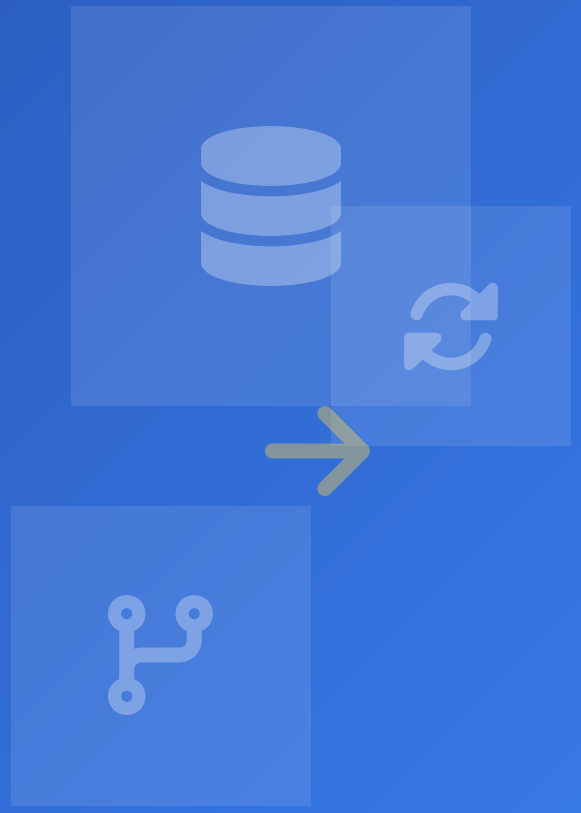
Gerenciamento de Esquema de Banco de Dados

DDL Avançado e Migrações

UC3 - Banco de Dados

Aula 06: ALTER TABLE, Migrações e Rollback



Contexto: A Evolução Contínua dos Bancos de Dados

> **Um banco de dados nunca está pronto; ele evolui com o negócio.**

Negócios Evoluem

Novos requisitos surgem constantemente: novos campos, novas regras de negócio, novas integrações. O banco de dados precisa acompanhar essas mudanças.

Trabalho Real

Na vida real, a maior parte do trabalho de um administrador ou desenvolvedor de banco de dados não é criar tabelas, mas sim **alterá-las** (migrar o esquema) sem perder dados.

ALTER TABLE

O comando **ALTER TABLE** é a ferramenta central para essa tarefa. Ele permite modificar a estrutura de tabelas existentes de forma segura e controlada.

Fundamentos do ALTER TABLE

O comando **ALTER TABLE** permite modificar a estrutura de tabelas existentes. Conheça os quatro comandos básicos de alteração DDL:

ADD COLUMN

Adiciona uma nova coluna à tabela existente. A nova coluna será criada com valores NULL para registros existentes.

```
ALTER TABLE Funcionarios
ADD COLUMN email_profissional VARCHAR(100);
```

DROP COLUMN

Remove uma coluna da tabela. **ATENÇÃO:** Os dados da coluna serão permanentemente perdidos. Faça backup antes!

```
ALTER TABLE Funcionarios
DROP COLUMN fax;
```

ALTER/MODIFY COLUMN

Altera o tipo de dado ou restrições de uma coluna existente. A sintaxe varia conforme o SGBD (PostgreSQL, MySQL, SQL Server).

```
-- PostgreSQL:
ALTER TABLE Funcionarios
ALTER COLUMN status SET NOT NULL;

-- MySQL/SQL Server:
ALTER TABLE Funcionarios
MODIFY COLUMN status VARCHAR(20) NOT NULL;
```

ADD CONSTRAINT

Adiciona uma restrição à tabela (PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK) para garantir a integridade dos dados.

```
ALTER TABLE Funcionarios
ADD CONSTRAINT fk_departamento
FOREIGN KEY (id_departamento)
REFERENCES Departamento(id);
```

O Risco da Migração DDL

⚠ O Risco da Migração DDL

Alterações no DDL (Data Definition Language) são **operações perigosas** que podem causar impactos graves no banco de dados de produção. Um erro pode levar à perda de dados, indisponibilidade do sistema ou corrupção da estrutura.

Principais riscos:

Perda de dados: Remover uma coluna (DROP COLUMN) apaga permanentemente todos os dados armazenados nela

Incompatibilidade de tipos: Alterar o tipo de dado pode causar perda de precisão ou falha na conversão

Violação de restrições: Adicionar NOT NULL em coluna com valores NULL causa erro e bloqueia a migração

Impacto em aplicações: Mudanças no esquema podem quebrar código que depende da estrutura antiga

A Importância do Rollback

O que é Rollback?

Rollback é o conjunto de comandos SQL que **desfazem as alterações** feitas por uma migração, restaurando o banco de dados ao estado anterior. Para cada comando de migração, deve haver um **comando reverso** correspondente para garantir a segurança e permitir a reversão em caso de problemas.

Por que o Rollback é Essencial?

Segurança: Se algo der errado durante a migração (erro de sintaxe, perda de dados, problema de desempenho), o rollback permite voltar ao estado estável anterior rapidamente.

Testes: Em ambientes de homologação, o rollback permite testar migrações e depois reverter para testar novamente com ajustes, sem precisar recriar todo o banco.

Profissionalismo: Em ambientes profissionais, ter um script de rollback documentado e testado é uma **exigência** antes de executar qualquer migração em produção.

Regra de Ouro: Execução na Ordem Reversa

→ MIGRAÇÃO (Ordem Normal)

Passo 1: ADD COLUMN email_profissional

Passo 2: DROP COLUMN fax

Passo 3: ALTER COLUMN status SET NOT NULL

← ROLLBACK (Ordem Reversa)

Passo 3: ALTER COLUMN status DROP NOT NULL

Passo 2: ADD COLUMN fax

Passo 1: DROP COLUMN email_profissional

Cenário de Evolução: Tabela Funcionarios

Estrutura Atual da Tabela Funcionarios

Coluna	Tipo de Dado	Restrições	Observações
id	INT	PK	Identificador único
nome	VARCHAR(100)	NOT NULL	Nome completo
email	VARCHAR(100)	UNIQUE	Email pessoal
fax	VARCHAR(15)	NULL	Número de fax (obsoleto)
status	VARCHAR(20)	NULL	Ativo/Inativo

Mudanças Necessárias para Evolução do Esquema

Mudança 1: Adicionar campo email_profissional

Justificativa: O campo email não é mais suficiente. A empresa precisa separar o email pessoal do email profissional dos funcionários para comunicações corporativas.

Mudança 2: Remover campo fax (obsoleto)

Justificativa: O campo fax é obsoleto e não é mais utilizado pela empresa. Nenhum funcionário possui fax atualmente, tornando a coluna desnecessária e ocupando espaço no banco de dados.

Mudança 3: Tornar o campo status obrigatório (NOT NULL)

Justificativa: O campo status (Ativo/Inativo) deve ser obrigatório para todos os funcionários. Atualmente permite valores NULL, o que causa inconsistências no controle de funcionários ativos.

Atividade Prática 1 - Scripts de Migração (Parte 1)

1 Adicionar Campo email_profissional

Objetivo e Justificativa

Objetivo: Adicionar uma nova coluna **email_profissional** à tabela Funcionarios para separar o email pessoal do profissional.

Justificativa: O campo email não é mais suficiente; precisamos distinguir entre email pessoal e email profissional para comunicações corporativas.

Instruções:

- Use o comando `ALTER TABLE ... ADD COLUMN`
- Defina o tipo de dado como `VARCHAR(100)`
- A nova coluna será criada com valores NULL para registros existentes
- Execute o script e verifique se a coluna foi adicionada

Script DDL: Adicionar Coluna

```sql
-- Adicionar nova coluna email_profissional
ALTER TABLE Funcionarios
ADD COLUMN email_profissional VARCHAR(100);

-- Verificação: Visualizar estrutura da tabela
SELECT * FROM Funcionarios LIMIT 5;
```

Verificação Esperada

A nova coluna **email_profissional** deve aparecer na estrutura da tabela com valores NULL para todos os registros existentes.

2 Remover Campo Obsoleto fax

Atividade Prática 1 - Passo 3: Adicionar NOT NULL

3 Adicionar Restrição NOT NULL no Campo status

Objetivo:

Tornar o campo **status** (Ativo/Inativo) obrigatório para todos os funcionários, garantindo que não haja valores NULL.

Passo a Passo:

Verificar valores NULL: Antes de adicionar NOT NULL, execute uma consulta para verificar se há valores NULL na coluna status

Atualizar valores NULL: Se houver valores NULL, atualize-os para um valor padrão (ex: 'Ativo')

Adicionar restrição: Execute o comando ALTER TABLE com a sintaxe apropriada para seu SGBD

⚠ **IMPORTANTE:** Se houver valores NULL na coluna, o comando falhará. Certifique-se de atualizar todos os valores NULL antes de adicionar a restrição NOT NULL.

Script DDL: Adicionar NOT NULL

```sql
-- 1. Verificar se há valores NULL
SELECT COUNT(*)
FROM Funcionarios
WHERE status IS NULL;

-- 2. Atualizar valores NULL (se houver)
UPDATE Funcionarios
SET status = 'Ativo'
WHERE status IS NULL;

-- 3. Adicionar restrição NOT NULL
-- PostgreSQL:
ALTER TABLE Funcionarios
ALTER COLUMN status SET NOT NULL;

-- MySQL/SQL Server:
ALTER TABLE Funcionarios
MODIFY COLUMN status VARCHAR(20) NOT NULL;
```

Atividade Prática 1 - Passo 4: Execução e Teste

4 Execução e Teste dos Scripts de Migração

Objetivo:

Executar os três scripts de migração em ordem e testar se as alterações foram aplicadas corretamente.

Passo a Passo:

Executar em ordem: Execute os scripts na ordem correta (Passo 1 → Passo 2 → Passo 3)

Verificar estrutura: Use DESCRIBE ou \d para verificar se as colunas foram adicionadas/removidas

Testar nova coluna: Verifique se a coluna email_profissional está visível

Testar restrição: Tente inserir um registro com status NULL para verificar se a restrição está funcionando

Scripts de Verificação e Teste

```
-- Verificar estrutura da tabela
DESCRIBE Funcionarios; -- MySQL
-- ou
\d Funcionarios; -- PostgreSQL

-- Testar nova coluna email_profissional
SELECT id, nome, email_profissional
FROM Funcionarios LIMIT 5;

-- Testar restrição NOT NULL (deve falhar)
INSERT INTO Funcionarios
(nome, email, status)
VALUES ('Teste', 'teste@email.com', NULL);
-- Erro esperado: NULL não permitido
```

A Importância do Rollback

O que é Rollback?

Rollback é o conjunto de comandos SQL que **desfazem as alterações** feitas por uma migração, restaurando o banco de dados ao estado anterior. Para cada comando de migração, deve haver um **comando reverso** correspondente para garantir a segurança e permitir a reversão em caso de problemas.

Por que o Rollback é Essencial?

Segurança: Se algo der errado durante a migração (erro de sintaxe, perda de dados, problema de desempenho), o rollback permite voltar ao estado estável anterior rapidamente.

Testes: Em ambientes de homologação, o rollback permite testar migrações e depois reverter para testar novamente com ajustes, sem precisar recriar todo o banco.

Profissionalismo: Em ambientes profissionais, ter um script de rollback documentado e testado é uma **exigência** antes de executar qualquer migração em produção.

Regra de Ouro: Execução na Ordem Reversa

→ MIGRAÇÃO (Ordem Normal)

Passo 1: ADD COLUMN email_profissional

Passo 2: DROP COLUMN fax

Passo 3: ALTER COLUMN status SET NOT NULL

← ROLLBACK (Ordem Reversa)

Passo 3: ALTER COLUMN status DROP NOT NULL

Passo 2: ADD COLUMN fax

Passo 1: DROP COLUMN email_profissional

Atividade Prática 2 - Rollback 1: Reverter NOT NULL

Lembre-se: Execute os comandos de rollback na **ORDEM REVERSA** da migração para garantir a integridade dos dados!

3 Rollback 1: Reverter Restrição NOT NULL do Campo status

Objetivo e Justificativa

Objetivo: Remover a restrição **NOT NULL** do campo status, permitindo novamente valores NULL.

Justificativa: O rollback de um `SET NOT NULL` é um `DROP NOT NULL`. Isso desfaz a obrigatoriedade do campo status.

Instruções:

- Use o comando `ALTER TABLE ... ALTER COLUMN ... DROP NOT NULL` (PostgreSQL)
- Ou use `ALTER TABLE ... MODIFY COLUMN ... NULL` (MySQL/SQL Server)
- Execute o script e verifique se a restrição foi removida

Script Rollback: Remover NOT NULL

```sql
-- Remover restrição NOT NULL do campo status
-- PostgreSQL:
ALTER TABLE Funcionarios
ALTER COLUMN status DROP NOT NULL;

-- MySQL/SQL Server:
ALTER TABLE Funcionarios
MODIFY COLUMN status VARCHAR(20) NULL;

-- Verificação: Confirmar remoção
DESCRIBE Funcionarios; -- MySQL
```

Observação Importante

Após o rollback, o campo **status** voltará a permitir valores NULL. Verifique se a coluna não tem mais a restrição NOT NULL na estrutura da tabela.

Atividade Prática 2 - Rollback 2: Adicionar Coluna fax

2 Rollback 2: Adicionar Novamente a Coluna fax

Objetivo e Justificativa

Objetivo: Adicionar novamente a coluna **fax** que foi removida durante a migração.

Justificativa: O rollback de um `DROP COLUMN` é um `ADD COLUMN`.

IMPORTANTE: O tipo de dado deve ser EXATAMENTE o mesmo que estava antes.

Instruções:

- Use o comando `ALTER TABLE ... ADD COLUMN`
- Defina o tipo de dado como `VARCHAR(15)` (tipo original)
- Execute o script e verifique se a coluna foi adicionada

Script Rollback: Adicionar Coluna fax

```sql
-- Adicionar novamente a coluna fax
ALTER TABLE Funcionarios
ADD COLUMN fax VARCHAR(15);

-- Verificação: Confirmar adição
SELECT * FROM Funcionarios LIMIT 5;
```

⚠ Observação Crítica

ATENÇÃO: Os dados antigos do campo fax foram **permanentemente perdidos** quando a coluna foi removida. Este rollback apenas recria a **estrutura** da coluna, mas os valores antigos NÃO serão recuperados automaticamente. Se houver backup dos dados, restaure manualmente.

Atividade Prática 2 - Scripts de Rollback (Parte 2)

1 Rollback 3: Remover Campo email_profissional

Objetivo e Justificativa

Objetivo: Remover a coluna **email_profissional** que foi adicionada durante a migração.

Justificativa: O rollback de um `ADD COLUMN` é um `DROP COLUMN`. Isso desfaz a adição da nova coluna email_profissional.

Instruções:

- Use o comando `ALTER TABLE ... DROP COLUMN`
- Execute o script e verifique se a coluna foi removida
- Confirme que a estrutura da tabela voltou ao estado original

Script Rollback: Remover Coluna

```sql
-- Remover coluna email_profissional
ALTER TABLE Funcionarios
DROP COLUMN email_profissional;

-- Verificação: Confirmar remoção
DESCRIBE Funcionarios; -- MySQL
-- ou
\d Funcionarios; -- PostgreSQL
```

Validação e Verificação Final do Esquema Restaurado

Documentação e Consolidação Final

Documentação dos Scripts de Migração e Rollback

Organize seus scripts em arquivos separados com **nomes descritivos** e **versionamento**. Cada arquivo deve conter comentários explicativos sobre o propósito da migração e instruções de execução.

MIGRACAO_V2.sql

```
-- Migração V2: Evolução Funcionarios
-- Data: 2025-10-19
-- Autor: [Seu Nome]
-- Descrição: Adiciona email_profissional,
--            remove fax, torna status NOT NULL

-- Passo 1: ADD COLUMN email_profissional
ALTER TABLE Funcionarios ...

-- Passo 2: DROP COLUMN fax
ALTER TABLE Funcionarios ...

-- Passo 3: ALTER COLUMN status NOT NULL
ALTER TABLE Funcionarios ...
```

ROLLBACK_V2.sql

```
-- Rollback V2: Reverter Migração V2
-- Data: 2025-10-19
-- Autor: [Seu Nome]
-- Descrição: Reverte alterações da V2
--            na ORDEM REVERSA

-- Passo 3: DROP NOT NULL do status
ALTER TABLE Funcionarios ...

-- Passo 2: ADD COLUMN fax
ALTER TABLE Funcionarios ...

-- Passo 1: DROP COLUMN email_profissional
ALTER TABLE Funcionarios ...
```

Processo Profissional de Gerenciamento de Migrações

1. **Planejar**
   Identificar mudanças necessárias
2. **Escrever**
   Criar scripts de migração e rollback
3. **Testar**
   Executar em ambiente de teste
4. **Revisar**
   Validar com equipe técnica
5. **Executar**
   Aplicar em produção com backup

Checklist de Boas Práticas

- Sempre crie scripts de **migração** e **rollback** em pares
- Documente cada script com **comentários explicativos**
- Teste scripts em **ambiente de desenvolvimento** antes de produção
- Verifique valores NULL antes de adicionar **NOT NULL**
- Execute rollback na **ordem reversa** da migração
- Use **versionamento** (V1, V2, V3...) para organizar migrações
- Faça **backup** antes de executar qualquer migração em produção
- Lembre-se: **DROP COLUMN** causa perda permanente de dados