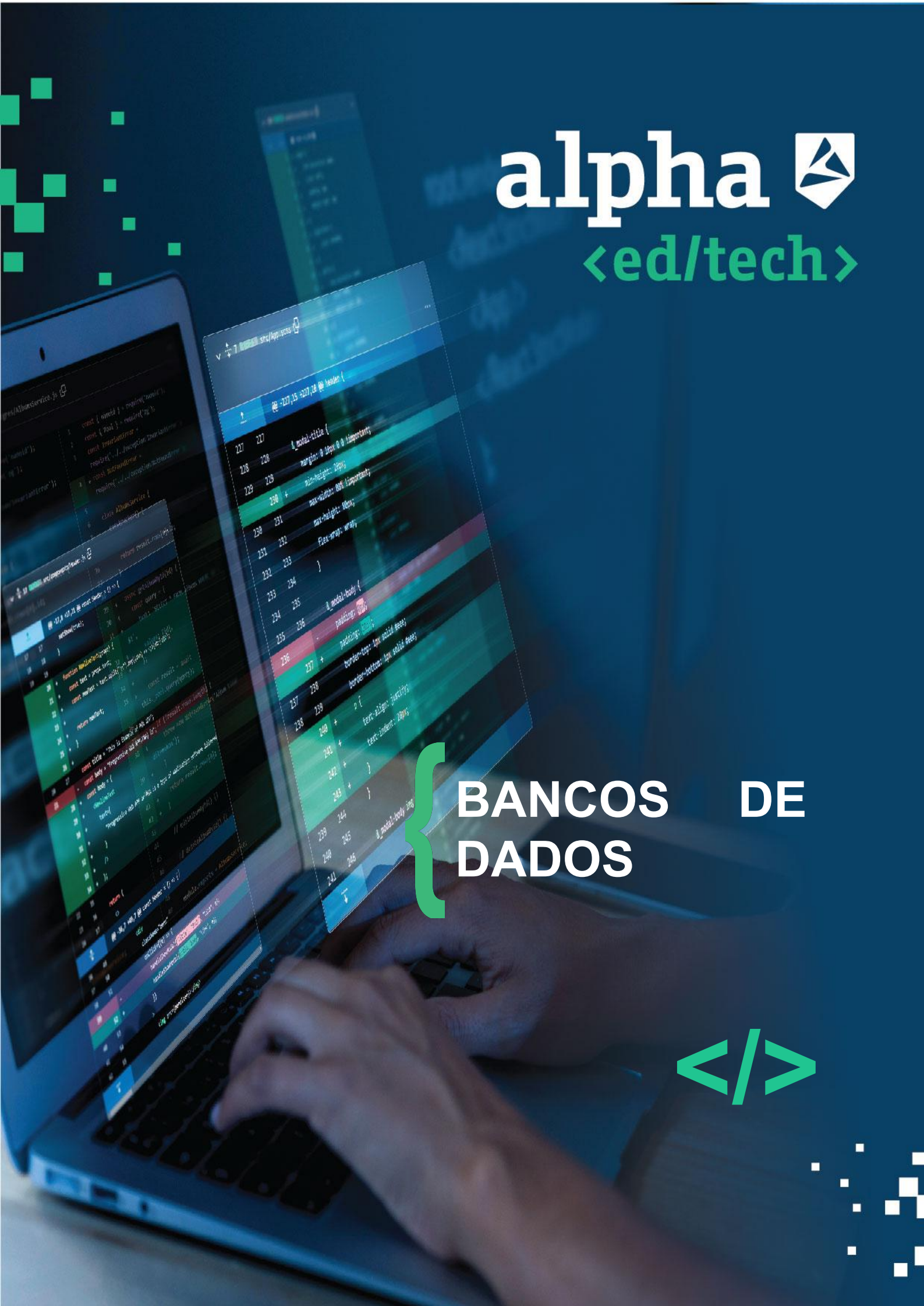
alpha
<ed/tech>
BANCOS DE DADOS
</>

Banco de dados Relacional

Cláusulas

Olá pessoal! Vamos explorar mais alguns conceitos fundamentais do nosso banco de dados que são:

- **CLÁUSULAS**
- **SUB-SELECT**
- **TRIGGERS e FUNCTIONS**

CONTRAINTS

As restrições, ou constraints, são regras aplicadas a uma tabela para limitar os tipos de dados que podem ser inseridos ou modificados em uma coluna.

Elas garantem a integridade e consistência dos dados em um banco de dados relacional.

Entre elas estão:

NOT NULL: essa restrição força uma coluna a não aceitar valores nulos. Por exemplo, na tabela "Pessoa", a coluna "Nome" pode ser definida como NOT NULL.

UNIQUE: essa restrição garante que todos os valores em uma coluna são únicos. Por exemplo, a coluna "CPF" em uma tabela de clientes deve ser única.

PRIMARY KEY: essa restrição identifica exclusivamente cada registro em uma tabela. Por exemplo, a coluna "ID" pode ser definida como PRIMARY KEY em uma tabela de funcionários.

FOREIGN KEY: essa restrição estabelece uma relação entre duas tabelas, onde a chave estrangeira em uma tabela se relaciona com a chave primária em outra tabela.

```
CREATE TABLE alunos (
      id SERIAL PRIMARY KEY,
      nome VARCHAR(50) UNIQUE NOT NULL,
      livro_id INTEGER FOREIGN KEY REFERENCES livros(id),
);
```

Neste exemplo, a tabela `alunos` possui diversas restrições aplicadas:

- A coluna `id` é a chave primária da tabela.
- As colunas `nome` é definido como NOT NULL e UNIQUE, para garantir que cada aluno tenha um nome único.
- O atributo `livro_id` possui a restrição uma chave estrangeira indicando um id da tabela de livros.

Essas restrições garantem a integridade e consistência dos dados na tabela `alunos`.

Banco de dados Relacional

Cláusulas

DEFAULT

A cláusula DEFAULT é usada para especificar um valor padrão para uma coluna em uma tabela. Essa cláusula é opcional e é adicionada durante a definição da estrutura da tabela.

O valor padrão especificado pela cláusula DEFAULT é inserido automaticamente na coluna quando nenhum valor é explicitamente fornecido durante uma operação de inserção de dados.

A cláusula DEFAULT permite que os desenvolvedores controlem o comportamento padrão de uma coluna quando nenhum valor é fornecido.

Isso pode ser útil em várias situações, como quando você deseja garantir que determinadas colunas sempre tenham um valor predefinido, ou para evitar erros quando dados não são fornecidos.

É importante observar que o valor padrão especificado pela cláusula DEFAULT deve ser um valor constante, ou seja, não pode ser uma expressão que depende de variáveis ou referências a outras colunas na mesma tabela.

Além disso, subconsultas não são permitidas na cláusula DEFAULT.

O tipo de dados do valor padrão deve corresponder ao tipo de dados da coluna em que está sendo definido.

Por exemplo, suponha que você tenha uma tabela chamada "clientes" com uma coluna "status" que indica o status do cliente. Você pode usar a cláusula DEFAULT para definir um valor padrão de "ativo" para essa coluna, como mostrado abaixo:

```
CREATE TABLE clientes (
    id INT PRIMARY KEY,
    nome VARCHAR(100),
    status VARCHAR(50) DEFAULT 'ativo'
);
```

Com essa definição, se nenhum valor for fornecido para a coluna "status" durante uma operação de inserção de dados, o valor padrão 'ativo' será automaticamente inserido. Isso pode simplificar o código e garantir consistência nos dados da tabela.

Banco de dados Relacional

Instruções

ALTER TABLE

Em PostgreSQL, a instrução ALTER TABLE é utilizada para realizar alterações na estrutura de uma tabela já existente no banco de dados. Com essa instrução, é possível adicionar, modificar ou excluir colunas, alterar restrições, renomear a tabela e executar outras operações relacionadas à sua estrutura.

A sintaxe básica da instrução ALTER TABLE é a seguinte:

```
ALTER TABLE nome_da_tabela
  [ALTER COLUMN nome da coluna tipo_de_alteração]
  [...outras alterações...];
```

Aqui estão dois exemplos simples de como usar o ALTER TABLE, um para adicionar uma nova coluna a uma tabela existente

```
ALTER TABLE minha_tabela
  ADD COLUMN nova_coluna VARCHAR(50);
```

Outro para adicionar restrições a uma tabela existente:

```
ALTER TABLE minha tabela
  ADD CONSTRAINT id_fk FOREIGN KEY REFERENCES outra_tabela(id);
```

Além de adicionar colunas, você pode usar o ALTER TABLE para:

- Alterar o tipo de dados de uma coluna existente.
- Renomear uma coluna.
- Excluir uma coluna.
- Adicionar ou remover restrições (como chaves primárias, chaves estrangeiras, restrições UNIQUE, etc.).
- Alterar o nome da tabela.
- Entre outras operações.

O ALTER TABLE é uma ferramenta poderosa que permite a você ajustar a estrutura das tabelas conforme necessário à medida que os requisitos do seu aplicativo evoluem. No entanto, é importante ter cuidado ao executar alterações em tabelas existentes, especialmente em ambientes de produção, para garantir a integridade dos dados e minimizar interrupções no funcionamento do sistema. Sempre faça backup dos dados importantes antes de realizar alterações significativas na estrutura das tabelas.

Banco de dados Relacional

Instruções

SUB-SELECT

Uma subconsulta, também conhecida como sub-seleção ou subquery, é uma consulta aninhada dentro de outra consulta SQL.

Essa subconsulta é usada para retornar um conjunto de resultados que é posteriormente usado na cláusula WHERE, HAVING ou em outra parte da consulta principal.

As subconsultas podem ser usadas em várias partes de uma consulta, como:

- WHERE Clause: Para filtrar linhas com base em condições específicas.
- FROM Clause: Para usar os resultados da subconsulta como uma tabela temporária.
- SELECT Clause: Para calcular valores com base nos resultados da subconsulta.
- INSERT, UPDATE, DELETE Statements: Para realizar operações com base nos resultados da subconsulta.

Aqui está um exemplo simples de como usar uma subconsulta na cláusula WHERE para filtrar dados com base em uma condição específica:

```
SELECT nome, idade
FROM clientes
WHERE idade > (SELECT AVG(idade) FROM clientes);
```

Neste exemplo, a subconsulta (SELECT AVG(idade) FROM clientes) é usada para calcular a idade média dos clientes e, em seguida, essa média é usada como critério na cláusula WHERE para selecionar apenas os clientes cuja idade é superior à média.

As subconsultas são úteis para realizar consultas mais complexas e permitem uma maior flexibilidade na manipulação de dados.

No entanto, é importante escrever subconsultas de forma eficiente, pois subconsultas mal escritas podem resultar em desempenho inadequado da consulta.

Além disso, é importante garantir que as subconsultas retornem um único valor ou uma única linha, conforme necessário para o contexto em que estão sendo usadas.

Banco de dados Relacional

Instruções

UPDATE

A instrução UPDATE é utilizada para modificar os valores de uma ou mais colunas em uma ou mais linhas de uma tabela existente. Essa instrução é essencial para atualizar registros em uma tabela de banco de dados de acordo com determinados critérios.

A sintaxe básica da instrução UPDATE é a seguinte:

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

Nesta sintaxe:

- O nome_da_tabela é o nome da tabela que será atualizada.
- As coluna1, coluna2, etc., são as colunas que serão atualizadas.
- Os valor1, valor2, etc., são os novos valores que serão atribuídos às colunas especificadas.
- O WHERE é uma cláusula opcional que define as condições que as linhas devem satisfazer para que a atualização ocorra. Se esta cláusula for omitida, todas as linhas da tabela serão atualizadas.

Aqui está um exemplo simples de como usar a instrução UPDATE:

```
UPDATE clientes
SET status = 'Ativo'
WHERE id = 1;
```

Neste exemplo, estamos atualizando o valor da coluna status para 'Ativo' para o cliente com id igual a 1.

Você também pode atualizar várias colunas simultaneamente e usar condições mais complexas na cláusula WHERE, conforme necessário para suas necessidades específicas.

É importante ter cuidado ao usar a instrução UPDATE, especialmente sem uma cláusula WHERE adequada, pois isso pode resultar na modificação de todas as linhas da tabela, o que pode não ser desejado.

Sempre teste suas consultas UPDATE em ambientes de teste antes de executá-las em um ambiente de produção para evitar alterações indesejadas nos dados.

Banco de dados Relacional

Instruções

DELETE

A instrução DELETE é usada para remover uma ou mais linhas de uma tabela de banco de dados. Essa instrução é fundamental para gerenciar o conteúdo de uma tabela, permitindo que você exclua registros com base em condições específicas.

Aqui está a sintaxe básica da instrução DELETE:

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

Nesta sintaxe:

- O nome_da_tabela: é o nome da tabela da qual você deseja excluir registros.
- O WHERE: é uma cláusula opcional que permite especificar uma condição que os registros devem satisfazer para serem excluídos. Se você omitir a cláusula WHERE, todos os registros da tabela serão excluídos.

Aqui está um exemplo simples de como usar a instrução DELETE:

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

Neste exemplo, estamos excluindo o cliente cujo id é igual a 1 da tabela clientes.

Você também pode usar a cláusula WHERE para excluir registros com base em condições mais complexas. Por exemplo:

```
DELETE FROM pedidos
WHERE data_pedido < '2023-01-01';
```

Neste exemplo, estamos excluindo todos os pedidos feitos antes de 1 de janeiro de 2023 da tabela pedidos.

É importante estar ciente de que a instrução DELETE é uma operação irreversível e, uma vez executada, os registros excluídos não podem ser recuperados sem um backup adequado dos dados. Portanto, sempre certifique-se de que está excluindo os registros corretos e teste suas instruções de exclusão em um ambiente de teste antes de executá-las em um ambiente de produção.

Além disso, em situações em que você precisa excluir uma grande quantidade de dados, pode ser útil dividir a operação de exclusão em lotes menores para evitar bloqueios prolongados da tabela e impactos no desempenho do sistema. Isso pode ser feito usando instruções DELETE em combinação com cláusulas LIMIT e OFFSET ou usando uma abordagem de exclusão em lotes dentro de um loop controlado.

Banco de dados Relacional

Abordagens

SOFT DELETE

O conceito de "soft delete" refere-se a uma abordagem de exclusão de registros em um banco de dados onde, em vez de remover fisicamente os registros da tabela, você marca os registros como "excluídos" por meio de uma coluna específica que indica o status de exclusão. Essa abordagem é frequentemente usada quando você deseja manter um histórico dos registros excluídos ou quando precisa manter a integridade referencial em sistemas complexos.

Aqui está como você pode implementar um "soft delete" em PostgreSQL:

- Adicionar uma coluna para indicar o status de exclusão: Você precisará adicionar uma coluna à sua tabela para armazenar o status de exclusão. Geralmente, essa coluna é uma coluna booleana (true/false) ou uma coluna de data/hora que registra quando o registro foi excluído.

```
ALTER TABLE sua tabela
ADD COLUMN excluido BOOLEAN DEFAULT FALSE;
```

ou

```
ALTER TABLE sua tabela
ADD COLUMN data_exclusao TIMESTAMP;
```

- Atualizar registros marcando-os como excluídos: Em vez de excluir fisicamente os registros, você atualiza o valor na coluna que indica o status de exclusão.

```
UPDATE sua_tabela
SET excluido = TRUE
WHERE id = 1;
```

ou

```
UPDATE sua_tabela
SET data exclusao = CURRENT_TIMESTAMP
WHERE id = 1;
```

- Consultar registros excluídos: Ao consultar os dados, você incluirá uma condição na cláusula WHERE para excluir os registros marcados como excluídos.

A principal vantagem do "soft delete" é que você mantém um histórico de todos os registros excluídos, o que pode ser útil para auditorias, recuperação de dados ou para manter a integridade referencial em sistemas que dependem desses registros. No entanto, é importante lembrar que, mesmo com o "soft delete", os registros marcados como excluídos ainda ocuparão espaço no banco de dados e devem ser gerenciados adequadamente para evitar o inchaço da tabela.

Além disso, ao usar "soft delete", é essencial garantir que todas as consultas de recuperação de

Banco de dados Relacional

Abordagens

SOFT DELETE

- Consultar registros excluídos: Ao consultar os dados, você incluirá uma condição na cláusula WHERE para excluir os registros marcados como excluídos.

```
SELECT *
FROM sua tabela
WHERE excluido = FALSE;
```

ou

```
SELECT *
FROM sua_tabela
WHERE data_exclusao IS NULL;
```

A principal vantagem do "soft delete" é que você mantém um histórico de todos os registros excluídos, o que pode ser útil para auditorias, recuperação de dados ou para manter a integridade referencial em sistemas que dependem desses registros.

No entanto, é importante lembrar que, mesmo com o "soft delete", os registros marcados como excluídos ainda ocuparão espaço no banco de dados e devem ser gerenciados adequadamente para evitar o inchaço da tabela.

Além disso, ao usar "soft delete", é essencial garantir que todas as consultas de recuperação de dados incluam a condição adequada para filtrar os registros excluídos, a fim de evitar que esses registros sejam incluídos inadvertidamente nos resultados das consultas.

Banco de dados Relacional

Abordagens

TRIGGERS

ma trigger (gatilho) é uma função especial que é executada automaticamente quando ocorre um evento específico em uma tabela. Esses eventos podem incluir operações como inserção, atualização ou exclusão de registros. As triggers permitem que você automatize tarefas comuns no banco de dados, como validações de dados, auditorias, manutenção de integridade referencial, replicação de dados, entre outros.

Aqui estão os principais componentes de uma trigger em PostgreSQL:

1. **Evento**: Um evento desencadeador especifica a ação que desencadeia a execução da trigger. Os eventos comuns incluem INSERT, UPDATE e DELETE.

2. **Momento de execução**: O momento de execução de uma trigger pode ser definido como BEFORE ou AFTER o evento que a dispara.

3. **BEFORE**: A trigger é acionada antes que a operação de banco de dados seja executada.

4. **AFTER**: A trigger é acionada após a conclusão bem-sucedida da operação de banco de dados.

5. **Condição**: A condição, também conhecida como cláusula WHEN, é uma expressão opcional que determina se a trigger será executada com base em certas condições. Se a condição for verdadeira, a trigger será executada; caso contrário, não será.

6. **Função de trigger**: A função de trigger é a rotina PL/pgSQL (ou em outra linguagem suportada pelo PostgreSQL) que define o comportamento da trigger. Esta função é executada quando a trigger é acionada e pode incluir lógica de programação complexa.

Aqui está um exemplo de criação de uma trigger em PostgreSQL:

```
CREATE OR REPLACE FUNCTION nome_da_funcao_trigger()
RETURNS TRIGGER AS $$
BEGIN
    -- lógica da trigger aqui
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER nome_do_gatilho
AFTER INSERT ON nome_da_tabela
FOR EACH ROW
EXECUTE FUNCTION nome_da_funcao_trigger();
```

Banco de dados Relacional

Abordagens

TRIGGERS

```
CREATE OR REPLACE FUNCTION nome_da_funcao_trigger()
RETURNS TRIGGER AS $$
BEGIN
    -- lógica da trigger aqui
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER nome_do_gatilho
AFTER INSERT ON nome_da_tabela
FOR EACH ROW
EXECUTE FUNCTION nome_da_funcao_trigger();
```

Neste exemplo:

- A função de trigger nome_da_funcao_trigger() é definida usando a linguagem PL/pgSQL. Esta função será executada sempre que a trigger for acionada.
- A trigger nome_do_gatilho é definida para ser acionada após a inserção de novos registros na tabela nome_da_tabela.
- A função de trigger nome_da_funcao_trigger() é associada à trigger nome_do_gatilho, especificando que ela será executada sempre que a trigger for acionada.
- As triggers são poderosas ferramentas de automação em PostgreSQL, mas devem ser usadas com cuidado, pois podem afetar significativamente o desempenho do banco de dados se forem mal projetadas ou mal utilizadas. É importante considerar o impacto das triggers no desempenho do sistema e garantir que elas sejam testadas completamente antes de serem implantadas em um ambiente de produção.

Aqui está um exemplo de criação de uma trigger em PostgreSQL:

FUNCTIONS

As funções são blocos de código SQL nomeados e armazenados no banco de dados, que podem ser invocados e reutilizados em várias consultas.

Essas funções permitem que você encapsule a lógica de negócios e a lógica de processamento de dados em um único local, facilitando a manutenção, reutilização e modularidade do código. As funções em PostgreSQL podem ser escritas em várias linguagens de programação, como PL/pgSQL, SQL, Python, entre outras.

Banco de dados Relacional

Abordagens

FUNCTIONS

Aqui estão alguns dos principais aspectos das funções em PostgreSQL:

Criação de Funções

- As funções podem ser criadas usando a linguagem de definição de dados (DDL) CREATE FUNCTION.
- Você pode especificar o nome da função, os parâmetros de entrada e saída, o tipo de retorno e a linguagem de programação usada para escrever a função.

Parâmetros

- As funções podem aceitar zero ou mais parâmetros de entrada.
- Os parâmetros podem ser de entrada, de saída ou de entrada e saída, dependendo da necessidade da função.
- Os parâmetros são definidos na declaração da função e podem ser usados dentro do corpo da função.
- Corpo da Função: O corpo da função contém a lógica de processamento de dados ou lógica de negócios.
- Isso pode incluir instruções SQL, operações de controle de fluxo (como IF, ELSE, CASE), manipulação de exceções, entre outros.
- Retorno de Valores:
- As funções podem retornar um valor único (escalar) ou uma tabela de resultados.
- O tipo de retorno da função é especificado na declaração da função.
- Para funções que retornam uma tabela, você pode usar a cláusula RETURNS TABLE ou RETURNS SETOF.

Execução da Função

- As funções podem ser executadas como parte de uma consulta SQL, como uma subconsulta ou diretamente através do comando SELECT.
- Você pode chamar uma função usando seu nome e fornecendo os argumentos necessários.

Banco de dados Relacional

Abordagens

FUNCTIONS

Aqui está um exemplo simples de criação e execução de uma função em PostgreSQL:

```
CREATE OR REPLACE FUNCTION soma(a INT, b INT)
RETURNS INT AS $$
BEGIN
    RETURN a + b;
END;
$$ LANGUAGE plpgsql;

-- Chamando a função
SELECT soma(5, 3); -- Retorna 8
```

Neste exemplo, estamos criando uma função chamada soma que aceita dois parâmetros de entrada a e b, e retorna a soma desses dois valores. A função é escrita em PL/pgSQL e retorna um valor inteiro.

As funções em PostgreSQL são uma maneira poderosa de encapsular lógica de negócios e reutilizar código em várias partes do banco de dados. Elas ajudam a tornar o código mais modular, legível e fácil de manter. No entanto, é importante usar funções com moderação e considerar o impacto no desempenho, especialmente em cenários de alta carga de trabalho.

TRANSACTIONS

Uma transação é uma unidade lógica de trabalho que consiste em uma ou mais operações de banco de dados que devem ser executadas de forma atômica, consistente, isolada e durável, comumente conhecidas como propriedades ACID:

Atomicidade: Isso significa que todas as operações em uma transação devem ser tratadas como uma única unidade. Se uma operação falhar, todas as operações anteriores devem ser revertidas para garantir que o banco de dados permaneça em um estado consistente.

Consistência: Uma transação deve levar o banco de dados de um estado consistente para outro estado consistente. Isso significa que todas as restrições de integridade referencial, restrições de chave primária, restrições de chave estrangeira, etc., devem ser mantidas antes e após a transação.

Isolamento: Cada transação deve ser isolada das outras transações concorrentes. Isso garante que os resultados de uma transação não sejam afetados por outras transações que estejam ocorrendo simultaneamente no banco de dados.

Durabilidade: Após a confirmação de uma transação (commit), as alterações feitas por essa transação devem ser permanentes e resistir a falhas de hardware ou software. Isso significa que as alterações devem ser gravadas permanentemente no armazenamento persistente do banco de dados.

Banco de dados Relacional

Abordagens

TRANSACTIONS

As transações são iniciadas implicitamente ao executar uma única instrução SQL ou explicitamente usando as palavras-chave BEGIN, COMMIT e ROLLBACK. A maioria das operações em PostgreSQL é realizada dentro de uma transação, mesmo que não seja explicitamente declarada.

Aqui está um exemplo de uso explícito de transações em PostgreSQL:

```
BEGIN; -- Inicia a transação

UPDATE conta SET saldo = saldo - 100 WHERE id = 1; -- Operação 1
UPDATE conta SET saldo = saldo + 100 WHERE id = 2; -- Operação 2

COMMIT; -- Confirma a transação
```

Neste exemplo:

- **BEGIN** inicia a transação.
- **UPDATE** são as operações realizadas dentro da transação.
- **COMMIT** confirma e encerra a transação com sucesso, aplicando todas as operações à base de dados.

Se ocorrer algum erro durante as operações dentro da transação, você pode usar ROLLBACK para reverter as alterações e cancelar a transação:

```
BEGIN;

-- Operações que podem falhar

ROLLBACK; -- Cancela a transação e reverte as operações
```

Transações são fundamentais para garantir a integridade e a consistência dos dados em um banco de dados relacional como PostgreSQL, especialmente em ambientes onde múltiplos usuários estão acessando e modificando os dados simultaneamente.

Banco de dados Relacional

Abordagens

TRANSACTIONS

Se ocorrer algum erro durante as operações dentro da transação, você pode usar ROLLBACK para reverter as alterações e cancelar a transação:

```
BEGIN;

-- Operações que podem falhar

ROLLBACK; -- Cancela a transação e reverte as operações
```

Transações são fundamentais para garantir a integridade e a consistência dos dados em um banco de dados relacional como PostgreSQL, especialmente em ambientes onde múltiplos usuários estão acessando e modificando os dados simultaneamente.

Obrigado e até a próxima aula!