Manual de Boas Práticas para Criação de Scripts SQL - PharmaFlow

O manual a seguir estabelece as **boas práticas e padrões SQL** a serem seguidos por todos os desenvolvedores e administradores de banco de dados dentro da **PharmaFlow**, com o objetivo de garantir que os scripts SQL sejam consistentes, seguros, eficientes e fáceis de manter. Este manual aborda práticas recomendadas para operações de criação e manutenção de tabelas, views, índices, transações, permissões e segurança, entre outras.

Objetivo

As boas práticas de desenvolvimento SQL são fundamentais para garantir a integridade dos dados, a segurança do banco de dados e a performance ideal de todas as operações executadas. Seguindo as diretrizes abaixo, garantimos que os scripts SQL da **PharmaFlow** atendam aos mais altos padrões de qualidade, segurança e eficiência.

1. Uso de WITH (NOLOCK)

O uso de WITH (NOLOCK) pode ser útil em algumas situações específicas, mas deve ser tratado com cuidado, pois ele permite leituras sujas (dirty reads), o que pode resultar em dados inconsistentes.

Regras para Uso de WITH (NOLOCK):

- **Evite WITH (NOLOCK) em transações críticas**: Para tabelas que envolvem informações sensíveis, como transações financeiras, dados de clientes e medicamentos, nunca use WITH (NOLOCK).
- **Utilize apenas para consultas de leitura**: WITH (NOLOCK) deve ser utilizado em consultas de leitura que não afetem a lógica de negócios, como relatórios históricos.
- **Justifique o uso em comentários**: Sempre que utilizar WITH (NOLOCK), adicione um comentário explicando o motivo da escolha.

Exemplo:

sql

Copiar código

-- Consulta para relatório de clientes inativos com NOLOCK para evitar bloqueios

```
SELECT nome, email

FROM clientes WITH (NOLOCK)

WHERE status = 'inativo';
```

2. Criação de Tabelas

A criação de tabelas é uma das operações mais importantes e deve seguir padrões consistentes. Isso garante que a estrutura do banco de dados esteja bem organizada, otimizada e fácil de manter.

Regras para Criação de Tabelas:

- **Nomenclatura Padrão**: Os nomes das tabelas devem ser descritivos e em plural. Por exemplo, use clientes e não cliente.
- **Chaves Primárias**: Toda tabela deve ter uma chave primária (PRIMARY KEY) definida, de preferência com um tipo de dado pequeno, como INT ou BIGINT.
- **Chaves Estrangeiras**: Sempre que houver relacionamento entre tabelas, use chaves estrangeiras (FOREIGN KEY) para garantir a integridade referencial.
- **Tipos de Dados**: Escolha tipos de dados que atendam aos requisitos de armazenamento e desempenho. Use tipos como VARCHAR, DECIMAL, INT ao invés de TEXT ou VARCHAR(MAX) sem necessidade.

Exemplo de Criação de Tabela:

sql

Copiar código

```
CREATE TABLE medicamentos (
   medicamento_id INT PRIMARY KEY,  -- Chave primária
   nome VARCHAR(255) NOT NULL,  -- Nome do medicamento
   categoria_id INT,  -- Chave estrangeira para categoria de medicamento
   preco DECIMAL(10, 2) NOT NULL,  -- Preço do medicamento
   estoque INT NOT NULL DEFAULT 0,  -- Quantidade em estoque
   CONSTRAINT fk_categoria FOREIGN KEY (categoria_id) REFERENCES
categorias(categoria_id)
);
```

3. Alterações em Tabelas (ALTER TABLE)

Alterações em tabelas são comuns durante a evolução de um sistema. Contudo, essas alterações precisam ser feitas com cuidado para não afetar dados críticos ou comprometer a performance.

Regras para Alterações em Tabelas:

- **Evite alterações frequentes**: Modificar tabelas em produção deve ser uma ação cuidadosamente planejada. Evite realizar alterações frequentes em tabelas críticas.
- **Adicione colunas e não remova**: Ao adicionar colunas, forneça um valor padrão para evitar valores nulos. Evite a remoção de colunas em ambientes de produção sem uma justificativa clara.
- **Renomeie tabelas ou colunas com precaução**: Se for necessário renomear tabelas ou colunas, faça isso de forma planejada, utilizando scripts de migração adequados.

Exemplo de Alteração de Tabela:

sql

Copiar código

-- Adicionando uma nova coluna de validade no estoque de medicamentos

ALTER TABLE medicamentos

ADD validade DATE;

4. Criação de Views

Views são usadas para simplificar consultas complexas, mas devem ser cuidadosas em termos de performance e manutenção.

Regras para Criação de Views:

- **Simplifique a consulta, não a torne mais complexa**: Views devem ser usadas para encapsular consultas complexas e promover reutilização. Evite views que fazem muitas junções ou agregações pesadas, pois isso pode prejudicar a performance.
- **Evite views que retornem grandes volumes de dados**: Sempre que possível, aplique filtros nas views para retornar apenas os dados necessários.

- **Adicione comentários explicativos**: As views devem ser bem documentadas, especialmente quando as consultas envolvem lógica complexa.

Exemplo de Criação de View:

sql

Copiar código

```
CREATE VIEW v_medicamentos_disponiveis AS

SELECT medicamento_id, nome, preco

FROM medicamentos

WHERE estoque > 0;  -- Apenas medicamentos com estoque disponível
```

5. Uso de Schemas

Schemas permitem a organização lógica dos objetos do banco de dados. É uma prática recomendada organizar objetos relacionados por módulos ou funcionalidades.

Regras para Uso de Schemas:

- **Organização por Módulos**: Use schemas diferentes para organizar objetos por módulo de negócios, como vendas, estoque, compras, etc.
- **Controle de Permissões**: Utilize schemas para aplicar permissões de acesso de maneira granular e evitar acessos indesejados.

Exemplo de Criação de Schema:

sql

Copiar código

```
-- Criando o schema para o módulo de compras

CREATE SCHEMA compras;


-- Criando uma tabela no schema de compras

CREATE TABLE compras.medicamentos (

  medicamento_id INT PRIMARY KEY,

  nome VARCHAR(255),

  preco DECIMAL(10, 2)
```

);

6. Transações e Controle de Concurrency

Transações são essenciais para garantir que operações no banco de dados sejam atômicas e consistentes. Devem ser utilizadas sempre que houver mais de uma operação de escrita.

Regras para Uso de Transações:

- **Use BEGIN TRANSACTION**: Toda operação que envolva múltiplas modificações no banco de dados deve estar encapsulada em uma transação.
- **Evite transações longas**: Transações que demoram muito para ser concluídas podem prejudicar a performance do banco de dados, especialmente em ambientes de alta concorrência.
- **Verifique o Status da Transação**: Certifique-se de que, em caso de erro, a transação seja revertida usando ROLLBACK.

Exemplo de Transação:

sql

Copiar código

```
BEGIN TRANSACTION;


-- Inserindo um novo pedido de medicamento

INSERT INTO pedidos (medicamento_id, quantidade, data_pedido)

VALUES (123, 10, GETDATE());


-- Atualizando o estoque de medicamentos

UPDATE medicamentos

SET estoque = estoque - 10

WHERE medicamento_id = 123;


-- Se não houver erro, faz o commit da transação

COMMIT;
```

7. Performance e Otimização

A performance de consultas SQL é uma das principais preocupações em bancos de dados. Devemos garantir que os scripts SQL sejam otimizados para minimizar o uso de recursos e melhorar o tempo de resposta.

Regras de Performance:

- **Criação de Índices**: Crie índices nas colunas frequentemente consultadas, como aquelas usadas em WHERE, JOIN ou ORDER BY.
- **Evite SELECT ***: Selecione apenas as colunas necessárias. O uso de SELECT * pode resultar em maior consumo de recursos.
- **Minimize Subconsultas**: Sempre que possível, prefira utilizar JOIN ao invés de subconsultas.

Exemplo de Otimização de Consulta:

sql

Copiar código

-- Usando JOIN ao invés de subconsulta

SELECT p.nome, c.nome AS cliente_nome

FROM pedidos p

JOIN clientes c ON p.cliente_id = c.cliente_id

WHERE p.data_pedido > '2023-01-01';

8. Segurança no Banco de Dados

A segurança do banco de dados é crucial, especialmente em sistemas que lidam com dados sensíveis, como informações de pacientes, médicos e medicamentos.

Regras de Segurança:

- **Evite credenciais hardcoded**: Nunca armazene credenciais no código. Use variáveis de ambiente ou arquivos de configuração seguros.
- **Privilégios Mínimos**: Conceda permissões mínimas aos usuários e grupos. Cada usuário deve ter acesso apenas aos dados que necessita.
- **Criptografia de Dados Sensíveis**: Sempre que possível, criptografe dados sensíveis, como senhas, informações pessoais e números de cartões de crédito.

Exemplo de Criação de Usuário com Permissões Mínimas:

sql

Copiar código

```
CREATE USER 'usuario_estoque'@'localhost' IDENTIFIED BY 'senhaSegura';

GRANT SELECT, INSERT, UPDATE ON estoque.* TO 'usuario_estoque'@'localhost';
```

Este manual contém as práticas que devem ser seguidas por todos os profissionais envolvidos na criação e manutenção de scripts SQL dentro da **PharmaFlow**. A adesão a estas práticas garante que o banco de dados esteja seguro, eficiente e fácil de gerenciar, além de promover a qualidade do código e a continuidade dos negócios.