Guia de consulta rápido

SQL

growdev

Sumário

Termos de Uso

Propriedade Growdev

Todo o conteúdo deste documento é propriedade da Growdev. O mesmo pode ser utilizado livremente para estudo pessoal.
É proibida qualquer utilização desse material que não se enquadre nas condições acima sem o prévio consentimento formal, por escrito, da Growdev. O uso indevido está sujeito às medidas legais cabíveis.

Intro dução

Linguagem SQL

A Linguagem SQL (Structured Query Language) é usada para gerenciar e consultar bancos de dados relacionais. Aqui está um guia de consulta rápida para as operações mais comuns em SQL.

Este guia oferece uma visão geral rápida das operações SQL mais comuns. A sintaxe e as capacidades exatas podem variar de acordo com o sistema de gerenciamento de banco de dados que você está usando. Certifique-se de consultar a documentação relevante do seu DBMS para obter informações detalhadas.

Para uma compreensão mais profunda e aplicação prática, é recomendável estudar cada conceito em detalhes e praticar com exemplos reais.

growdev

Banco de dados Relacional

Conceitos fundamentais

Um banco de dados relacional é uma estrutura de armazenamento e organização de dados que utiliza tabelas para representar informações inter-relacionadas:

Banco de Dados Relacional:
Um sistema que organiza dados em tabelas com linhas (registros) e colunas (campos), permitindo relacionamentos entre tabelas.

Tabela:
Uma estrutura fundamental que armazena dados em linhas e colunas. Cada tabela tem um nome exclusivo e é composta por registros (ou tuplas) e campos.

Registro:
Também conhecido como tupla, é uma única entrada na tabela que contém dados específicos relacionados a um objeto ou entidade.

Campo:
Uma coluna individual em uma tabela que armazena um tipo de dado específico, como texto, número ou data.

Chave Primária (PK):
Um campo ou conjunto de campos que identifica de forma exclusiva cada registro em uma tabela. Garante integridade e unicidade dos dados.

Chave Estrangeira (FK):
Um campo em uma tabela que estabelece um relacionamento com a chave primária de outra tabela. Usado para criar relacionamentos entre tabelas.

growdev

Conceitos Fundamentais

Relacionamento:

A associação lógica entre tabelas usando chaves primárias e estrangeiras para representar conexões entre entidades.

Consulta (Query):

Uma solicitação para recuperar, modificar ou manipular dados de um banco de dados. Geralmente escrito em SQL (Structured Query Language).

SQL (Structured Query Language):

Uma linguagem utilizada para gerenciar e consultar bancos de dados relacionais.

Normalização:

O processo de projetar tabelas de banco de dados para minimizar a redundância e melhorar a eficiência, dividindo dados em tabelas menores e relacionadas.

Desnormalização:

O processo de otimizar um banco de dados ao permitir um certo grau de redundância de dados para melhorar o desempenho de consultas.

Índice:

Uma estrutura de dados que melhora a velocidade de busca e recuperação de dados, criando uma lista ordenada de valores de um ou mais campos.

Transação:

Uma sequência de operações que são tratadas como uma unidade única, garantindo a consistência e a integridade dos dados.

ACID:

Acrônimo para Atomicidade, Consistência, Isolamento e Durabilidade - um conjunto de propriedades garantidas em bancos de dados transacionais para garantir a confiabilidade das operações.

JOIN:

Uma operação que combina registros de duas ou mais tabelas com base em um critério comum, permitindo a recuperação de informações relacionadas.

growdev

Linguagens
DML, DDL, DCL, DTL e DQL

Cada uma dessas linguagens (DML, DDL, DCL, DTL e DQL) desempenha um papel importante no gerenciamento e manipulação de bancos de dados usando a linguagem SQL. É fundamental entender esses conceitos para usar efetivamente o SQL para interagir com os dados em um banco de dados relacional.

DML
(Data Manipulation Language)

A Linguagem de Manipulação de Dados (DML) é usada para inserir, atualizar, recuperar e excluir dados de um banco de dados. As principais operações DML são:

- **INSERT:** Usado para adicionar novos registros a uma tabela.
- **UPDATE:** Utilizado para modificar registros existentes em uma tabela.
- **DELETE:** Usado para excluir registros de uma tabela.
- **SELECT:** Usado para recuperar dados de uma ou mais tabelas.

Exemplos DML:

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

UPDATE produtos SET preco = 25.99 WHERE id = 123;
DELETE FROM pedidos WHERE status = 'cancelado';

SELECT nome, telefone FROM contatos WHERE cidade = 'São Paulo';
```

growdev

DDL (Data Definition Language)

A Linguagem de Definição de Dados (DDL) é usada para definir e gerenciar estruturas de banco de dados, como tabelas, índices e restrições. As principais operações DDL são:

- **CREATE:** Utilizado para criar objetos do banco de dados, como tabelas e índices.
- **ALTER:** Usado para modificar a estrutura de objetos existentes.
- **DROP: U**tilizado para excluir objetos do banco de dados.

Exemplos:

```
CREATE TABLE funcionarios (
    id INT PRIMARY KEY,
    nome VARCHAR(50),
    cargo VARCHAR(30)
);

ALTER TABLE clientes ADD COLUMN data_registro DATE;

DROP TABLE pedidos;
```

DCL (Data Control Language)

A Linguagem de Controle de Dados (DCL) é usada para controlar o acesso aos dados no banco de dados. As principais operações DCL são:

- **GRANT:** Usado para conceder privilégios a usuários e papéis.
- **REVOKE:** Utilizado para remover privilégios concedidos anteriormente.

Exemplos:

GRANT SELECT, INSERT ON produtos TO vendedor;

REVOKE UPDATE ON pedidos FROM atendente;

growdev

DTL (Data Transaction Language)

A Linguagem de Transação de Dados (DTL) é usada para controlar transações no banco de dados, permitindo que você inicie, confirme ou reverta transações.

- **COMMIT:** Confirma uma transação, tornando suas alterações permanentes.
- **ROLLBACK:** Reverte uma transação, desfazendo suas alterações.
- **SAVEPOINT:** Cria um ponto de salvamento em uma transação para permitir o rollback para esse ponto específico.
- **SET TRANSACTION:** Define características de uma transação, como isolamento e controle de concorrência.

Exemplos:

```
BEGIN;
UPDATE saldo_conta SET valor = valor - 100 WHERE conta_id = 123;

SAVEPOINT ponto1;
UPDATE saldo_conta SET valor = valor + 50 WHERE conta_id = 456;

ROLLBACK TO ponto1;

COMMIT;
```

DQL (Data Query Language)

A Linguagem de Consulta de Dados (DQL) é usada para recuperar informações específicas do banco de dados. A operação principal em DQL é o comando SELECT, que é usado para consultar dados.

Exemplos:

SELECT nome, idade FROM alunos WHERE curso = 'Engenharia';

SELECT COUNT(*) FROM produtos WHERE preço > 100;

growdev

Cláusulas em SQL

As cláusulas são partes de uma consulta SQL que permitem filtrar, ordenar e limitar os resultados. Algumas cláusulas importantes incluem:

- **FROM:** Indica a tabela da qual você está recuperando os dados.
- **WHERE:** Define as condições de filtro para os registros que serão recuperados.
- **ORDER BY:** Ordena os resultados com base em uma ou mais colunas.
- **GROUP BY:** Agrupa os resultados com base em uma ou mais colunas.
- **HAVING:** Filtra os resultados de grupos criados pelo GROUP BY.
- **LIMIT:** Limita o número de registros retornados na consulta.

Exemplos:

```
SELECT nome, idade
FROM alunos
WHERE curso = 'Engenharia'
ORDER BY idade DESC
LIMIT 10;
```

growdev

Operadores Lógicos

Os operadores lógicos são usados para combinar condições em uma cláusula WHERE. Os principais operadores lógicos são:

- **AND:** Retorna true se todas as condições forem verdadeiras.
- **OR:** Retorna true se pelo menos uma das condições for verdadeira.
- **NOT:** Inverte o valor da condição, ou seja, retorna true se a condição for falsa e vice-versa.

Exemplos:

```
SELECT nome, idade
FROM alunos
WHERE curso = 'Engenharia' AND idade > 20;
```

growdev

Operadores Relacionais

Os operadores relacionais são usados para comparar valores em condições. Os principais operadores relacionais são:

- =: Igual a.
- <: Menor que.
- >: Maior que.
- <=: Menor ou igual a.
- >=: Maior ou igual a.
- <> ou !=: Diferente de.

Exemplos:

```
SELECT nome, salario
FROM funcionarios
WHERE salario >= 50000 AND salario <= 80000;
```

growdev

Combinando Cláusulas, Operadores Lógicos e Operadores Relacionais

Você pode combinar cláusulas, operadores lógicos e operadores relacionais para criar consultas mais complexas e específicas.

Exemplo:

```
SELECT nome, idade
FROM alunos
WHERE curso = 'Engenharia' AND (idade > 20
OR idade < 18);
```

Nesse exemplo, estamos buscando alunos que estejam cursando Engenharia e tenham mais de 20 anos ou menos de 18 anos.

Compreender e utilizar corretamente as cláusulas, operadores lógicos e operadores relacionais é essencial para criar consultas SQL eficazes e obter os resultados desejados ao trabalhar com bancos de dados relacionais.

growdev

Operador LIKE

O LIKE é usado em consultas SQL para realizar correspondências de padrões em valores de texto. Ele é frequentemente usado em conjunto com o operador % (porcentagem) para representar zero, um ou vários caracteres, e o operador _ (sublinhado) para representar um único caractere. O LIKE é útil quando você deseja pesquisar valores que correspondam a um padrão específico, como encontrar palavras que contenham certos caracteres ou sequências.

growdev

Sintaxe Básica:

```
SELECT coluna
FROM tabela
WHERE coluna LIKE padrão;
```

Exemplo:

Suponha que você tenha uma tabela chamada produtos e deseje encontrar todos os produtos cujo nome comece com "Camiseta". Você pode usar o comando LIKE com o operador % da seguinte maneira:

```
SELECT nome
FROM produtos
WHERE nome LIKE
'Camiseta%';
```

Neste exemplo, a consulta retornará todos os produtos cujo nome começa com "Camiseta", independentemente dos caracteres que seguem essa parte do texto.

Exemplo:

Encontrar todos os produtos que têm "azul" em algum lugar no nome:

```
SELECT nome
FROM produtos
WHERE nome LIKE '%azul%';
```

Exemplo:

Encontrar todos os produtos que terminam com "calça":

```
SELECT nome
FROM produtos
WHERE nome LIKE '%calça';
```

Exemplo:

Encontrar todos os produtos que começam com qualquer caractere e depois têm "lápis":

```
SELECT nome
FROM produtos
WHERE nome LIKE '_lápis';
```

O comando LIKE é útil para realizar buscas flexíveis em valores de texto, permitindo que você recupere dados que correspondam a padrões específicos, independentemente das posições exatas dos caracteres no texto.

Funções de Agregação em SQL

COUNT

A função COUNT é usada para contar o número de registros em uma coluna ou tabela.

SELECT COUNT(*) FROM produtos;
SELECT COUNT(distinct categoria) FROM produtos;

SUM

A função SUM é usada para somar os valores de uma coluna numérica.

SELECT SUM(total_vendas) FROM pedidos;

AVG

A função AVG é usada para calcular a média dos valores em uma coluna numérica.

SELECT AVG(salario) FROM funcionarios;

growdev

Funções de Agregação em SQL

MAX

A função MAX é usada para encontrar o valor máximo em uma coluna.

SELECT MAX(preco) FROM produtos;

MIN

A função MIN é usada para encontrar o valor mínimo em uma coluna.

SELECT MIN(data_registro) FROM clientes;

growdev

Exemplo de uso de várias funções de agregação

```
SELECT COUNT(*) AS total_pedidos,
    SUM(valor_total) AS total_vendas,
    AVG(valor_total) AS
media_vendas,
    MAX(valor_total) AS maior_venda,
    MIN(valor_total) AS menor_venda
FROM pedidos;
```

Essas funções de agregação podem ser usadas em conjunto com outras cláusulas SQL, como WHERE e GROUP BY, para realizar cálculos específicos em conjuntos de dados selecionados ou agrupados. Elas são essenciais para extrair informações estatísticas e resumidas de grandes conjuntos de dados em bancos de dados relacionais.

growdev

Criar Tabelas

Criar Tabelas

```
CREATE TABLE nome_da_tabela (
    coluna1 tipo_de_dado,
    coluna2 tipo_de_dado,
```

Exemplo:

```
CREATE TABLE alunos (
    id INT PRIMARY KEY,
    nome VARCHAR(50),
    idade INT
);
```

Inserir Dados:

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

Exemplo:

```
INSERT INTO alunos (id, nome,
idade)
VALUES (1, 'Maria', 25);
```

growdev

Consultar Dados

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

Exemplo:

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

Atualizar Dados

```
UPDATE nome_da_tabela
SET coluna1 = novo_valor1, coluna2 = novo_valor2, ...
WHERE condição;
```

Exemplo:

```
UPDATE alunos
SET idade = 26
WHERE nome = 'Maria';
```

Excluir Dados

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

Exemplo:

```
DELETE FROM alunos
WHERE nome = 'Maria';
```

Ordenar Resultados

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

Exemplo:

```
SELECT nome, idade
FROM alunos
ORDER BY idade DESC;
```

Filtrar com Operadores Lógicos

```
SELECT coluna1, coluna2, ...
FROM nome_da_tabela
WHERE coluna1 = valor1 AND coluna2 > valor2 OR
coluna3 LIKE 'padrão';
```

Exemplo:

```
SELECT nome, idade
FROM alunos
WHERE idade >= 18 AND idade <= 30 OR nome LIKE
'Jo%';
```

Usar Funções Agregadas

```
SELECT COUNT(coluna), AVG(coluna), SUM(coluna),
MAX(coluna), MIN(coluna)
FROM nome_da_tabela;
```

Exemplo:

```
SELECT COUNT(*) AS total_alunos,
    AVG(idade) AS media_idade,
    MAX(idade) AS maior_idade
FROM alunos;
```

Juntar Tabelas

É possível realizar diferentes tipos de junções (joins) para combinar dados de várias tabelas.

INNER JOIN

Retorna apenas às linhas em que há uma correspondência entre as tabelas envolvidas. Ou seja, apenas as linhas com valores coincidentes nas colunas especificadas serão retornadas.

Exemplo:

```
SELECT orders.order_id, customers.customer_name
FROM orders
INNER JOIN customers ON orders.customer_id = customers.customer_id;
```

LEFT JOIN (ou LEFT OUTER JOIN)

Retorna todas as linhas da tabela à esquerda e as linhas correspondentes da tabela à direita. Se não houver correspondência na tabela à direita, serão retornados valores nulos.

Exemplo:

```
SELECT customers.customer_id, customers.customer_name, orders.order_id
FROM customers
LEFT JOIN orders ON customers.customer_id = orders.customer_id;
```

RIGHT JOIN (ou RIGHT OUTER JOIN)

É semelhante ao LEFT JOIN, mas retorna todas as linhas da tabela à direita e as linhas correspondentes da tabela à esquerda. Se não houver correspondência na tabela à esquerda, serão retornados valores nulos

Exemplo:

```
SELECT orders.order_id, orders.order_date, customers.customer_name
FROM orders
RIGHT JOIN customers ON orders.customer_id = customers.customer_id;
```

FULL JOIN (ou FULL OUTER JOIN)

Retorna todas as linhas de ambas as tabelas, combinando os resultados do LEFT JOIN e RIGHT JOIN. Se não houver correspondência em uma tabela, os valores nulos serão retornados na outra tabela.

Exemplo:

```
SELECT customers.customer_id, customers.customer_name, orders.order_id
FROM customers
FULL JOIN orders ON customers.customer_id = orders.customer_id;
```

growdev

Juntar Tabelas

CROSS JOIN

O CROSS JOIN retorna o produto cartesiano de todas as linhas das tabelas envolvidas, ou seja, todas as combinações possíveis entre as linhas.

Exemplo:

```
SELECT customers.customer_name, products.product_name
FROM customers
CROSS JOIN products;
```

É importante lembrar que, ao realizar joins, é necessário identificar corretamente as colunas nas cláusulas ON, para garantir a integridade dos resultados. Cada tipo de join serve a propósitos diferentes e deve ser escolhido conforme a necessidade específica de cada consulta.

Agrupar Resultados (GROUP BY)

```
SELECT coluna1, AVG(coluna2)
FROM nome_da_tabela
GROUP BY coluna1;
```

Exemplo:

```
SELECT curso_id, COUNT(*) AS total_alunos
FROM alunos
GROUP BY curso_id;
```

Filtros com HAVING

```
SELECT coluna1, AVG(coluna2)
FROM nome_da_tabela
GROUP BY coluna1
HAVING AVG(coluna2) > valor;
```

Exemplo:

```
SELECT curso_id, AVG(idade) AS media_idade
FROM alunos
GROUP BY curso_id
HAVING AVG(idade) > 25;
```

Tabelas

Subconsultas (Subqueries)

SELECT coluna1
FROM nome_da_tabela
WHERE coluna2 IN (SELECT coluna2 FROM outra_tabela WHERE condição);

Exemplo:
SELECT nome
FROM alunos
WHERE curso_id IN (SELECT id FROM cursos WHERE nome = 'Engenharia');

Alterar Tabela (ALTER)

ALTER TABLE nome_da_tabela
ADD coluna_nova tipo_de_dado;

ALTER TABLE nome_da_tabela
MODIFY coluna tipo_de_dado;

ALTER TABLE nome_da_tabela
DROP COLUMN coluna;

Exemplo:
ALTER TABLE alunos
ADD COLUMN curso_id INT;

ALTER TABLE alunos
MODIFY COLUMN nome VARCHAR(60);

ALTER TABLE alunos
DROP COLUMN idade;

growdev

Criar Índices

CREATE INDEX index_nome ON nome_da_tabela (coluna);

Exemplo:

CREATE INDEX idx_nome ON alunos (nome);

Exemplos de Funções

SELECT CONCAT(primeiro_nome, ' ', sobrenome) AS
nome_completo
FROM nome_da_tabela;

SELECT DATE_FORMAT(data, '%d/%m/%Y') AS data_formatada
FROM nome_da_tabela;

Exemplo:
SELECT CONCAT(primeiro_nome, ' ', sobrenome) AS
nome_completo
FROM funcionarios;

SELECT DATE_FORMAT(data_pedido, '%d/%m/%Y') AS
data_formatada
FROM pedidos

Expressão condicional CASE

Permite realizar diferentes ações com base em condições específicas. Ele pode ser usado para executar avaliações condicionais dentro das instruções SQL, o que pode ser útil para realizar transformações nos dados, gerar colunas calculadas ou aplicar lógica condicional.

A estrutura básica de uma cláusula CASE em SQL é a seguinte:

```
CASE
  WHEN condição_1 THEN resultado_1
  WHEN condição_2 THEN resultado_2
  ...
  ELSE resultado_padrão
END
```

Exemplo:
Suponha que temos uma tabela chamada "Produtos" que contém informações sobre produtos em um estoque, incluindo o preço de cada produto. Gostaríamos de criar uma nova coluna chamada "Categoria" com base no preço dos produtos. Se o preço for menor que 50, a categoria será 'Barato', se estiver entre 50 e 100, a categoria será 'Médio', e se for maior que 100, a categoria será 'Caro'.

```
SELECT ProdutoID, Nome, Preco,
  CASE
    WHEN Preco < 50 THEN 'Barato'
    WHEN Preco BETWEEN 50 AND 100 THEN 'Médio'
    ELSE 'Caro'
  END AS Categoria
FROM Produtos;
```

Neste exemplo, estamos usando a cláusula CASE para avaliar a coluna "Preco" e gerar a coluna calculada "Categoria" com base nas condições definidas.

growdev

Cláusula WITH

Também conhecida como Common Table Expressions ou CTEs é usada para criar consultas temporárias nomeadas que podem ser referenciadas em uma consulta principal. Essa cláusula oferece uma maneira mais organizada e legível de escrever consultas complexas, permitindo que você divida a lógica da consulta em partes mais gerenciáveis.

Basicamente, você pode pensar em uma cláusula WITH como uma maneira de criar tabelas temporárias que existem apenas pelo tempo de execução da consulta. Isso pode tornar as consultas mais claras e facilitar a reutilização de partes da consulta em várias partes do código.

growdev

Exemplo

```
WITH produtos_caros AS (
  SELECT nome, preco
  FROM produtos
  WHERE preco > 100
)
SELECT nome, preco
FROM produtos_caros
ORDER BY preco DESC;
```

Nesse exemplo, a cláusula WITH cria uma tabela temporária chamada produtos_caros que contém os produtos com preço superior a 100. A consulta principal, que segue a cláusula WITH, então seleciona e ordena os produtos da tabela temporária produtos_caros.

O uso de WITH pode ser especialmente útil quando você deseja realizar várias operações em uma tabela ou conjunto de dados antes de executar a consulta principal. Isso melhora a legibilidade do código, ajuda na depuração e pode até mesmo melhorar o desempenho em algumas situações.

É importante notar que a cláusula WITH não cria tabelas reais no banco de dados. Ela apenas cria uma estrutura temporária que existe durante a execução da consulta que a utiliza.

Views

Uma VIEW (visão) no PostgreSQL é uma representação virtual de uma tabela ou consulta que permite aos usuários recuperar e manipular os dados de maneira semelhante a uma tabela real, mas sem armazenar fisicamente os dados.

As VIEWs são criadas usando a cláusula CREATE VIEW. Você especifica a consulta SQL que define a VIEW e atribui a ela um nome. A consulta pode incluir junções, filtros, agregações e outras operações SQL.

As VIEWs não armazenam dados fisicamente. Elas são consultas armazenadas que são executadas dinamicamente toda vez que você consulta a VIEW. Isso significa que os dados exibidos por uma VIEW são sempre atualizados em tempo real, refletindo quaisquer alterações nos dados subjacentes.

Uma vez criada, uma VIEW pode ser consultada usando uma sintaxe SQL padrão, assim como se estivesse consultando uma tabela real. Isso simplifica a recuperação de dados complexos e elimina a necessidade de escrever consultas SQL longas e repetitivas.

VIEWs podem ser usadas para criar "tabelas virtuais" personalizadas que representam uma visão específica dos dados, adaptada às necessidades dos usuários. Isso pode acelerar o desenvolvimento de aplicativos, fornecendo uma camada abstrata que isola os usuários da complexidade subjacente dos dados.

Exemplo

Suponha que temos uma tabela chamada "pedidos" com as seguintes colunas: id, cliente_id, data_pedido e valor_total. Vamos criar uma VIEW que exiba os detalhes dos pedidos feitos por um cliente específico.

```
CREATE VIEW detalhes_pedidos_cliente AS
SELECT p.id, p.data_pedido, p.valor_total
FROM pedidos p
WHERE p.cliente_id = 1;
```

Neste exemplo, criamos uma VIEW chamada "detalhes_pedidos_cliente" que seleciona os detalhes (id, data_pedido e valor_total) dos pedidos feitos por um cliente específico (no caso, cliente_id = 1).

Agora, você pode consultar essa VIEW da mesma forma que faria com uma tabela real:

```
SELECT * FROM detalhes_pedidos_cliente;
```

Isso retornará os detalhes dos pedidos feitos pelo cliente com o ID 1.

GROWDEV - Academy