Procedures no SQL Server – Guia Completo com Exemplos, Validações, Transações e RAISERROR

O que são Procedures?

Procedures são **blocos de comandos SQL armazenados no banco de dados**, que podem ser **executados quando necessário**. Elas organizam a lógica do sistema, padronizam operações e facilitam a manutenção.

✅ Vantagens

- Reutilização de lógica SQL
- Redução de código duplicado
- Maior segurança e controle
- Facilidade de manutenção
- Suporte a **parâmetros**, **validações**, **transações** e **tratamento de exceções**

Estrutura Básica

```
CREATE PROCEDURE NomeDaProcedure
AS
BEGIN
    -- Comandos SQL aqui
END
```

Procedure com Parâmetros

```
CREATE PROCEDURE BuscarCursoPorID
    @CursoID INT
AS
BEGIN
    SELECT * FROM Curso WHERE CursoID = @CursoID
END
```

Execução:

```
EXEC BuscarCursoPorID @CursoID = 1
```

Exemplo Prático – Sistema Escolar

```
CREATE TABLE Curso (
    CursoID INT PRIMARY KEY,
    Nome VARCHAR(100),
    ValorMensalidade DECIMAL(10,2)
);
CREATE TABLE Aluno (
    AlunoID INT PRIMARY KEY IDENTITY,
    Nome VARCHAR(100),
    CPF VARCHAR(11),
    CursoID INT FOREIGN KEY REFERENCES Curso(CursoID)
);

-- Inserir Cursos
INSERT INTO Curso (CursoID, Nome, ValorMensalidade) VALUES (1, 'Análise e
Desenvolvimento de Sistemas', 850.00);
INSERT INTO Curso (CursoID, Nome, ValorMensalidade) VALUES (2, 'Administração',
780.00);
INSERT INTO Curso (CursoID, Nome, ValorMensalidade) VALUES (3, 'Engenharia
Civil', 1200.00);
INSERT INTO Curso (CursoID, Nome, ValorMensalidade) VALUES (4, 'Direito',
1100.00);
INSERT INTO Curso (CursoID, Nome, ValorMensalidade) VALUES (5, 'Enfermagem',
900.00);

-- Inserir Alunos
INSERT INTO Aluno (Nome, CPF, CursoID) VALUES ('Mariana Lopes Silva',
'12345678901', 1);
INSERT INTO Aluno (Nome, CPF, CursoID) VALUES ('João Pedro Almeida',
'23456789012', 2);
INSERT INTO Aluno (Nome, CPF, CursoID) VALUES ('Beatriz Costa Rocha',
'34567890123', 1);
INSERT INTO Aluno (Nome, CPF, CursoID) VALUES ('Lucas Henrique Sousa',
'45678901234', 3);
INSERT INTO Aluno (Nome, CPF, CursoID) VALUES ('Ana Clara Fernandes',
'56789012345', 4);
INSERT INTO Aluno (Nome, CPF, CursoID) VALUES ('Carlos Eduardo Braga',
'67890123456', 5);
INSERT INTO Aluno (Nome, CPF, CursoID) VALUES ('Juliana Martins', '78901234567',
3);
```

1️⃣ Procedure: Inserir Aluno

```
CREATE PROCEDURE InserirAluno
    @Nome VARCHAR(100),
    @CPF VARCHAR(11),
    @CursoID INT
AS
BEGIN
    INSERT INTO Aluno (Nome, CPF, CursoID)
    VALUES (@Nome, @CPF, @CursoID)
END
```

Execução:

```
EXEC InserirAluno 'Mariana Lopes', '12345678901', 1
```

2️⃣ Procedure: Atualizar Mensalidade do Curso

```
CREATE PROCEDURE AtualizarMensalidadeCurso
    @CursoID INT,
    @NovoValor DECIMAL(10,2)
AS
BEGIN
    UPDATE Curso
    SET ValorMensalidade = @NovoValor
    WHERE CursoID = @CursoID
END
```

Execução:

```
EXEC AtualizarMensalidadeCurso @CursoID = 2, @NovoValor = 950.00
```

3️⃣ Procedure: Remover Aluno

```
CREATE PROCEDURE RemoverAluno
    @AlunoID INT
AS
BEGIN
    DELETE FROM Aluno WHERE AlunoID = @AlunoID
END
```

Execução:

```
EXEC RemoverAluno @AlunoID = 3
```

4️⃣ Procedure: Listar Alunos por Curso

```
CREATE PROCEDURE ListarAlunosPorCurso
    @CursoID INT
AS
BEGIN
    SELECT
        A.AlunoID,
        A.Nome AS NomeAluno,
        A.CPF,
        C.Nome AS NomeCurso,
        C.ValorMensalidade
    FROM Aluno A
    INNER JOIN Curso C ON A.CursoID = C.CursoID
    WHERE A.CursoID = @CursoID
END
```

Execução:

```
EXEC ListarAlunosPorCurso @CursoID = 1
```

Transações no SQL Server

O que são?

Transações garantem que **um grupo de comandos SQL sejam executados de forma atômica**: se tudo der certo, os dados são salvos (`COMMIT`); se algo falhar, nada é alterado (`ROLLBACK`).

Componentes de uma Transação

Comando	Função
`BEGIN TRANSACTION`	Inicia uma transação
`COMMIT`	Confirma as operações realizadas
`ROLLBACK`	Desfaz todas as operações desde o início da transação
`TRY...CATCH`	Captura e trata erros dentro de procedures

TRY...CATCH – Tratamento de Erros

Você pode usar o bloco `TRY...CATCH` para interceptar falhas de execução:

- `BEGIN TRY`
- `    -- Comandos que podem falhar`
- `END TRY`
- `BEGIN CATCH`
- `    -- Comandos de tratamento (ex: ROLLBACK, mensagens)`
- `END CATCH`

RAISERROR – Gerando Mensagens de Erro

O que é?

`RAISERROR` é usado para **lançar mensagens de erro personalizadas** no SQL Server e interromper a execução quando necessário.

Sintaxe:

```
RAISERROR('mensagem', severidade, estado)
```

- **mensagem**: Texto a ser exibido
- **severidade**: Nível do erro (11 a 16 para erros definidos pelo usuário)
- **estado**: Número de controle (use 1)

Exemplo de uso:

```
IF @ValorMensalidade <= 0
BEGIN
    RAISERROR('Valor inválido para mensalidade.', 16, 1)
END
```

Exemplos Práticos utilizando Procedures, Transações e Tratamento de Erros

1️⃣ Procedure: Matricular Aluno (com transação e validação)

```
CREATE PROCEDURE MatricularAluno
    @Nome VARCHAR(100),
    @CPF VARCHAR(11),
    @CursoID INT
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION

        -- Validação
        IF NOT EXISTS (SELECT 1 FROM Curso WHERE CursoID = @CursoID)
        BEGIN
            RAISERROR('Curso informado não existe.', 16, 1)
        END

        -- Inserção
        INSERT INTO Aluno (Nome, CPF, CursoID)
        VALUES (@Nome, @CPF, @CursoID)

        COMMIT
    END TRY
    BEGIN CATCH
        ROLLBACK
        PRINT 'Erro ao matricular aluno: ' + ERROR_MESSAGE()
    END CATCH
END
```

Execução:

```
EXEC MatricularAluno 'Carlos Silva', '12345678901', 2
```

2️⃣ Procedure: Inserir Curso com Validação

```
CREATE PROCEDURE InserirCursoComValidacao
    @Nome VARCHAR(100),
    @ValorMensalidade DECIMAL(10,2)
AS
BEGIN
    BEGIN TRY
        IF @ValorMensalidade <= 0
        BEGIN
            RAISERROR('Valor da mensalidade deve ser maior que zero.', 16, 1)
        END

        INSERT INTO Curso (Nome, ValorMensalidade)
        VALUES (@Nome, @ValorMensalidade)
    END TRY
    BEGIN CATCH
        PRINT 'Erro: ' + ERROR_MESSAGE()
    END CATCH
END
```

Execução:

```
EXEC InserirCursoComValidacao 'Engenharia', -850.00
```

3️⃣ Procedure: Matricular Aluno com Verificação de Vagas (Limite de 3 alunos por curso)

```
CREATE PROCEDURE MatricularAlunoComLimite
    @Nome VARCHAR(100),
    @CPF VARCHAR(11),
    @CursoID INT
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION

        -- Verifica existência do curso
        IF NOT EXISTS (SELECT 1 FROM Curso WHERE CursoID = @CursoID)
        BEGIN
            RAISERROR('Curso informado não existe.', 16, 1)
            ROLLBACK
            RETURN
        END

        -- Verifica limite de vagas
        DECLARE @TotalAlunos INT
        SELECT @TotalAlunos = COUNT(*) FROM Aluno WHERE CursoID = @CursoID

        IF @TotalAlunos >= 3
        BEGIN
            RAISERROR('Curso atingiu o limite de vagas.', 16, 1)
            ROLLBACK
            RETURN
        END

        -- Realiza matrícula
        INSERT INTO Aluno (Nome, CPF, CursoID)
        VALUES (@Nome, @CPF, @CursoID)

        COMMIT
    END TRY
    BEGIN CATCH
        ROLLBACK
        PRINT 'Erro ao matricular aluno: ' + ERROR_MESSAGE()
    END CATCH
END
```

Execução:

```
EXEC MatricularAlunoComLimite 'Renata Oliveira', '98765432100', 1
```

Comandos Úteis

Ação	Comando SQL
Executar procedure	`EXEC NomeDaProcedure [@parametros]`
Alterar procedure	`ALTER PROCEDURE NomeDaProcedure AS ...`
Excluir procedure	`DROP PROCEDURE NomeDaProcedure`
Listar procedures	`SELECT * FROM sys.procedures`