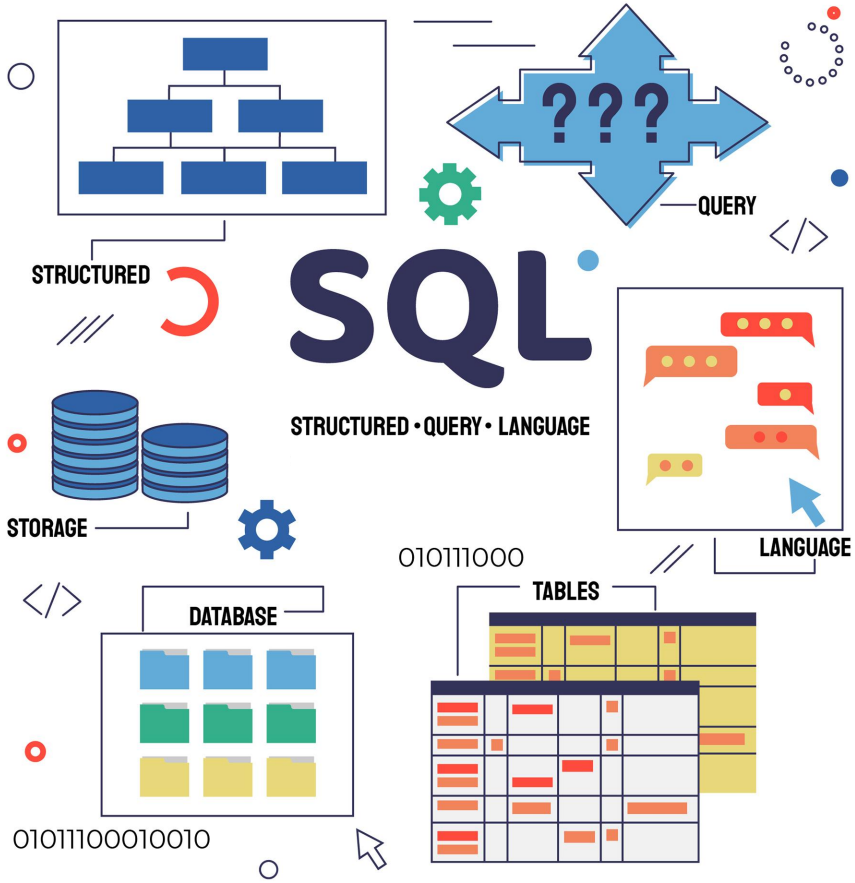
escola britânica de artes criativas & tecnologia

DE OLHO NO CÓDIGO



Índices

Índices em SQL são **estruturas de dados especiais** que melhoram a velocidade das operações de consulta em tabelas de um banco de dados.

Funcionam de maneira semelhante aos **índices de um livro**, permitindo acesso rápido a linhas específicas em uma tabela, sem precisar escanear cada linha.

Índices

escola britânica de artes criativas & tecnologia

Para que Servem os Índices?

- **Acelerar Consultas**: Reduzem o tempo necessário para recuperar registros de uma tabela, especialmente quando se trabalha com grandes volumes de dados.
- **Melhorar o Desempenho**: Ajudam a melhorar o desempenho de operações de seleção (SELECT), junção (JOIN), e outras consultas de busca.
- **Facilitar Ordenação e Agrupamento**: Otimizam operações que envolvem ordenação (ORDER BY) e agrupamento (GROUP BY).

Índices

escola britânica de artes criativas & tecnologia

Exemplos

Um **índice simples** é criado em uma única coluna da tabela. Ele é útil quando você consulta frequentemente a tabela com base em valores de uma única coluna.

Exemplo:

```
CREATE INDEX idx_customer_name ON customers(name);
```

Uso Prático:
Se você frequentemente realiza consultas para buscar clientes pelo nome, este índice irá acelerar a execução da consulta.

Consulta de Exemplo:

```
SELECT * FROM customers WHERE name = 'José Silva';
```

Índices

Exemplos

Um **índice composto** é criado em duas ou mais colunas de uma tabela. Ele é útil quando as consultas envolvem múltiplas colunas no filtro WHERE.

Exemplo:

```
CREATE INDEX idx_customer_name_city ON customers(name, city);
```

Uso Prático:
Se você realiza consultas frequentes usando o nome e a cidade como critérios de busca, este índice composto acelerará a consulta.

Consulta de Exemplo:

```
SELECT * FROM customers WHERE name = 'José Silva' AND city = São Paulo';
```

Índices

escola britânica de artes criativas & tecnologia

Exemplos

Um **índice único** não permite valores duplicados na coluna ou conjunto de colunas que ele indexa. É similar a uma chave primária, mas pode ser aplicado em colunas que não são chaves primárias.

Exemplo:

```
CREATE UNIQUE INDEX idx_email_unique ON customers(email);
```

Uso Prático:
Este índice assegura que o campo de e-mail de cada cliente seja único na tabela, evitando duplicatas.

Consulta de Exemplo:

```
SELECT * FROM customers WHERE email = 'john.doe@example.com';
```

Índices

Exemplos

Um **índice clusterizado** altera a forma como os dados são armazenados na tabela, ordenando fisicamente as linhas na base de dados. Geralmente, uma tabela pode ter apenas um índice clusterizado.

Exemplo:

```
CREATE CLUSTERED INDEX idx_order_date ON orders(order_date);
```

Uso Prático:
Se você frequentemente ordena ou filtra a tabela de pedidos pela data do pedido, um índice clusterizado na coluna order_date melhorará a performance.

Consulta de Exemplo:

```
SELECT * FROM orders ORDER BY order_date;
```

Índices

escola britânica de artes criativas & tecnologia

Exemplos

Um **índice não-clusterizado** é similar a um índice de livro. Ele não altera a ordem física dos dados na tabela, mas cria uma estrutura separada que aponta para os dados.

Exemplo:

```
CREATE NONCLUSTERED INDEX idx_product_price ON products(price);
```

Uso Prático:
Para consultas que frequentemente filtram ou ordenam por preço, este índice não-clusterizado pode melhorar o desempenho sem alterar a ordem física dos dados.

Consulta de Exemplo:

```
SELECT * FROM products WHERE price < 100 ORDER BY price;
```

Procedimentos armazenados (Stored Procedures)

Procedimentos armazenados (Stored Procedures) são **blocos de código SQL armazenados no banco de dados,** que podem ser **executados repetidamente para realizar operações específicas**.

Eles **encapsulam uma série de comandos SQL**, permitindo que você execute operações complexas de maneira mais eficiente e segura. Stored procedures são muito úteis em situações onde você precisa executar operações frequentes, complexas ou que envolvem várias etapas.

Procedimentos armazenados (Stored Procedures)

escola britânica de artes criativas & tecnologia

Vantagens dos Stored Procedures

Melhoria de Performance: Os procedimentos armazenados são compilados e otimizados pelo banco de dados na primeira vez que são executados, o que reduz o tempo de execução em execuções subsequentes.

Reutilização de Código: Eles permitem encapsular lógica de negócios, permitindo a reutilização de código em várias partes da aplicação, facilitando a manutenção e reduzindo a duplicação de código.

Procedimentos armazenados (Stored Procedures)

escola britânica de artes criativas & tecnologia

Vantagens dos Stored Procedures

Segurança: Eles permitem restringir o acesso direto às tabelas e permitem a implementação de controles de segurança, uma vez que o acesso aos dados pode ser gerido por meio dos procedimentos.

Redução do Tráfego de Rede: Como os procedimentos armazenados são executados no servidor de banco de dados, eles reduzem o tráfego de rede ao minimizar a quantidade de dados enviados entre a aplicação e o servidor.

Consistência de Transações: Eles ajudam a garantir que as transações sejam executadas de forma consistente, pois permitem agrupar várias operações em uma única unidade de execução.

Procedimentos armazenados (Stored Procedures)

escola britânica de artes criativas & tecnologia

Usos Comuns dos Stored Procedures

Operações CRUD: Automação de operações Create, Read, Update, Delete em tabelas de banco de dados.

Validação de Dados: Implementação de regras de negócios e validação de dados antes de efetuar operações de escrita.

Relatórios: Geração de relatórios complexos que envolvem várias tabelas e cálculos agregados.

Controle Transacional: Gerenciamento de transações que envolvem várias operações de leitura e escrita.

Integração de Aplicações: Facilitação de integração com outras aplicações que exigem processamento de dados padronizado.

Procedimentos armazenados (Stored Procedures)

escola britânica de artes criativas & tecnologia

Exemplo: Procedure Simples de Seleção

Este exemplo demonstra como criar um procedimento armazenado para buscar informações de clientes com base no ID.

```
CREATE PROCEDURE GetCustomerById
    @CustomerId INT
AS
BEGIN
    SELECT CustomerName, Email, Phone
    FROM Customers
    WHERE CustomerId = @CustomerId;
END;
```

Uso Prático:

```
EXEC GetCustomerById @CustomerId = 5;
```

Explicação: Este procedimento armazena uma consulta que retorna informações de um cliente específico, recebendo o ID do cliente como parâmetro de entrada.

Procedimentos armazenados (Stored Procedures)

Procedure para Gerar Relatório de Vendas

escola britânica de artes criativas & tecnologia

Este exemplo demonstra como criar um procedimento armazenado para gerar um relatório de vendas de um determinado período.

```
CREATE PROCEDURE SalesReport

    @StartDate DATE,

    @EndDate DATE

AS

BEGIN

    SELECT p.ProductName, SUM(o.Quantity) AS TotalSold, SUM(o.Quantity *
p.Price) AS TotalRevenue

    FROM Orders o

    JOIN Products p ON o.ProductId = p.ProductId

    WHERE o.OrderDate BETWEEN @StartDate AND @EndDate

    GROUP BY p.ProductName

    ORDER BY TotalRevenue DESC;

END;
```

Procedimentos armazenados (Stored Procedures)

Uso Prático:

```
EXEC SalesReport @StartDate = '2024-01-01', @EndDate = '2024-07-31';
```

Explicação: Este procedimento gera um relatório de vendas agrupado por produto, mostrando a quantidade total vendida e a receita total para um período específico.

Aplique subqueries Prática

- Evite subqueries aninhadas excessivas, pois elas podem tornar o código SQL difícil de entender e depurar.
- Prefira utilizar subqueries em vez de joins quando não for necessário obter todos os dados das tabelas envolvidas na consulta.
- Se estiver usando subqueries em uma cláusula WHERE, tente usar EXISTS em vez de IN, pois EXISTS geralmente é mais eficiente.
- Use subqueries nomeadas (*common table expressions*) para tornar o código mais legível e manutenível.

Faça agregações por particionamento Prática

- Use as funções de agregação por particionamento (como RANK, DENSE_RANK, ROW_NUMBER) para criar categorias em seus dados.
- Utilize as cláusulas PARTITION BY e ORDER BY para especificar como os dados devem ser particionados e ordenados, respectivamente.
- Tenha cuidado ao usar funções de agregação sem especificar a cláusula OVER, pois isso pode gerar resultados incorretos.
- Evite agrupar dados de uma tabela inteira ao usar funções de agregação por particionamento, pois isso pode afetar o desempenho.

Crie views

- Use views para simplificar consultas complexas ou para ocultar detalhes de implementação.
- Certifique-se de que as views sejam atualizáveis, se necessário, definindo as cláusulas WITH CHECK OPTION e/or INSTEAD OF.
- Dê nomes significativos às suas views, para que outras pessoas possam entender facilmente o que elas fazem.
- Se você tiver muitas views, organize-as em esquemas lógicos ou físicos para facilitar a localização.

Use joins

- As junções permitem que você combine dados de duas ou mais tabelas em uma única consulta SQL. Há diferentes tipos de junções que você pode utilizar, cada um com seu próprio uso e finalidade.

- **Entenda o tipo de join você precisa utilizar**:
Antes de começar a escrever sua consulta SQL, você precisa entender qual tipo de junção é necessário para obter os dados desejados. Por exemplo, se você quiser combinar dados de duas tabelas onde todos os valores correspondem, você precisará usar uma junção INNER. Se você deseja combinar dados de duas tabelas, mas incluir registros que não têm correspondência na outra tabela, você precisa usar uma junção LEFT ou RIGHT.

Use joins

- **Escreva a cláusula ON corretamente**:
  A cláusula ON é usada para especificar as colunas que serão usadas para a junção. É importante escrever a cláusula ON corretamente para garantir que os dados sejam combinados. Certifique-se de que as colunas na cláusula ON correspondem às colunas que você deseja combinar.

- **Use aliases para melhorar a legibilidade**:
  Quando você está trabalhando com várias tabelas, pode ser difícil ler e entender a consulta SQL. Use aliases para dar às tabelas e às colunas nomes mais fáceis de ler. Isso tornará a consulta mais fácil de entender e também pode economizar tempo.

Use joins

- **Teste sua consulta SQL**: Certifique-se de testar sua consulta SQL com diferentes conjuntos de dados para garantir que ela esteja retornando os resultados esperados. Verifique se todos os dados foram incluídos e se não há duplicatas.
- **Evite junções desnecessárias**: Usar muitas junções pode afetar o desempenho da consulta. Se possível, evite junções desnecessárias e opte por uma abordagem mais simples e direta para combinar seus dados.

Use joins

- **Considere a normalização de dados**:
  Se você está trabalhando com várias tabelas, é importante considerar a normalização de dados. A normalização pode ajudar a evitar duplicatas e reduzir o tamanho dos dados armazenados, o que pode melhorar o desempenho das consultas.

- **Use índices para melhorar o desempenho**:
  Se você estiver trabalhando com grandes conjuntos de dados, pode ser útil usar índices para melhorar o desempenho da consulta. Os índices podem ajudar a localizar rapidamente os registros necessários, em vez de fazer uma varredura completa da tabela.

Use joins

- **Documente sua consulta SQL**:
  Documentar sua consulta SQL pode ser útil para garantir que outros membros da equipe possam entender o que a consulta está fazendo. Inclua comentários explicando as diferentes partes da consulta e como ela funciona. Isso pode economizar tempo e evitar erros no futuro.

Use a função count e cláusula group by

A função COUNT é utilizada para contar a quantidade de linhas em uma determinada coluna. Já a cláusula GROUP BY é utilizada para agrupar as linhas da tabela por um determinado conjunto de colunas. Ao utilizá-las juntas, podemos obter a quantidade de linhas por grupo.

Use a função count e cláusula group by

- Utilize aliases para nomear as colunas geradas pela função COUNT, para que a saída seja mais fácil de ler.
- Tenha cuidado com a utilização de funções agregadas com colunas NULL, pois podem resultar em resultados inesperados.
- Verifique se as colunas selecionadas na cláusula GROUP BY são as mesmas que as colunas selecionadas na consulta, para evitar erros de sintaxe.

Use as funções min/max/sum /avg

- Utilize aliases para nomear as colunas geradas pelas funções, para que a saída seja mais fácil de ler.
- Verifique se as colunas selecionadas estão no formato correto para serem utilizadas nas funções, por exemplo, se uma coluna numérica não contém valores de texto.
- Lembre-se de que essas funções agregadas podem ser utilizadas em conjunto com outras cláusulas SQL, como WHERE e GROUP BY, para filtrar os dados adequadamente.
- As funções MIN, MAX, SUM e AVG são utilizadas para calcular valores agregados em uma coluna, como o valor mínimo, máximo, a soma e a média, respectivamente.

Use a cláusula having

- Utilize aliases para nomear as colunas geradas pelas funções, para que a saída seja mais fácil de ler.
- Verifique se as colunas utilizadas nas funções agregadas estão presentes na cláusula GROUP BY.
- Tenha cuidado ao utilizar a cláusula HAVING com grandes conjuntos de dados, pois pode afetar significativamente o desempenho da consulta.
- A cláusula HAVING é utilizada para filtrar grupos de linhas com base em condições agregadas. Ela é semelhante à cláusula WHERE, mas é utilizada com funções agregadas.

Uso do filtro where com and / or / between

- Use operadores lógicos `and` e `or` para filtrar resultados com múltiplas condições.
- Use o operador `in` para verificar se um valor corresponde a um conjunto de valores.
- Use o operador `between` para filtrar valores dentro de um intervalo.

Exemplos:

```
SELECT * FROM tabela
WHERE coluna1 > 10 AND coluna2 < 20;
```

```
SELECT * FROM tabela
WHERE coluna1 IN (1, 2, 3);
```

```
SELECT * FROM tabela
WHERE data BETWEEN '2022-01-01' AND
'2022-12-31';
```

Uso do filtro where com like / wildcards

- Use o operador `like` para buscar por padrões de caracteres em uma coluna.
- Use o caractere `%` para representar qualquer número de caracteres.
- Use o caractere `_` para representar um único caractere.

Exemplos:

```
SELECT * FROM tabela
WHERE nome LIKE 'João%';

SELECT * FROM tabela
WHERE telefone LIKE '55_%';

SELECT * FROM tabela
WHERE email LIKE '%@gmail.com';
```

Uso da seleção condicional com case / when / then

- Use a cláusula `case` para selecionar valores baseados em condições.
- Use a cláusula `when` para especificar condições.
- Use a cláusula `else` para definir um valor padrão caso nenhuma das condições seja atendida.

Exemplos:

```
SELECT nome,
CASE WHEN idade < 18 THEN 'Menor de idade'
     WHEN idade >= 18 AND idade <= 64 THEN
'Adulto'
     ELSE 'Idoso'
END AS faixa_etaria
         FROM tabela;
```

Restrinja os dados

- Defina restrições de coluna para garantir a integridade dos dados e evitar erros de inserção ou atualização de dados. As restrições podem ser definidas durante a criação da tabela ou posteriormente, usando a instrução ALTER TABLE.
- As restrições podem incluir restrições NOT NULL, UNIQUE, PRIMARY KEY e FOREIGN KEY.
- Use nomes descritivos para as restrições, para facilitar a manutenção do banco de dados.

Restrinja os dados

- Use a instrução SELECT para recuperar dados de uma ou mais tabelas do banco de dados.
- Especifique as colunas que deseja recuperar, em vez de selecionar todas as colunas (*).
- Use a cláusula WHERE para filtrar os resultados com base em critérios específicos.
- Evite o uso de subconsultas desnecessárias ou complexas, que podem afetar negativamente o desempenho do banco de dados.

Ordene e limite os resultados

- Use a cláusula ORDER BY para ordenar os resultados com base em uma ou mais colunas.
- Use a cláusula LIMIT para limitar o número de resultados retornados.
- Evite o uso excessivo de ORDER BY ou LIMIT, pois isso pode afetar negativamente o desempenho do banco de dados. Certifique-se de que essas cláusulas sejam usadas apenas quando necessário.

Crie tabelas

- **Definir chaves primárias**: Cada tabela deve ter uma chave primária que identifica de forma única cada registro. Ela deve ser definida como uma restrição na criação da tabela para garantir que não haja duplicatas.
- **Usar tipos de dados apropriados**: Escolha o tipo de dado apropriado para cada coluna da tabela, com base no tipo de dados que ela armazenará. Use tipos numéricos para valores numéricos, strings para valores de texto, etc.

Crie tabelas

- **Normalizar os dados**: Normalização é o processo de dividir os dados em várias tabelas menores para evitar redundância e inconsistências. Siga as regras de normalização para garantir que o esquema do banco de dados seja eficiente e fácil de manter.

- **Definir restrições e índices**: Restrições definem regras para os dados, como valores mínimos e máximos permitidos, ou restrições de chave estrangeira para garantir a integridade referencial. Índices ajudam a acelerar as consultas e reduzem o tempo de execução.

Altere e descarte tabelas

- **Faça backup dos dados**: Antes de fazer qualquer alteração ou exclusão, faça backup dos dados. Isso ajuda a garantir que, se algo der errado, você não perderá informações importantes.
- **Faça alterações em pequenas etapas**: Faça alterações incrementais em vez de fazer alterações em grande escala. Isso ajuda a garantir que você possa detectar e corrigir erros à medida que eles ocorrem.

Altere e descarte tabelas

- **Use transações**: Envolva as alterações em uma transação para garantir que elas sejam aplicadas corretamente ou revertidas se ocorrer algum erro.
- **Verifique as dependências**: Antes de descartar uma tabela, verifique se há outras tabelas que dependem dela. Se houver, você precisará atualizar ou descartar essas tabelas primeiro.

Insira, atualize e apague dados

- **Use transações**: Assim como ao alterar tabelas, use transações ao inserir, atualizar ou excluir dados. Isso ajuda a garantir que as alterações sejam aplicadas corretamente e evita erros.

- **Use declarações preparadas**: Declarações preparadas são instruções que são compiladas uma vez e reutilizadas várias vezes com diferentes parâmetros. Eles ajudam a evitar a injeção de SQL e melhorar o desempenho.

escola britânica de artes criativas & tecnologia

Bons estudos!