**Para saber mais: tipos de dados**

Os bancos de dados armazenam uma variedade de tipos de dados para atender às necessidades de diferentes tipos de informações e aplicativos. A escolha dos tipos de dados a serem usados em um banco de dados depende da natureza dos dados que serão armazenados e processados. Aqui estão alguns dos tipos de dados mais comuns em um banco de dados:

1 - **Texto (String):**

* **CHAR:** Armazena strings de tamanho fixo. Usado quando os valores têm um comprimento constante.
* **VARCHAR:** Armazena strings de tamanho variável. Apropriado para valores com comprimentos variáveis.
* **TEXTO (TEXT):** Armazena strings muito longas, como documentos ou descrições.

2 - **Numérico:**

* **INTEGER (INT):** Armazena números inteiros.
* **FLOAT:** Armazena números de ponto flutuante, geralmente usados para valores com casas decimais.
* **NUMERIC (DECIMAL):** Armazena números com uma precisão específica, geralmente usados em aplicações financeiras.

3 - **Data e Hora:**

* **DATE:** Armazena datas sem informações de horário.
* **TIME:** Armazena informações de horário.
* **TIMESTAMP:** Combina data e horário em um único tipo.

4 - **Booleano:**

* **BOOLEAN (BOOL):** Armazena valores verdadeiros ou falsos.

5 - **Binário:**

* **BLOB (Binary Large Object):** Armazena dados binários, como imagens, vídeos ou arquivos.
* **BIT:** Armazena valores binários, como 0 ou 1.

Esses são alguns dos tipos de dados comuns em bancos de dados relacionais. No entanto, a escolha dos tipos de dados pode variar dependendo do sistema de gerenciamento de banco de dados (SGBD) e das necessidades específicas de um aplicativo. Além disso, alguns SGBDs também oferecem tipos de dados personalizados que podem ser adaptados para requisitos específicos de negócios.

**Para saber mais: utilizando o comando CREATE**

SQL é uma linguagem de consulta estruturada que permite criar, modificar e gerenciar bancos de dados e seus objetos. Para criar tabelas, bancos de dados e esquemas, utilizamos o comando CREATE, mas vamos entender a diferença de cada uma dessas estruturas que esse comando nos possibilita criar?

**Banco de Dados:**

* Um banco de dados é uma coleção de dados organizados e relacionados.
* Ele atua como um contêiner para todos os objetos relacionados a dados, como tabelas, índices, procedimentos armazenados, visões e esquemas.
* Pode conter múltiplos esquemas, que são usados para organizar objetos de banco de dados.
* É uma entidade de nível superior que armazena e gerencia informações em um sistema de gerenciamento de banco de dados (SGBD).

**Esquema (Schema):**

* Um esquema é um contêiner lógico para objetos de banco de dados, como tabelas, visões, procedimentos armazenados, etc.
* É usado para organizar e segmentar objetos dentro de um banco de dados.
* Vários esquemas podem existir dentro de um único banco de dados, permitindo a separação de objetos e a aplicação de permissões específicas em cada esquema.
* É uma estrutura de nível inferior em relação ao banco de dados.

Aqui está como criar um banco de dados e um esquema em SQL, utilizando como exemplo um banco de dados para uma aplicação de gerenciamento de biblioteca:

**Criar um Banco de Dados:**

**CREATE** DATABASE BibliotecaDB;

Neste exemplo, "BibliotecaDB" é o nome do banco de dados que estamos criando. Este comando cria o banco de dados como uma entidade de nível superior que conterá todos os objetos relacionados à biblioteca.

**Criar um Esquema no Banco de Dados:**

**CREATE** SCHEMA LivrosSchema;

Aqui, "LivrosSchema" é o nome do esquema que estamos criando dentro do banco de dados. Este esquema pode ser usado para organizar todas as tabelas e objetos relacionados aos livros da biblioteca. Observe que, em muitos SGBDs, você pode criar um esquema implicitamente ao criar objetos, como tabelas. Por exemplo, ao criar uma tabela, você pode especificar o esquema no qual a tabela deve residir.

Lembre-se de que a sintaxe e a maneira de criar um banco de dados e um esquema podem variar entre os diferentes sistemas de gerenciamento de banco de dados (como MySQL, PostgreSQL, SQL Server, etc.), portanto, consulte a documentação específica do seu SGBD para obter detalhes exatos.

**Para saber mais: aprofundamento na sintaxe do SQL para os comandos DROP e ALTER**

Os comandos DROP e ALTER podem ajudar muito durante a manipulação dos dados, porém devemos entender muito bem como eles se comportam, pois são comandos que modificam a estrutura dos objetos e/ou do nosso banco de dados de forma que, muitas vezes, não conseguimos reverter.

**Entendendo o Comando DROP**

O comando DROP em SQL é usado para excluir um objeto do banco de dados. Podemos excluir índices, tabelas, bancos de dados, usuários, funções, entre outros.

Aqui está a sintaxe básica do comando DROP:

**DROP** tipo\_do\_objeto nome\_do\_objeto;

Por exemplo, para excluir uma tabela chamada "Estudantes", você usaria:

**DROP** **TABLE** Estudantes;

Agora se deseja excluir um banco de dados chamado “Colégio São Paulo”, seria assim:

**DROP** DATABASE Colégio\_São\_Paulo;

E caso queira excluir um esquema chamado “Turno da manhã”, contido em um banco de dados, fica assim:

**DROP** SCHEMA Turno\_da\_manhã;

Atenção: Cautela ao usar o comando DROP! Uma vez que um objeto é deletado, não pode ser recuperado.

**Entendendo o Comando ALTER**

O comando ALTER é usado para alterar a estrutura de um objeto existente em um banco de dados. Ele pode ser usado para adicionar, modificar ou excluir colunas em uma tabela existente. Também pode ser usado para adicionar e soltar várias restrições em uma tabela existente.

Aqui está a sintaxe básica do comando ALTER:

**ALTER** **TABLE** nome\_da\_tabela

**ADD** nome\_da\_coluna tipo\_de\_dado;

Por exemplo, para adicionar uma coluna chamada "Idade" à tabela "Estudantes", você usaria:

**ALTER** **TABLE** Estudantes

**ADD** Idade **INT**;

Para excluir uma coluna de uma tabela, usamos a sintaxe:

**ALTER** **TABLE** nome\_da\_tabela

**DROP** **COLUMN** nome\_da\_coluna;

Por exemplo:

**ALTER** **TABLE** Estudantes

**DROP** **COLUMN** Idade;

Lembre-se de sempre testar seus comandos SQL em um ambiente controlado antes de implementá-los em um banco de dados de produção. Além disso, é sempre uma boa prática fazer backup dos seus dados antes de fazer alterações estruturais.

Praticar o uso dos comandos DROP e ALTER em SQL vai te ajudar a se sentir mais confortável ao lidar com tabelas e outros objetos em um banco de dados. Lembre-se: a prática leva à perfeição!

**Para saber mais: Primary Key**

A "Primary Key" (Chave Primária) é um conceito fundamental no contexto de bancos de dados. Ela desempenha um papel crucial na organização e estruturação dos dados em um banco de dados relacional. Aqui vamos trazer uma explicação mais detalhada porque esse tema é muito importante e fará diferença na sua carreira na área de dados:

**O que é uma Primary Key?**

A Primary Key é um campo ou conjunto de campos em uma tabela de banco de dados que serve para identificar de forma exclusiva cada registro nessa tabela. Em outras palavras, é o meio pelo qual o sistema de gerenciamento de banco de dados (SGBD) garante que cada linha de dados seja única e possa ser acessada de forma eficiente.

**Características da Primary Key:**

1. **Unicidade:** Cada valor na coluna de chave primária deve ser único em relação a todos os outros valores na mesma coluna. Isso garante que nenhum registro duplicado seja inserido na tabela.
2. **Não nulo:** A chave primária não pode conter valores nulos. Cada registro deve ter um valor na coluna da chave primária.
3. **Eficiência de pesquisa:** A chave primária é usada para acelerar a pesquisa de registros no banco de dados. Os SGBDs criam índices automaticamente nas colunas de chave primária para otimizar a recuperação de dados.

**Por que a Primary Key é importante?**

A chave primária é fundamental para manter a integridade dos dados e garantir que as operações de consulta e atualização sejam eficientes. Veja:

1. **Identificação única:** Ela permite identificar cada registro de forma única, o que é essencial em muitos cenários de negócios. Por exemplo, em um banco de dados de clientes, a chave primária pode ser o número de identificação de cada cliente.
2. **Integridade referencial:** A chave primária é usada em relacionamentos entre tabelas para garantir a integridade referencial. Isso significa que os registros relacionados em tabelas diferentes podem ser conectados de maneira confiável.
3. **Eficiência:** Como mencionado anteriormente, as chaves primárias são indexadas automaticamente pelos SGBDs, tornando as consultas mais rápidas e eficientes.

**Exemplos de Primary Key:**

* Em uma tabela de "Clientes", o número de identificação de cada cliente pode ser a chave primária.
* Em uma tabela de "Pedidos", um número de pedido exclusivo pode servir como a chave primária.
* Em uma tabela de "Funcionários", o número de identificação de cada funcionário pode ser a chave primária.

Em resumo, a chave primária desempenha um papel central na estrutura de um banco de dados relacional, garantindo a integridade dos dados e possibilitando a recuperação eficiente das informações. É um conceito essencial para qualquer pessoa que trabalhe com bancos de dados e sistemas de informação.