aprendizado transformadora.

Boa leitura e estudos!

Prof. Esp. Murilo Almeida Pacheco

OBJETIVOS

OBJETIVO GERAL

OBJETIVOS ESPECÍFICOS

Conheça como esse conteúdo foi organizado!

Unidade 1: Instalação e Configuração do MySQL, Criação do banco de dados com integridade referencial.

Unidade 2: SQL básico

Unidade 3: SQL intermediário

Unidade 4: Stored Procedures e Functions

Unidade 5: Indexação e Otimização

UNIDADE 1 FUNDAMENTOS DA CIÊNCIA DE DADOS

OBJETIVOS DA UNIDADE 1

Ao final dos estudos, você deverá ser capaz de:

- Compreender os conceitos de integridade referencial e sua importância no contexto de bancos de dados.
- Dominar os comandos DDL (Data Definition Language), como CREATE, ALTER e DROP, para manipulação de estruturas de banco de dados.
- Capacitar-se na criação de um banco de dados completo a partir de um modelo utilizando SQL.

1.1 • Requisitos de hardware e software e Processo de instalação.

Videoaula do tópico disponível no AVA:

Videoaula 1: Requisitos de hardware e software e Processo de instalação passo a passo.

O MySQL é um dos sistemas de gerenciamento de banco de dados relacionais mais populares e amplamente utilizados no mundo, conhecido por sua confiabilidade, escalabilidade e facilidade de uso. A versão mais recente, MySQL 8, traz várias melhorias e recursos avançados para atender às crescentes demandas de aplicativos modernos.

O SQL (Structured Query Language) ou Linguagem de consulta Estruturada é uma linguagem fundamental no mundo da tecnologia da informação, sendo amplamente utilizada para gerenciar e manipular dados em sistemas de banco de dados relacionais. Dominar o SQL pode abrir portas para uma ampla gama de oportunidades de carreira.

Requisitos de Hardware:

Antes de iniciar a instalação do MySQL 8, é importante garantir que o sistema atenda aos requisitos mínimos de hardware para garantir um desempenho adequado. Embora os requisitos específicos possam variar de acordo com o tamanho e a complexidade do ambiente de banco de dados, aqui estão os requisitos gerais:

- Processador: Recomenda-se um processador de 64 bits com múltiplos núcleos para melhor desempenho.
- Memória RAM: Recomenda-se pelo menos 2 GB de RAM, mas o ideal é ter mais para ambientes mais exigentes.
- Espaço em disco: O espaço em disco necessário dependerá do tamanho dos dados e do número de usuários. Recomenda-se ter pelo menos 1 GB de espaço disponível.
- Sistema Operacional: O MySQL 8 é compatível com uma variedade de sistemas operacionais, incluindo Windows, Linux e macOS. Certifique-se de verificar os requisitos específicos do sistema operacional escolhido.

Requisitos de Software:

Além dos requisitos de hardware, é necessário garantir que o sistema atenda aos requisitos de software para a instalação e execução do MySQL 8:

- Sistema Operacional: Verifique se o sistema operacional é suportado pelo MySQL 8 e instale as atualizações mais recentes.
- Dependências: Certifique-se de ter as dependências necessárias instaladas, como bibliotecas e ferramentas adicionais que o MySQL possa exigir.

- Java (opcional): Se você planeja usar o MySQL Workbench ou outras ferramentas gráficas, pode ser necessário ter o Java instalado em seu sistema.

Instalação do MySQL 8:

Agora que você verificou os requisitos de hardware e software, está pronto para instalar o MySQL 8. O processo de instalação pode variar ligeiramente dependendo do sistema operacional, mas geralmente envolve os seguintes passos:

Faça o download do instalador do MySQL 8 no site oficial do MySQL.

Execute o instalador e siga as instruções na tela para configurar o MySQL.

Durante a instalação, você será solicitado a definir uma senha para o usuário root do MySQL. Certifique-se de escolher uma senha forte e segura.

Após a conclusão da instalação, você pode começar a usar o MySQL 8 imediatamente.

Certifique-se de consultar a documentação oficial do MySQL para obter instruções detalhadas de instalação e configuração para o seu sistema operacional específico.

Com o MySQL 8 instalado e configurado, você estará pronto para começar a criar e gerenciar bancos de dados para seus aplicativos e projetos.

Aprenda Mais: Em quais Áreas Posso Trabalhar com SQL?

Disponível em: https://www.youtube.com/watch?v=RJkniBiJaRk

1.2 Integridade Referencial e exemplos

Videoaula do tópico disponível no AVA:

Videoaula 2: Integridade Referencial e exemplos

A integridade referencial é um conceito fundamental em bancos de dados relacionais, incluindo o MySQL. Neste capítulo, exploraremos o que é integridade referencial, sua importância e como implementá-la em um banco de dados MySQL..

O que é Integridade Referencial?

Em termos simples, integridade referencial refere-se à consistência e precisão dos dados em um banco de dados relacionado. Isso significa que as relações entre as tabelas devem ser mantidas e respeitadas para garantir a integridade dos dados.

Importância da Integridade Referencial

A integridade referencial é crucial para garantir a precisão e consistência dos dados em um banco de dados. Ao impor relações entre as tabelas e garantir que todas as referências entre elas sejam válidas, podemos evitar inconsistências e erros nos dados.

Implementação da Integridade Referencial

No MySQL, a integridade referencial é implementada principalmente por meio do uso de chaves estrangeiras (foreign keys). Uma chave estrangeira é uma coluna ou conjunto de colunas em uma tabela que faz referência à chave primária de outra tabela. Ao definir uma chave estrangeira, podemos garantir que os valores na coluna de referência existam na tabela referenciada.

Exemplos de Integridade Referencial

Quando colocamos uma coluna como chave estrangeira em uma tabela, assumimos responsabilidade com o banco de dados.
As colunas pertencentes à chave estrangeira da tabela A devem ter o mesmo domínio das colunas pertencentes à chave primária da tabela B.
O valor de uma chave estrangeira em uma tabela A deve ser de chave primária da tabela B, ou então ser nulo. Sintetizando, uma tabela contém uma chave estrangeira, então, o valor dessa chave só pode ser:

- Nulo – nesse caso pode, pois representa a inexistência de referência para uma linha da tabela.
  Igual ao valor de alguma chave primária na tabela referenciada.
- Você pode perguntar como ficaria uma tabela chave estrangeira nula. Vejamos:

Tabela 3.22 Funcionário

NumReg	NomeFunc	DtAdmissão	Sexo	CdCargo	CdDepto
101	Luis Sampaio	10/08/2003	M	C3	D5
104	Carlos Pereira	02/03/2004	M	C4	D6
134	Jose Alves	23/05/2002	M	C5	D1
121	Luis Paulo Souza	10/12/2001	M	C3	D5
123	Pedro Sergio Doto	29/06/2003	M	Nulo	D3
115	Roberto Fernandes	15/10/2003	M	C3	D5
22	Sergio Nogueira	10/02/2000	M	C2	D4

Na linha de Pedro Sergio Doto (NumReg:123), o valor para CdDepto está nulo, o que pode significar que ainda não está alocado a nenhum departamento ou foi deslocado de algum departamento. O que importa é que ele não tem um departamento assinalado, o que é uma situação válida.

O que não pode haver é um valor de chave estrangeira que não exista como chave primária de nenhuma linha da tabela referenciada, no caso, a Tabela 3.20.

Na definição de uma chave estrangeira, somente podemos nos referenciar a uma chave primária de uma outra tabela? Nem sempre isso é verdade.

Na criação de uma chave estrangeira, além de podemos nos referenciar a um campo chave primária de outra tabela, também podemos nos referenciar a uma coluna que tenha sido definida como única, uma chave candidata.

As chaves estrangeiras baseiam-se em valores (dados) e são puramente lógicas, ou seja, não existem apontadores físicos.

Restrições para Garantir a Integridade Referencial

Ao trabalhar com integridade referencial em um banco de dados MySQL, é essencial impor restrições que garantam a consistência e a validade dos dados. As restrições ajudam a evitar inserções, atualizações ou exclusões que possam

violar a integridade referencial. Vamos explorar algumas das principais restrições utilizadas no MySQL:

1. Chaves Estrangeiras (Foreign Keys): As chaves estrangeiras são a pedra angular para garantir a integridade referencial. Elas estabelecem uma relação entre duas tabelas, onde a chave estrangeira em uma tabela faz referência à chave primária em outra. No MySQL, as chaves estrangeiras são definidas durante a criação da tabela ou posteriormente, utilizando a cláusula FOREIGN KEY.
2. Restrição ON DELETE: Esta restrição define o que acontece com os registros dependentes quando o registro pai (referenciado) é excluído. As opções mais comuns são CASCADE, que exclui automaticamente os registros dependentes, SET NULL, que define os valores das chaves estrangeiras nas linhas dependentes como NULL, e RESTRICT, que impede a exclusão se houver registros dependentes.
3. Restrição ON UPDATE: Similar à restrição ON DELETE, a restrição ON UPDATE define o que acontece com os registros dependentes quando a chave primária na tabela pai é atualizada. As opções são as mesmas: CASCADE, SET NULL e RESTRICT.
4. Restrição UNIQUE: Embora não seja especificamente para integridade referencial, as restrições UNIQUE são úteis para garantir a unicidade de valores em uma coluna. Elas podem ser usadas em colunas que não são chaves primárias para evitar a duplicação de dados.
5. Restrição NOT NULL: Esta restrição garante que um campo não pode ter valores NULL, o que pode ajudar a garantir a integridade dos dados.

Exemplo de Implementação:

```sql
CREATE TABLE clientes (
    id_cliente INT PRIMARY KEY,
    nome VARCHAR(100),
    email VARCHAR(100) UNIQUE,
    telefone VARCHAR(15)
);

CREATE TABLE pedidos (
    id_pedido INT PRIMARY KEY,
    id_cliente INT,
    valor_total DECIMAL(10,2),
    FOREIGN KEY (id_cliente) REFERENCES clientes(id_cliente)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

Aprenda Mais: Caves estrangeiras.

Disponível em: https://www.youtube.com/watch?v=o-I9l30GEnw

1.3 Comandos DDL (Create, Alter, Drop)

Videoaula do tópico disponível no AVA:

Videoaula 3: Criação de um banco de dados usando SQL a partir de um modelo.

Os comandos DDL (Data Definition Language) no MySQL são usados para definir e gerenciar a estrutura dos objetos do banco de dados, como tabelas, índices e restrições. Neste capítulo, exploraremos os principais comandos DDL, incluindo CREATE, ALTER e DROP, e forneceremos exemplos de implementação para cada um.

1. CREATE:

O comando CREATE é usado para criar novos objetos no banco de dados, como tabelas, índices, visões e procedimentos armazenados.

Exemplo de Implementação:

```
-- Criar uma nova tabela de produtos
CREATE TABLE produtos (
    id_produto INT PRIMARY KEY,
    nome VARCHAR(100),
    preco DECIMAL(10,2),
    quantidade INT
);
```

Neste exemplo, estamos criando uma nova tabela chamada produtos com quatro colunas: id_produto, nome, preco e quantidade.

2. ALTER:

O comando ALTER é usado para modificar a estrutura de um objeto existente no banco de dados, como adicionar, modificar ou excluir colunas de uma tabela.

Exemplo de Implementação:

```
-- Adicionar uma nova coluna à tabela produtos
ALTER TABLE produtos
ADD COLUMN descricao TEXT;
```

Neste exemplo, estamos adicionando uma nova coluna chamada descricao à tabela produtos, para armazenar descrições dos produtos.

3. DROP:

O comando DROP é usado para excluir objetos do banco de dados, como tabelas, índices, visões e procedimentos armazenados.

Exemplo de Implementação:

```
-- Excluir a tabela produtos
DROP TABLE produtos;
```

DROP TABLE produtos;

Neste exemplo, estamos excluindo a tabela produtos do banco de dados.

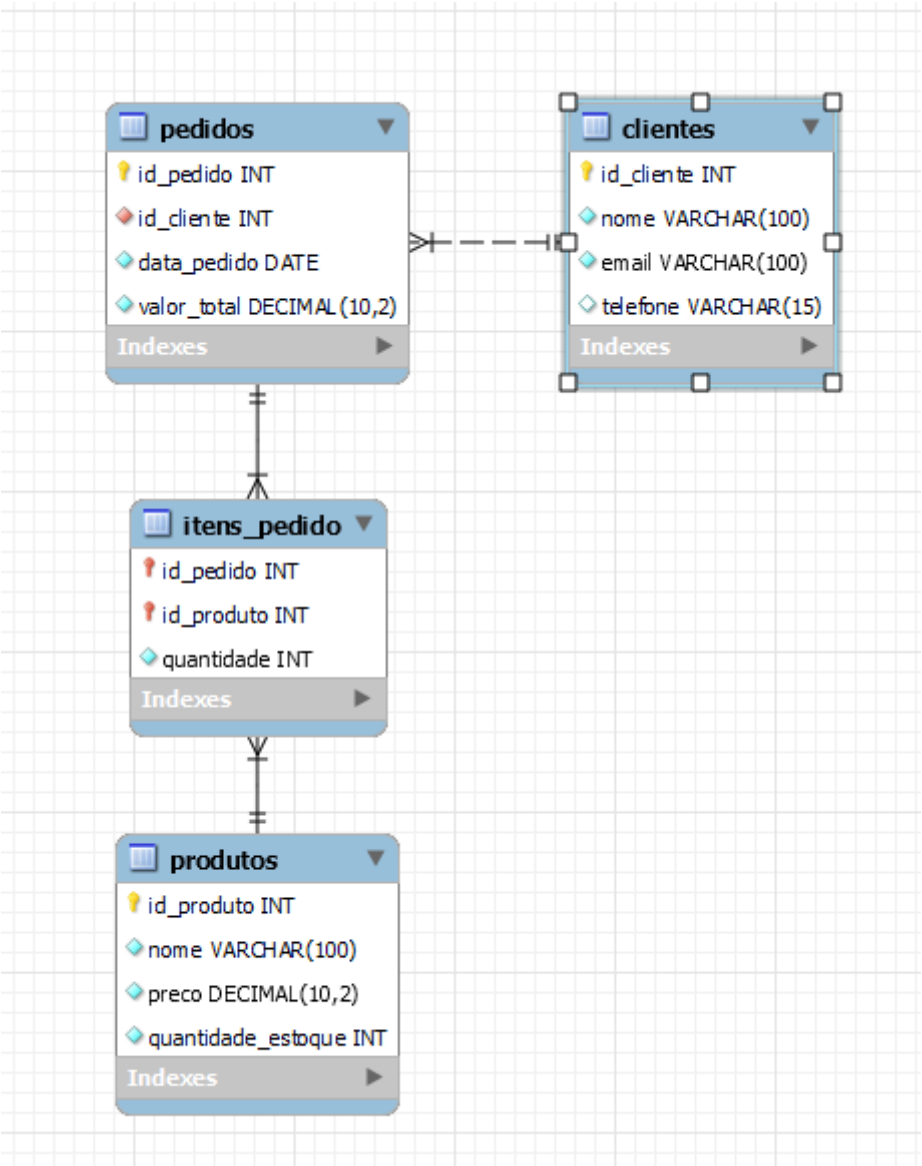
Os comandos DDL são fundamentais para definir e gerenciar a estrutura dos objetos do banco de dados no MySQL. Com o comando CREATE, podemos criar novos objetos, o comando ALTER nos permite modificar a estrutura dos objetos existentes e o comando DROP nos permite excluir objetos do banco de dados. É importante usar esses comandos com cuidado para garantir a integridade dos dados e evitar perda de informações importantes.

1.4 Criando um banco de dados completo a partir de um modelo usando SQL

A criação de um banco de dados a partir de um modelo é uma prática comum no desenvolvimento de bancos de dados relacionais. Neste capítulo, vamos explorar como podemos utilizar SQL para criar um banco de dados seguindo um modelo predefinido.

1. Modelagem do Banco de Dados:

Antes de começarmos a criar o banco de dados, é importante ter um modelo de dados que represente a estrutura desejada do banco de dados. O modelo pode ser criado usando ferramentas de modelagem de bancos de dados, como o MySQL Workbench ou o Draw.io, e deve incluir tabelas, colunas, relacionamentos e restrições.



2. Tradução do Modelo para SQL:

Uma vez que tenhamos o modelo de dados pronto, podemos traduzi-lo para comandos SQL que criarão as tabelas e definirão as relações entre elas.

Exemplo de Implementação:

```sql
-- Criação da Tabela de Clientes
CREATE TABLE clientes (
    id_cliente INT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    telefone VARCHAR(15)
);

-- Criação da Tabela de Produtos
CREATE TABLE produtos (
    id_produto INT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(100) NOT NULL,
    preco DECIMAL(10,2) NOT NULL,
    quantidade_estoque INT NOT NULL DEFAULT 0
);
```

```sql
-- Criação da Tabela de Pedidos
CREATE TABLE pedidos (
    id_pedido INT PRIMARY KEY AUTO_INCREMENT,
    id_cliente INT NOT NULL,
    data_pedido DATE NOT NULL,
    valor_total DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (id_cliente) REFERENCES clientes(id_cliente)
);

-- Tabela de Relacionamento entre Pedidos e Produtos
CREATE TABLE itens_pedido (
    id_pedido INT NOT NULL,
    id_produto INT NOT NULL,
    quantidade INT NOT NULL,
    PRIMARY KEY (id_pedido, id_produto),
    FOREIGN KEY (id_pedido) REFERENCES pedidos(id_pedido),
    FOREIGN KEY (id_produto) REFERENCES produtos(id_produto)
);
```

Neste exemplo, estamos criando duas tabelas: clientes e pedidos. A tabela clientes possui uma chave primária id_cliente e colunas para o nome, e-mail (com restrição UNIQUE) e telefone do cliente. A tabela pedidos possui uma chave primária id_pedido, uma chave estrangeira id_cliente que faz referência à tabela de clientes, e colunas para a data do pedido e o valor total.

3. Execução dos Comandos SQL:

Depois de escrever os comandos SQL, podemos executá-los em um cliente MySQL, como o MySQL Workbench ou o phpMyAdmin, para criar o banco de dados conforme definido pelo modelo.

Exercício proposto:

Exercício 1: Integridade Referencial

Suponha que você tenha as seguintes tabelas em um banco de dados:

```sql
CREATE TABLE clientes (
    id_cliente INT PRIMARY KEY,
    nome VARCHAR(100),
    email VARCHAR(100) UNIQUE
);

CREATE TABLE pedidos (
    id_pedido INT PRIMARY KEY,
    id_cliente INT,
    valor_total DECIMAL(10,2),
    FOREIGN KEY (id_cliente) REFERENCES clientes(id_cliente)
);
```

Seu exercício é:

Adicione um novo cliente à tabela clientes.

Tente inserir um novo pedido na tabela pedidos, atribuindo-o a um cliente que não existe na tabela clientes.

Execute as instruções SQL necessárias para garantir a integridade referencial.

Exercício 2: Criação do Banco de Dados

Suponha que você esteja iniciando um novo projeto e precise criar um banco de dados para armazenar informações sobre alunos e cursos em uma escola.

Seu exercício é:

Crie um modelo conceitual do banco de dados, identificando as entidades e os relacionamentos entre elas.

Traduza o modelo conceitual em um esquema de banco de dados relacional usando comandos SQL.

Inclua instruções SQL para criar pelo menos duas tabelas principais, como alunos e cursos, com suas respectivas colunas e restrições.

Adicione instruções SQL para criar chaves primárias e estrangeiras, se necessário, para garantir a integridade dos dados.

UNIDADE 2 SQL BÁSICO

Na segunda unidade, exploramos as operações de manipulação de dados em um banco de dados MySQL, incluindo INSERT, UPDATE e DELETE. Essas

operações são essenciais para adicionar, modificar e excluir dados de tabelas do banco de dados.

OBJETIVOS DA UNIDADE 2

- **Ao final dos estudos, você deverá ser capaz de:**
- **Adquirir conhecimento sobre a estrutura básica de consultas SQL.**
- **Familiarizar-se com os operadores de filtro (WHERE) para recuperar dados específicos.**
- **Aprender a ordenar resultados usando a cláusula ORDER BY.**
- **Dominar as operações de manipulação de dados, como INSERT, UPDATE e DELETE.**

2.1 Manipulação de dados (INSERT, UPDATE, DELETE)

Videoaula do tópico disponível no AVA:

Vídeo aula 4: Manipulação de dados (INSERT, UPDATE, DELETE)

2.1.1 Inserção de Dados (INSERT):

A operação INSERT é usada para adicionar novos registros a uma tabela. Sua estrutura básica é a seguinte:

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

- nome_da_tabela: o nome da tabela onde os dados serão inseridos.
- (coluna1, coluna2, ...): a lista de colunas onde os dados serão inseridos.
- VALUES (valor1, valor2, ...): os valores a serem inseridos nas colunas correspondentes.

Usando o banco criado na unidade anterior seguiremos manipulando os dados das tabelas daquele modelo.

Inserindo um novo cliente:

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

Inserindo um novo pedido:

```
INSERT INTO pedidos (id_pedido, id_cliente, valor_total, data_pedido)
VALUES (1, 1, 150.00, '2024-04-01');
```

Inserindo um novo produto:

```
INSERT INTO produtos (id_produto, nome, preco, quantidade_estoque)
VALUES (1, 'Camisa', 29.99, 100);
```

2.1.2 Atualização de Dados (UPDATE):

A operação UPDATE é usada para modificar os registros existentes em uma tabela. Sintaxe básica:

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

Atualizando o email de um cliente:

```
UPDATE clientes
SET email = 'joao_novo@example.com'
WHERE id_cliente = 1;
```

Atualizando o valor total de um pedido:

```
UPDATE pedidos
SET valor_total = 200.00
WHERE id_pedido = 1;
```

2.1.3 Exclusão de Dados (DELETE):

A operação DELETE é usada para remover registros de uma tabela. Sintaxe básica:

```
DELETE FROM nome_da_tabela
WHERE condição;
```

- nome_da_tabela: o nome da tabela onde os registros serão excluídos.
- WHERE condição: a condição que especifica quais registros serão excluídos.

Excluindo um produto do estoque:

```
DELETE FROM produtos
WHERE id_produto = 1;
```

Excluindo um cliente e seus pedidos associados:

```
DELETE FROM clientes
WHERE id_cliente = 1;
```

As operações de manipulação de dados são fundamentais para interagir com um banco de dados MySQL. Com os comandos INSERT, UPDATE e DELETE, podemos adicionar, modificar e excluir dados das tabelas conforme necessário, mantendo assim a integridade e a precisão dos dados armazenados.

Aprenda Mais:

Inserindo Dados

Disponível em:

https://www.youtube.com/watch?v=NCG9niOlm40&list=PLHz_AreHm4dkBs-795Dsgvau_ekxg8g1r&index=7

2.2 Estrutura básica de uma consulta SQL

Videoaula do tópico disponível no AVA:

Video aula 5: Estrutura básica de uma consulta SQL

Neste capítulo, exploraremos a estrutura básica de uma consulta SQL usando como exemplo o banco de dados que criamos anteriormente, composto pelas tabelas clientes, pedidos e produtos. Vamos detalhar cada exemplo e explicar a estrutura de consulta.

1. **Consulta Simples (SELECT):**

A operação SELECT é usada para recuperar dados de uma ou mais tabelas. Sua estrutura básica é a seguinte:

```
SELECT coluna1, coluna2, ...
FROM nome_da_tabela;
```

- coluna1, coluna2, ...: as colunas que você deseja recuperar.

- nome_da_tabela: o nome da tabela da qual você deseja recuperar os dados.

Exemplo de Consulta Simples:

```
SELECT *
FROM clientes;
```

2. **Consulta com Condições (WHERE):**

A cláusula WHERE é usada para filtrar os resultados com base em uma condição específica. Sua estrutura básica é a seguinte:

```
SELECT coluna1, coluna2, ...
FROM nome_da_tabela
WHERE condição;
```

- condição: a condição que os registros devem atender para serem incluídos no resultado.

Exemplo de Consulta com Condição:

```
SELECT *
FROM pedidos
WHERE id_cliente = 1;
```

,

A estrutura básica de uma consulta SQL envolve as cláusulas SELECT, FROM, WHERE e ORDER BY, que permitem recuperar e manipular dados de um banco de dados. Compreender essa estrutura é fundamental para realizar consultas eficazes e obter os resultados desejados.

Consultas.

Disponível em: https://www.youtube.com/watch?v=GaOlyL3Uv9M&list=PLHz_AreHm4dkBs-795Dsgvau_ekxg8g1r&index=13

Consultas parte 2.

Disponível em:

https://www.youtube.com/watch?v=q4hPo83-Buo&list=PLHz_AreHm4dkBs-795Dsgvau_ekxg8g1r&index=14

2.3 Operadores de filtro (WHERE)

Videoaula do tópico disponível no AVA:

Video aula 6: Operadores de filtro (WHERE)

2.3.1 Operadores de Filtro (WHERE)

Neste capítulo, abordaremos os operadores de filtro SQL utilizando como exemplo o banco de dados composto pelas tabelas clientes, pedidos e produtos. Exploraremos vários filtros e detalharemos sua estrutura, fornecendo exemplos práticos de consultas.

1. Operador de Igualdade (=):

O operador de igualdade é usado para filtrar os registros que possuem um valor específico em uma coluna. Sua estrutura básica é a seguinte:

```
SELECT coluna1, coluna2, ...
FROM nome_da_tabela
WHERE coluna = valor;
```

Exemplo de Uso:

```
SELECT *
FROM pedidos
WHERE id_cliente = 1;
```

Esta consulta retorna todos os pedidos feitos pelo cliente com o ID igual a 1.

2. Operador de Diferente (!= ou <>):

O operador de diferente é usado para filtrar os registros que não possuem um valor específico em uma coluna. Sua estrutura básica é semelhante ao operador de igualdade:

```
SELECT coluna1, coluna2, ...
FROM nome_da_tabela
WHERE coluna != valor;
```

Exemplo de Uso:

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

Esta consulta retorna todos os clientes cujo ID não é igual a 1.

3. Operadores Lógicos (AND, OR):

Os operadores lógicos são usados para combinar múltiplas condições em uma cláusula WHERE. Por exemplo:

- O operador AND retorna registros que atendem a todas as condições especificadas.
- O operador OR retorna registros que atendem a pelo menos uma das condições especificadas.

Exemplo de Uso:

```
SELECT *
FROM pedidos
WHERE id_cliente = 1 AND valor_total > 100;
```

Esta consulta retorna todos os pedidos feitos pelo cliente com ID igual a 1 e com um valor total superior a 100.

Os operadores de filtro WHERE são fundamentais para a criação de consultas SQL específicas que retornam os dados desejados de um banco de dados. Compreender a estrutura e o uso desses operadores é essencial para realizar consultas eficazes e obter resultados precisos.

Exercício proposto:

Exercícios: Operadores de Filtro (WHERE)

Exercício 1: Escreva uma consulta SQL para recuperar todos os clientes que têm mais de 30 anos.

Exercício 2: Escreva uma consulta SQL para recuperar todos os pedidos feitos após 01 de janeiro de 2023.

Exercício 3: Escreva uma consulta SQL para recuperar todos os produtos com um preço superior a R$ 50,00 e uma quantidade em estoque inferior a 50 unidades.

Respostas esperadas:

1)
```
SELECT *

FROM clientes

WHERE idade > 30;
```

2)
```
SELECT *
FROM pedidos
WHERE data_pedido > '2023-01-01';
```

3)
```
SELECT *
FROM produtos
WHERE preco > 50.00 AND quantidade_estoque < 50;
```

2.3.2 Outros Operadores

1. Operador BETWEEN:

O operador BETWEEN é usado para selecionar valores dentro de um intervalo específico. Sua estrutura básica é a seguinte:

```sql
SELECT coluna1, coluna2, ...
FROM nome_da_tabela
WHERE coluna BETWEEN valor1 AND valor2;
```

Exemplo de Uso:

```sql
SELECT *
FROM pedidos
WHERE data_pedido BETWEEN '2023-01-01' AND '2023-12-31';
```

Esta consulta retorna todos os pedidos feitos durante o ano de 2023.

2. Operadores de Comparação (> , < , >= , <=):

Além do operador BETWEEN, temos os operadores de comparação padrão para verificar se um valor é maior, menor, maior ou igual, ou menor ou igual a outro valor.

Exemplo de Uso:

```sql
SELECT *
FROM produtos
WHERE preco > 50.00;
```

Esta consulta retorna todos os produtos com preço superior a R$ 50,00.

3. Operador IN:

O operador IN é usado para verificar se um valor está presente em um conjunto específico de valores. Sua estrutura básica é a seguinte:

```sql
SELECT coluna1, coluna2, ...
FROM nome_da_tabela
WHERE coluna IN (valor1, valor2, ...);
```

Exemplo de Uso:

```sql
SELECT *
FROM clientes
WHERE tipo IN ('Premium', 'VIP');
```

Esta consulta retorna todos os clientes que têm o tipo 'Premium' ou 'VIP'.

4. Operador LIKE:

O operador LIKE é usado para pesquisar por padrões em uma coluna de texto. Ele pode ser combinado com o caractere % para representar zero ou mais caracteres, ou _ para representar um único caractere.

Exemplo de Uso:

```sql
SELECT *
FROM produtos
WHERE nome LIKE 'Camisa%';
```

Exercícios:

Operadores de Filtro (WHERE)

Exercício 1: Escreva uma consulta SQL para recuperar todos os pedidos feitos entre 01 de janeiro de 2022 e 31 de dezembro de 2022.

Exercício 2: Escreva uma consulta SQL para recuperar todos os clientes que têm entre 25 e 40 anos de idade.

Exercício 3: Escreva uma consulta SQL para recuperar todos os produtos cujo preço está entre R$ 100,00 e R$ 200,00.

Exercício 4: Escreva uma consulta SQL para recuperar todos os clientes que são do tipo 'Premium' ou 'VIP'.

Exercício 5: Escreva uma consulta SQL para recuperar todos os produtos cujo nome começa com a letra 'A'.

Respostas Esperadas:

Exercício 1:

```
SELECT *
FROM pedidos
WHERE data_pedido BETWEEN '2022-01-01' AND '2022-12-31';
```

Exercício 2:

```
SELECT *
FROM clientes
WHERE idade BETWEEN 25 AND 40;
```

Exercício 3:

```
SELECT *
FROM produtos
WHERE preco BETWEEN 100.00 AND 200.00;
```

Exercício 4:

```
SELECT *
FROM clientes
WHERE tipo IN ('Premium', 'VIP');
```

Exercício 5:

```
SELECT *
FROM produtos
WHERE nome LIKE 'A%';
```

Estes exercícios permitem praticar o uso dos diferentes operadores de filtro WHERE em consultas SQL, aplicando diversas condições para recuperar dados específicos do banco de dados.

Aprenda Mais:

Consultas parte 3.

Disponível em:

https://www.youtube.com/watch?v=ocyVJ9gRUaE&list=PLHz_AreHm4dkBs-795Dsgvau_ekxg8g1r&index=15

2.4 Ordenação de resultados (ORDER BY)

Videoaula do tópico disponível no AVA:

Video aula 7: Ordenação de resultados (ORDER BY)

2.4.1 Ordenação de Resultados (ORDER BY)

Neste capítulo, exploraremos a cláusula ORDER BY, que é usada para classificar os resultados de uma consulta SQL. Continuaremos utilizando o mesmo banco de dados composto pelas tabelas clientes, pedidos e produtos, e detalharemos sua estrutura e exemplos de uso.

1. Ordenação Ascendente:

A cláusula ORDER BY pode ser usada para classificar os resultados em ordem ascendente. Sua estrutura básica é a seguinte:

```
SELECT coluna1, coluna2, ...
FROM nome_da_tabela
ORDER BY coluna ASC;
```

Exemplo de Uso:

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

Esta consulta retorna todos os clientes ordenados em ordem alfabética pelo nome.

2. Ordenação Descendente:

Também é possível ordenar os resultados em ordem descendente usando o modificador DESC. Por exemplo:

```
SELECT *
FROM produtos
ORDER BY preco DESC;
```

Esta consulta retorna todos os produtos ordenados em ordem decrescente pelo preço.

3. Ordenação por Múltiplas Colunas:

É possível ordenar os resultados por múltiplas colunas, aplicando uma ordem de prioridade. Por exemplo:

```
SELECT *
FROM pedidos
ORDER BY data_pedido ASC, valor_total DESC;
```

Esta consulta retorna todos os pedidos ordenados em ordem crescente pela data do pedido e, em seguida, em ordem decrescente pelo valor total.

Exercícios:

Exercício 1: Escreva uma consulta SQL para recuperar todos os clientes ordenados por idade em ordem descendente.

Exercício 2: Escreva uma consulta SQL para recuperar todos os produtos ordenados por quantidade em estoque em ordem ascendente.

Exercício 3: Escreva uma consulta SQL para recuperar todos os pedidos ordenados pela data do pedido em ordem descendente.

Respostas Esperadas

Exercício 1:

```
SELECT *
FROM clientes
ORDER BY idade DESC;
```

Exercício 2:

```
SELECT *
FROM produtos
ORDER BY quantidade_estoque ASC;
```

Exercício 3:

```
SELECT *
FROM pedidos
ORDER BY data_pedido DESC;
```

Estes exercícios permitem praticar o uso da cláusula ORDER BY para ordenar os resultados de uma consulta SQL de acordo com critérios específicos.

Aprenda Mais:

Order by de várias formas

Disponível em: https://www.youtube.com/watch?v=cerkGuFNPGY

UNIDADE 3 SQL INTERMEDIÁRIO

OBJETIVOS DA UNIDADE 3

Ao final dos estudos, você deverá ser capaz de:

- **Explorar técnicas avançadas de consultas SQL, incluindo junções (INNER JOIN, LEFT JOIN, RIGHT JOIN) para combinar dados de múltiplas tabelas.**
- **Dominar o uso de subconsultas (Subquery) para realizar consultas mais complexas.**
- **Compreender o uso do GROUP BY e funções de agregação (SUM, AVG, COUNT, etc.) para resumir dados.**
- **Utilizar Views para simplificar consultas complexas e melhorar a manutenibilidade do código SQL.**

3.1 Junções (INNER JOIN, LEFT JOIN, RIGHT JOIN)

Videoaula do tópico disponível no AVA:

Videoaula 9: Junções (INNER JOIN, LEFT JOIN, RIGHT JOIN)

Neste capítulo, vamos explorar as junções em SQL, incluindo INNER JOIN, LEFT JOIN e RIGHT JOIN. Continuaremos utilizando o mesmo banco de dados composto pelas tabelas clientes, pedidos e produtos.

1. INNER JOIN:

A junção INNER JOIN é usada para combinar linhas de duas ou mais tabelas com base em uma condição especificada. Sua estrutura básica é a seguinte:

```
SELECT colunas
FROM tabela1
INNER JOIN tabela2 ON tabela1.coluna_chave = tabela2.coluna_chave;
```

Exemplo de Uso:

```
SELECT clientes.nome, pedidos.valor_total
FROM clientes
INNER JOIN pedidos ON clientes.id_cliente = pedidos.id_cliente;
```

Esta consulta retorna o nome do cliente e o valor total de cada pedido correspondente.

2. LEFT JOIN:

A junção LEFT JOIN retorna todas as linhas da tabela à esquerda (tabela1) e as linhas correspondentes da tabela à direita (tabela2). Se não houver correspondência, são retornados valores NULL para as colunas da tabela à direita. Sua estrutura é similar à do INNER JOIN.

Exemplo de Uso:

```sql
SELECT clientes.nome, pedidos.valor_total
FROM clientes
LEFT JOIN pedidos ON clientes.id_cliente = pedidos.id_cliente;
```

Esta consulta retorna todos os clientes, incluindo aqueles que não fizeram nenhum pedido. Se um cliente não tiver nenhum pedido correspondente, o valor_total será NULL.

3. RIGHT JOIN:

A junção RIGHT JOIN é semelhante ao LEFT JOIN, mas retorna todas as linhas da tabela à direita (tabela2) e as linhas correspondentes da tabela à esquerda (tabela1). Se não houver correspondência, são retornados valores NULL para as colunas da tabela à esquerda. Sua estrutura é similar à do INNER JOIN.

Exemplo de Uso:

```sql
SELECT clientes.nome, pedidos.valor_total
FROM clientes
RIGHT JOIN pedidos ON clientes.id_cliente = pedidos.id_cliente;
```

Esta consulta retorna todos os pedidos, incluindo aqueles sem um cliente correspondente. Se um pedido não tiver um cliente correspondente, o nome do cliente será NULL.

As junções INNER JOIN, LEFT JOIN e RIGHT JOIN são ferramentas poderosas para combinar dados de múltiplas tabelas em consultas SQL. Compreender suas diferenças e saber quando usá-las é essencial para

Aprenda Mais:

Inner Join com várias tabelas

Disponível em:

https://www.youtube.com/watch?v=jx2ne8iZMOA&list=PLHz_AreHm4dkBs-795Dsgvau_ekxg8g1r&index=18

Exercícios: Junções (INNER JOIN, LEFT JOIN, RIGHT JOIN)

Exercício 1: Escreva uma consulta SQL que retorne o nome do cliente e o nome do produto de cada pedido.

Exercício 2: Escreva uma consulta SQL que retorne todos os clientes, juntamente com o valor total de todos os pedidos feitos por cada cliente.

Exercício 3: Escreva uma consulta SQL que retorne todos os pedidos, incluindo aqueles que não têm um cliente correspondente.

Respostas esperadas:

Exercício 1:

```
SELECT clientes.nome AS nome_cliente, produtos.nome AS nome_produto

FROM clientes

INNER JOIN pedidos ON clientes.id_cliente = pedidos.id_cliente

INNER JOIN produtos ON pedidos.id_produto = produtos.id_produto;
```

Exercício 2:

```
SELECT clientes.nome, SUM(pedidos.valor_total) AS total_pedidos

FROM clientes

LEFT JOIN pedidos ON clientes.id_cliente = pedidos.id_cliente

GROUP BY clientes.nome;
```

Exercício 3:

```
SELECT pedidos.id_pedido, clientes.nome AS nome_cliente, pedidos.valor_total

FROM pedidos

LEFT JOIN clientes ON pedidos.id_cliente = clientes.id_cliente;
```

3.2 Subconsultas (Subquery)

Videoaula do tópico disponível no AVA:

Videoaula 10: Subconsultas (Subquery)

Neste capítulo, vamos explorar o uso de subconsultas em SQL. Uma subconsulta é uma consulta aninhada dentro de outra consulta SQL. Continuaremos utilizando o mesmo banco de dados composto pelas tabelas clientes, pedidos e produtos.

1. Subconsulta na Cláusula WHERE:

As subconsultas podem ser usadas na cláusula WHERE para filtrar os resultados com base em uma condição proveniente de outra consulta. Por exemplo:

```sql
SELECT nome
FROM clientes
WHERE id_cliente IN (SELECT id_cliente FROM pedidos);
```

Esta consulta retorna o nome de todos os clientes que fizeram pelo menos um pedido.

2. Subconsulta na Cláusula FROM:

As subconsultas também podem ser usadas na cláusula FROM para fornecer dados temporários para a consulta principal. Por exemplo:

```sql
SELECT AVG(total_pedidos)
FROM (SELECT id_cliente, SUM(valor_total) AS total_pedidos
      FROM pedidos GROUP BY id_cliente) AS subconsulta;
```

Esta consulta retorna a média do valor total de todos os pedidos feitos por cliente.

3. Subconsulta na Cláusula SELECT:

As subconsultas podem ser usadas na cláusula SELECT para calcular valores ou fornecer informações adicionais. Por exemplo:

```sql
SELECT nome, (SELECT COUNT(*)
              FROM pedidos WHERE pedidos.id_cliente = clientes.id_cliente) AS num_pedidos
FROM clientes;
```

Esta consulta retorna o nome de cada cliente juntamente com o número total de pedidos feitos por cliente.

Exercícios: Subconsultas (Subquery)

Exercício 1: Escreva uma consulta SQL para recuperar o nome de todos os clientes que fizeram pedidos de produtos com um preço superior a R$ 500,00.

Exercício 2: Escreva uma consulta SQL para recuperar o nome de todos os clientes que fizeram mais de dois pedidos.

Exercício 3: Escreva uma consulta SQL para recuperar o nome e o preço do produto mais caro em todo o banco de dados.

Respostas esperadas:

Exercício 1:

SELECT nome

FROM clientes

WHERE id_cliente IN (SELECT id_cliente FROM pedidos WHERE id_produto IN (SELECT id_produto FROM produtos WHERE preco > 500.00));

Exercício 2:

SELECT nome

FROM clientes

WHERE id_cliente IN (SELECT id_cliente FROM pedidos GROUP BY id_cliente HAVING COUNT(*) > 2);

Exercício 3:

SELECT nome, preco

FROM produtos

WHERE preco = (SELECT MAX(preco) FROM produtos);

3.3 Group By e Funções Agregação (SUM, AVG, COUNT, etc.)

Videoaula do tópico disponível no AVA:

Videoaula 11: Group By e Funções de Agregação (SUM, AVG, COUNT, etc.)

Neste capítulo, exploraremos o uso da cláusula GROUP BY e funções de agregação como SUM, AVG e COUNT em consultas SQL. Continuaremos utilizando o mesmo banco de dados composto pelas tabelas clientes, pedidos e produtos.

1. Group By:

A cláusula GROUP BY é usada para agrupar linhas que têm o mesmo valor em uma ou mais colunas. Por exemplo:

```
SELECT categoria, COUNT(*) AS num_produtos
FROM produtos
GROUP BY categoria;
```

Esta consulta retorna o número de produtos em cada categoria.

2. Funções de Agregação:

As funções de agregação, como SUM, AVG e COUNT, são usadas para realizar cálculos em um conjunto de valores. Por exemplo:

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

Esta consulta retorna o preço médio de todos os produtos.

3. Group By com Funções de Agregação:

As funções de agregação podem ser combinadas com a cláusula GROUP BY para calcular valores agregados para grupos de linhas. Por exemplo:

```sql
SELECT id_cliente, COUNT(*) AS num_pedidos, SUM(valor_total) AS total_gasto
FROM pedidos
GROUP BY id_cliente;
```

Esta consulta retorna o número de pedidos e o total gasto por cada cliente.

Exercícios: Group By e Funções de Agregação

Exercício 1: Escreva uma consulta SQL que retorne o número de pedidos feitos por cada cliente.

Exercício 2: Escreva uma consulta SQL que retorne o total gasto em cada pedido.

Exercício 3: Escreva uma consulta SQL que retorne a categoria de produto com o preço médio mais alto.

Respostas Esperadas

Exercício 1:

```sql
SELECT id_cliente, COUNT(*) AS num_pedidos
FROM pedidos
GROUP BY id_cliente;
```

Exercício 2:

```
SELECT id_pedido, SUM(valor_total) AS total_gasto
FROM pedidos
GROUP BY id_pedido;
```

Exercício 3:

```
SELECT categoria, AVG(preco) AS preco_medio
FROM produtos
GROUP BY categoria
ORDER BY preco_medio DESC
LIMIT 1;
```

3.4 Utilização de Views para simplificação de consultas complexas

Videoaula do tópico disponível no AVA:

Videoaula 12: Utilização de Views para simplificação de consultas complexas.

Neste capítulo, vamos explorar o uso de Views em SQL para simplificar consultas complexas e facilitar o acesso aos dados. Uma View é uma consulta SQL armazenada no banco de dados e tratada como uma tabela virtual.

1. Criação de Views:

As Views são criadas usando a cláusula CREATE VIEW. Por exemplo:

```sql
CREATE VIEW vw_pedidos_cliente AS
SELECT pedidos.id_pedido, pedidos.valor_total, clientes.nome AS nome_cliente
FROM pedidos
INNER JOIN clientes ON pedidos.id_cliente = clientes.id_cliente;
```

Esta View permite acessar os detalhes dos pedidos juntamente com o nome do cliente.

2. Utilização de Views:

Uma vez criada, uma View pode ser consultada como uma tabela normal. Por exemplo:

```
SELECT * FROM vw_pedidos_cliente;
```

Esta consulta retorna todos os pedidos juntamente com os nomes dos clientes.

3. Atualização de Views:

As Views podem ser atualizadas usando a cláusula CREATE OR REPLACE VIEW. Por exemplo:

```
CREATE OR REPLACE VIEW vw_pedidos_cliente AS
SELECT pedidos.id_pedido, pedidos.valor_total, clientes.nome AS nome_cliente, clientes.endereco
FROM pedidos
        INNER JOIN clientes ON pedidos.id_cliente = clientes.id_cliente;
```

Esta instrução atualiza a View vw_pedidos_cliente para incluir o endereço dos clientes.

Aprenda Mais:

Diferenças entre views e tabelas no banco de dados.

Disponível em: https://www.youtube.com/watch?v=MOXHM8tdiYU

UNIDADE 4 STORED PROCEDURES E FUNCTIONS

Nesta unidade, vamos abordar os princípios essenciais das funções mais avançadas de bancos de dados que podem ser consideradas verdadeiros algoritmos usando linguagem SQL.

OBJETIVOS DA UNIDADE 4

- **Ao final dos estudos, você deverá ser capaz de:**
- **Entender os conceitos e vantagens das Stored Procedures e Functions.**
- **Dominar a sintaxe para criação de Procedures e Functions, incluindo parâmetros de entrada e saída.**
- **Aplicar os conceitos em cenários reais para automatizar tarefas e melhorar a eficiência do sistema.**

4.1 Stored Procedures e Functions

Videoaula do tópico disponível no AVA:

Videoaula 13: Stored Procedures e Functions Conceitos e vantagens.

Neste capítulo, exploraremos os conceitos, vantagens e uso prático de Stored Procedures e Functions em bancos de dados relacionais.

1. Conceitos:

- Stored Procedures e Functions são blocos de código SQL armazenados no banco de dados para execução posterior.
- Stored Procedures: São conjuntos de instruções SQL que podem realizar operações complexas, como inserções, atualizações e exclusões de dados, além de lógica de programação como loops e condicionais.
- Functions: São rotinas que retornam um valor escalar baseado em parâmetros de entrada, sendo úteis para cálculos repetitivos ou operações comuns.

2. Vantagens:

- Reutilização de Código: Stored Procedures e Functions permitem encapsular lógica de negócios complexa, facilitando a reutilização em várias partes do sistema.
- Melhoria de Desempenho: Ao serem armazenados no banco de dados, Stored Procedures e Functions podem ser compilados e otimizados, resultando em melhor desempenho de consulta.
- Segurança: As Stored Procedures podem controlar o acesso aos dados, permitindo aos usuários executar apenas operações específicas.

Aprenda Mais:

Usos de Procedures. Disponível em: https://www.youtube.com/watch?v=lHE5i7sbhhU

4.2 Sintaxe para criação de Procedures e Functions

Videoaula do tópico disponível no AVA:

Videoaula 14: Sintaxe para criação de Procedures e Functions

Neste capítulo, vamos aprofundar na sintaxe para criação de Stored Procedures e Functions em SQL, detalhando seu uso e fornecendo exemplos práticos.

1. Sintaxe para Criação de Procedures:

A sintaxe básica para criar uma Stored Procedure em SQL é a seguinte:

```
CREATE PROCEDURE nome_procedure(param1 tipo_param1, param2 tipo_param2)
BEGIN
    -- Corpo da Stored Procedure
END;
```

Os parâmetros são opcionais e podem ser utilizados para passar valores para dentro da Procedure. O corpo da Procedure contém as instruções SQL que serão executadas quando a Procedure for chamada.

Exemplo de Procedure:

Vamos criar uma Stored Procedure simples que retorna o número total de clientes:

```
CREATE PROCEDURE total_clientes()
BEGIN
    SELECT COUNT(*) AS total FROM clientes;
END;
```

2. Sintaxe para Criação de Functions:

A sintaxe para criar uma Function em SQL é semelhante à sintaxe de uma Procedure:

```
CREATE FUNCTION nome_function(param1 tipo_param1, param2 tipo_param2) RETURNS tipo_retorno
BEGIN
    -- Corpo da Function
END;
```

A diferença principal é que uma Function deve ter um tipo de retorno definido, indicado pelo RETURNS.

Exemplo de Function:

Vamos criar uma Function que calcula o valor total de um pedido, dado o ID do pedido:

```
CREATE FUNCTION calcular_total_pedido(id_pedido INT) RETURNS DECIMAL(10,2)
BEGIN
    DECLARE total DECIMAL(10,2);
    SELECT SUM(valor_total) INTO total FROM pedidos WHERE id_pedido = id_pedido;
    RETURN total;
END;
```

Stored Procedures e Functions são recursos essenciais em bancos de dados relacionais, oferecendo flexibilidade e reutilização de código. Ao compreender a sintaxe para criar esses objetos no banco de dados, é possível aproveitar ao máximo seu potencial em diferentes cenários de aplicação.

Exercícios: Procedures e Functions

Exercício 1: Stored Procedure para Inserir um Novo Cliente

Crie uma Stored Procedure chamada inserir_cliente que receba como parâmetros o nome, email e telefone de um cliente e insira esses dados na tabela de clientes.

Exercício 2: Function para Calcular o Total de Vendas de um Produto

Crie uma Function chamada calcular_total_vendas que receba como parâmetro o ID de um produto e retorne o total de vendas desse produto.

Respostas esperadas

Exercício 1:

```
CREATE PROCEDURE inserir_cliente(nome VARCHAR(255), email VARCHAR(255), telefone VARCHAR(20))

BEGIN

    INSERT INTO clientes (nome, email, telefone) VALUES (nome, email, telefone);

END;
```

Exercício 2:

```
CREATE FUNCTION calcular_total_vendas(id_produto INT) RETURNS DECIMAL(10,2)

BEGIN

    DECLARE total_vendas DECIMAL(10,2);

    SELECT SUM(valor_total) INTO total_vendas FROM pedidos WHERE id_produto = id_produto;

    RETURN total_vendas;

END;
```

4.3 Parâmetros de entrada e saída

Videoaula do tópico disponível no AVA:

Videoaula 15: Parâmetros de entrada e saída

Neste capítulo, exploraremos como definir e utilizar parâmetros de entrada e saída em Stored Procedures e Functions em bancos de dados relacionais.

1. Parâmetros de Entrada:

Os parâmetros de entrada são utilizados para passar valores para dentro de uma Stored Procedure ou Function durante sua execução. Eles são especificados na declaração da Procedure ou Function e podem ser utilizados dentro do seu corpo.

Sintaxe para Parâmetros de Entrada:

```
CREATE PROCEDURE nome_procedure(param1 tipo_param1, param2 tipo_param2)
```

Exemplo de Procedure com Parâmetros de Entrada:

```
CREATE PROCEDURE buscar_cliente_por_id(id_cliente INT)
BEGIN
    SELECT * FROM clientes WHERE id = id_cliente;
END;
```

2. Parâmetros de Saída:

Os parâmetros de saída são utilizados para retornar valores de uma Stored Procedure ou Function para o código que a chamou. Eles são definidos como variáveis de saída e podem ser utilizados para transmitir informações de volta para o código cliente.

Sintaxe para Parâmetros de Saída:

```
CREATE PROCEDURE nome_procedure(OUT param_saida tipo_param_saida)
```

Exemplo de Procedure com Parâmetros de Saída:

```
CREATE PROCEDURE obter_total_clientes(OUT total INT)
BEGIN
    SELECT COUNT(*) INTO total FROM clientes;
END;
```

3. Uso Prático:

Os parâmetros de entrada e saída proporcionam uma maneira poderosa de interagir com Procedures e Functions, permitindo que elas recebam dados e retornem resultados de forma dinâmica e flexível.

4.4 Uso prático em cenários reais

Videoaula do tópico disponível no AVA:

Videoaula 16 Uso prático em cenários reais.

Neste capítulo, exploraremos exemplos práticos de como utilizar Stored Procedures e Functions em cenários reais de aplicação em bancos de dados relacionais.

1. Automatização de Tarefas Recorrentes:

Stored Procedures são frequentemente utilizadas para automatizar tarefas recorrentes que envolvem manipulação de dados. Por exemplo, uma Procedure pode ser criada para realizar backups regulares do banco de dados, executar rotinas de manutenção ou enviar notificações automáticas.

Exemplo: Backup Automático do Banco de Dados

```sql
CREATE PROCEDURE realizar_backup()
BEGIN
    -- Lógica para realizar o backup do banco de dados
END;
```

2. Implementação de Regras de Negócio Complexas:

Stored Procedures e Functions podem ser utilizadas para implementar regras de negócio complexas diretamente no banco de dados. Isso permite que a lógica de negócio seja centralizada e executada de forma consistente em todas as interações com o banco de dados.

Exemplo: Validação de Dados ao Inserir um Pedido

```sql
CREATE PROCEDURE inserir_pedido(id_cliente INT, total_pedido DECIMAL)
BEGIN
    IF total_pedido <= 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'O total do pedido deve ser maior que zero';
    ELSE
        INSERT INTO pedidos (id_cliente, total_pedido) VALUES (id_cliente, total_pedido);
    END IF;
END;
```

3. Otimização de Desempenho:

Stored Procedures e Functions podem ser utilizadas para otimizar o desempenho de consultas e operações no banco de dados. Ao encapsular consultas complexas ou cálculos repetitivos em Procedures ou Functions, é possível reduzir a sobrecarga de rede e melhorar o tempo de resposta das consultas.

Exemplo: Function para Calcular o Total de Vendas de um Produto

```sql
CREATE FUNCTION calcular_total_vendas(id_produto INT) RETURNS DECIMAL(10,2)

BEGIN
    DECLARE total_vendas DECIMAL(10,2);
    SELECT SUM(valor_total) INTO total_vendas FROM pedidos WHERE id_produto = id_produto;
    RETURN total_vendas;
END;
```

Exercícios:

Exercício 1: Implementação de Regra de Negócio

Crie uma Stored Procedure chamada inserir_pedido que valida se o total do pedido é maior que zero antes de inseri-lo na tabela de pedidos. Caso o total seja zero ou negativo, a Procedure deve retornar uma mensagem de erro.

Exercício 2: Otimização de Consulta

Crie uma Function chamada calcular_total_vendas que recebe o ID de um produto como parâmetro e retorna o total de vendas desse produto. Esta Function deve ser utilizada para otimizar consultas que necessitam calcular o total de vendas de um produto.

Respostas esperadas:

Exercício 1:

```
CREATE PROCEDURE inserir_pedido(id_cliente INT, total_pedido
DECIMAL)
BEGIN
   IF total_pedido <= 0 THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'O total do
pedido deve ser maior que zero';
   ELSE
      INSERT INTO pedidos (id_cliente, total_pedido) VALUES
(id_cliente, total_pedido);
   END IF;
END;
```

Exercício 2:

```
CREATE FUNCTION calcular_total_vendas(id_produto INT) RETURNS
DECIMAL(10,2)
BEGIN
   DECLARE total_vendas DECIMAL(10,2);
   SELECT SUM(valor_total) INTO total_vendas FROM pedidos WHERE
id_produto = id_produto;
   RETURN total_vendas;
END;
```

UNIDADE 5 INDEXAÇÃO E OTIMIZAÇÃO

OBJETIVOS DA UNIDADE 5

Ao final dos estudos, você deverá ser capaz de:

- **Explorar a utilização de índices compostos para melhorar a performance de consultas.**
- **Conhecer estratégias de indexação eficientes para otimizar a recuperação de dados.**
- **Aprender a analisar o plano de execução de consultas para identificar possíveis gargalos de desempenho.**
- **Realizar ajustes nas configurações do servidor para otimizar o desempenho do banco de dados.**

5.1 Estratégias de indexação eficientes

Videoaula do tópico disponível no AVA:

Videoaula 17: Estratégias de indexação eficientes

Neste capítulo, exploraremos estratégias avançadas de indexação para melhorar o desempenho de consultas em bancos de dados relacionais. Uma indexação eficiente é crucial para acelerar consultas e reduzir o tempo de resposta em sistemas de gerenciamento de banco de dados.

1. Índices Compostos:

Os índices compostos são criados em múltiplas colunas de uma tabela e podem ser utilizados para otimizar consultas que envolvem múltiplos critérios de seleção. Ao criar índices compostos nas colunas mais frequentemente utilizadas em cláusulas WHERE, JOIN e ORDER BY, é possível melhorar significativamente o desempenho das consultas.

Exemplo de Criação de Índice Composto:

```
CREATE INDEX idx_composto ON tabela (coluna1, coluna2);
```

2.Estratégias de Indexação Eficientes:

Além dos índices compostos, existem outras estratégias de indexação que podem ser aplicadas para otimizar consultas. Isso inclui o uso de índices únicos, índices bitmap, índices de texto completo e índices espaciais, dependendo das necessidades específicas da aplicação e dos padrões de acesso aos dados.

3. Análise de Plano de Execução de Consultas:

É importante analisar o plano de execução de consultas para identificar oportunidades de otimização de indexação. O plano de execução fornece informações detalhadas sobre como o banco de dados está processando uma consulta e pode ajudar a identificar gargalos de desempenho e oportunidades de melhoria.

4. Ajuste de Configurações do Servidor:

Além da indexação, o ajuste de configurações do servidor também pode influenciar significativamente o desempenho do banco de dados. Isso inclui ajustes de memória, configurações de cache, parâmetros de consulta e outras configurações relacionadas ao desempenho.

5.2 Utilização de índices compostos

Videoaula do tópico disponível no AVA:

Videoaula 18: Utilização de índices compostos

Os índices compostos são uma técnica avançada de indexação em bancos de dados relacionais, onde múltiplas colunas são combinadas em um único índice. Isso permite otimizar consultas que envolvem condições em mais de uma coluna.

1. Benefícios dos Índices Compostos:

- Melhora o desempenho de consultas que envolvem múltiplos critérios de seleção.
- Reduz o tempo de resposta em consultas complexas.
- Minimiza a sobrecarga de leitura no banco de dados.

2. Uso Eficiente de Índices Compostos:

Os índices compostos devem ser utilizados em colunas frequentemente utilizadas em cláusulas WHERE, JOIN e ORDER BY. É importante analisar o padrão de acesso aos dados e identificar quais combinações de colunas podem se beneficiar de um índice composto.

Exemplo de Uso de Índice Composto:

Suponha que temos uma tabela de pedidos com colunas id_cliente e data_pedido. Criar um índice composto nessas duas colunas pode acelerar consultas que filtram pedidos por cliente e data.

```
CREATE INDEX idx_pedido_cliente_data ON pedidos (id_cliente, data_pedido);
```

Os índices compostos são uma ferramenta poderosa para melhorar o desempenho de consultas em bancos de dados relacionais. Ao utilizar índices

compostos de forma eficiente, é possível reduzir o tempo de resposta e melhorar a experiência do usuário em sistemas de informação.

Aprenda Mais:

Querys indo de 1 hora a 1 minuto

Disponível em: https://www.youtube.com/watch?v=ZVwY9r8JVtk

5.3 Análise de plano de execução de consultas

Videoaula do tópico disponível no AVA:

Videoaula 19: Análise de plano de execução de consultas

A compreensão do plano de execução de consultas é fundamental para otimizar o desempenho de consultas em bancos de dados relacionais. O plano de execução fornece informações detalhadas sobre como o banco de dados está processando uma consulta e pode ser usado para identificar oportunidades de otimização.

1. Componentes do Plano de Execução:

- Árvore de execução: Representa a ordem das operações realizadas pelo banco de dados para processar a consulta.
- Métodos de acesso: Indica como o banco de dados acessará os dados, como por meio de índices ou varreduras de tabela.
- Operações de junção: Descreve como as tabelas são unidas entre si.
- Métodos de ordenação: Indica como os resultados são ordenados, se necessário.

2. Análise do Plano de Execução:

É importante analisar o plano de execução para identificar possíveis gargalos de desempenho e oportunidades de otimização. Isso pode incluir a adição de novos índices, ajuste das cláusulas WHERE e JOIN, ou reescrita da consulta para torná-la mais eficiente.

3. Ferramentas de Análise de Plano de Execução:

Muitos sistemas de gerenciamento de banco de dados fornecem ferramentas para visualizar o plano de execução de consultas. Estas ferramentas podem ser usadas para analisar o desempenho de consultas em tempo real e identificar áreas de melhoria.

Exemplo de Plano de Execução:

```
EXPLAIN SELECT * FROM tabela WHERE coluna = 'valor';
```

Esta consulta irá mostrar o plano de execução utilizado pelo banco de dados para recuperar os dados da tabela.

A compreensão do plano de execução de consultas é essencial para otimizar o desempenho de consultas em bancos de dados relacionais. Ao analisar cuidadosamente o plano de execução, é possível identificar e corrigir problemas de desempenho, garantindo assim uma experiência mais rápida e eficiente para os usuários do sistema.

Aprenda Mais:

Tudo sobre NLP: o que é? Quais os desafios? Disponível em: https://www.blip.ai/blog/tecnologia/nlp-processamento-linguagem-natural/

5.4 Ajuste de configurações do servidor para otimização de desempenho

Videoaula do tópico disponível no AVA:

Videoaula 20: Ajuste de configurações do servidor para otimização de desempenho

Os ajustes de configurações do servidor são uma parte essencial da otimização de desempenho em bancos de dados relacionais. Configurações adequadas podem melhorar o desempenho, a estabilidade e a segurança do servidor de banco de dados.

1. Ajustes de Memória:

Configurações relacionadas à alocação de memória podem afetar significativamente o desempenho do servidor. É importante configurar corretamente os buffers de memória, cache e área de troca para atender às necessidades do banco de dados e das aplicações.

2. Configurações de Cache:

O cache é utilizado para armazenar dados frequentemente acessados na memória, reduzindo assim o tempo de acesso ao disco. Configurar corretamente o cache do banco de dados pode melhorar drasticamente o desempenho de consultas.

3. Parâmetros de Consulta:

Além das configurações do servidor, é importante otimizar as consultas individuais para obter o melhor desempenho possível. Isso pode incluir o uso adequado de índices, reescrita de consultas e ajuste de parâmetros de consulta.

4. Monitoramento e Ajuste Contínuo:

O desempenho do servidor de banco de dados deve ser monitorado continuamente e ajustes devem ser feitos conforme necessário para garantir um desempenho ideal. Isso pode incluir ajustes sazonais, otimizações de consultas e atualizações de hardware.

Os ajustes de configurações do servidor são uma parte fundamental da otimização de desempenho em bancos de dados relacionais. Ao configurar adequadamente o servidor e monitorar continuamente o desempenho, é possível garantir um ambiente de banco de dados rápido, estável e confiável.

FINALIZAR

Chegamos ao fim desta jornada parabéns por concluir o nosso curso sobre Banco de Dados! Ao longo das cinco unidades, você teve a oportunidade de explorar os principais conceitos e práticas relacionadas ao uso do MySQL e SQL em geral. Vamos relembrar um pouco do que aprendemos:

Na Unidade 1, você mergulhou na instalação e configuração do MySQL, além de entender os fundamentos da integridade referencial e sua importância na manutenção da consistência dos dados.

Na Unidade 2, exploramos os comandos DDL (Data Definition Language) para criação, alteração e exclusão de estruturas de banco de dados, além de aprender a manipular dados usando comandos DML (Data Manipulation Language) como INSERT, UPDATE e DELETE.

Na Unidade 3, aprofundamos nossos conhecimentos em consultas SQL, desde a estrutura básica até operadores de filtro, ordenação de resultados e técnicas avançadas como junções, subconsultas e agregações.

Na Unidade 4, você descobriu como criar e usar Stored Procedures e Functions, ferramentas poderosas para modularizar e reutilizar lógica de banco de dados de forma eficiente.

Finalmente, na Unidade 5, exploramos estratégias de indexação e otimização para melhorar o desempenho do banco de dados, incluindo a análise de planos de execução de consultas e ajustes de configurações do servidor.

Ao concluir este curso, você está preparado para aplicar seus conhecimentos em projetos reais, seja desenvolvendo sistemas de informação, analisando dados ou administrando bancos de dados em ambientes corporativos.

Lembre-se sempre de praticar e explorar novos recursos e técnicas para aprimorar suas habilidades em Banco de Dados. O aprendizado é uma jornada contínua, e estamos aqui para ajudá-lo em cada passo do caminho.

Continue explorando e bons estudos

Prof. Esp. Murilo Almeida Pacheco

Sobre o autor

Murilo Almeida Pacheco possui MBA em Gerenciamento de Projetos pela Unialfa e é graduado em Sistemas de Informação pela Unialfa.

Sua trajetória acadêmica inclui também passagens como docente de graduação e pós-graduação em diversas instituições de ensino superior, como a Faculdade Fan-Padrão, Universidade Paulista, e Centro Universitário Alves Farias.

Com uma sólida experiência profissional, atua como Gestor de Projetos e Líder Técnico na fábrica de software da Universidade Federal de Goiás e como Analista de Sistemas Pleno na empresa M9 Solutions LLC situada em Miami, Flórida. Além de ter coordenado o departamento de tecnologia da informação do Centro Universitário Goyazes.

Referências Bibliográficas

Silberschatz, Abraham. **Sistema de banco de dados**, GEN LTC, 2020.

Machado, Felipe Nery Rodrigues. **Banco de dados: projeto e implementação**. 4. ed. São Paulo: Érica, 2020.

Elmasri, R.; Navathe, S. B.; Vieira, D. **Sistemas de banco de dados**. 7ª ed. São Paulo: Pearson, 2018.

Alves, William Pereira. **Banco de dados: teoria e desenvolvimento**. 2. ed. São Paulo: Érica, 2020.

Medeiros, Luciano Frontino de. **Banco de dados: princípios e prática**. 1. ed. São Paulo: Intersaberes, 2013.

Amadeu, Claudia Vicci (Org.). **Banco de dados**. São Paulo: Pearson, 2014.

Alves, William Pereira. **Banco de dados**. São Paulo: Érica, 2014.

Teorey, Toby et al. **Projeto e modelagem de banco de dados**. 2. ed. Rio de Janeiro: GEN LTC, 2013.