C# com MySQL – Apostila Didática

Siga um passo a passo, você encontrará explicações simples de cada etapa, boas práticas e uma atividade prática para fixar o conteúdo.

> **Atividade prática:** Para exercitar sua atenção, esta apostila contém **cinco erros intencionais** espalhados ao longo do texto e dos exemplos de código. Leia com calma, identifique-os e corrija-os em seu projeto.

1. Introdução

1.1 O que é um banco de dados?

Um **banco de dados** é como um **arquivo organizado** onde guardamos informações. Pense nele como uma planilha: cada **tabela** possui **colunas** (campos) e **linhas** (registros). O **MySQL** é um dos sistemas mais populares para gerenciar bancos de dados relacionais. Ele é gratuito, confiável e utilizado por empresas de todo o mundo.

1.2 Por que usar C# com MySQL?

O C# é uma linguagem forte no ecossistema .NET, muito usada no mercado de trabalho. Com a biblioteca **MySQL Connector/NET**, é possível conectar aplicações C# diretamente ao MySQL. Essa biblioteca expõe classes como `MySqlConnection` e `MySqlCommand`, que permitem enviar comandos SQL e recuperar resultados.

1.3 Objetivo desta apostila

Ao final deste guia, você será capaz de:

- Preparar o ambiente de desenvolvimento, instalando o **MySQL Server**, o **Visual Studio** e o **MySQL Connector/NET**.
- Criar um projeto de console em C# no Visual Studio.
- Montar a **string de conexão** e abrir/fechar a conexão com o banco.
- Executar operações **CRUD** usando classes do namespace `MySql.Data.MySqlClient`.
- Aplicar boas práticas de segurança e desempenho.
- Desenvolver uma aplicação simples para gerenciar registros em um banco de dados.

> **Dica inicial:** Não tente memorizar tudo de uma vez. Leia, experimente no seu projeto e volte à apostila sempre que precisar. A prática é a chave para aprender programação.

2. Preparação do ambiente

Antes de começar a programar, vamos preparar as ferramentas necessárias. Este módulo é dividido em três partes: instalação do MySQL, instalação do Visual Studio e obtenção do **MySQL Connector/NET**.

2.1 Instalando o MySQL Server

1. Acesse o site oficial do MySQL (https://dev.mysql.com/downloads/installer/) e baixe o instalador compatível com seu sistema operacional (Windows 64 bits, por exemplo).
2. Execute o instalador e siga as instruções de instalação. Escolha a opção **Developer Default** ou **Server only**, conforme sua preferência.
3. Durante a instalação, configure uma senha para o usuário **root**. Guarde esta senha, pois ela será usada em sua string de conexão.
4. Após a instalação, inicie o serviço do MySQL. Você pode usar o MySQL Workbench ou a linha de comando (`mysql -u root -p`) para testar se está funcionando.

2.2 Instalando o Visual Studio

1. Baixe a versão **Community** ou superior do Visual Studio em https://visualstudio.microsoft.com/pt-br/downloads/.
2. Durante a instalação, marque a carga de trabalho **Desenvolvimento em .NET** (ou **Desenvolvimento em .NET Desktop**) para garantir que o .NET e o C# serão instalados.
3. Após a instalação, execute o Visual Studio e faça login com sua conta Microsoft (caso solicitado).

2.3 Obtendo o MySQL Connector/NET

Para que o C# "converse" com o MySQL, precisamos do **conector**. Existem duas formas de obtê-lo:

A) Instalando via NuGet (método recomendado)

A documentação do MySQL informa que o **repositório NuGet** contém as versões mais recentes do **Connector/NET**【867570227690941†L231-L244】. O NuGet é um gerenciador de pacotes integrado ao Visual Studio. Para instalar:

1. No **Gerenciador de Soluções** do Visual Studio, clique com o botão direito em **Dependências** do seu projeto e escolha **Gerenciar Pacotes NuGet...**.
2. Na guia **Procurar**, digite `MySql.Data` e pressione **Enter**. Selecione o pacote **MySql.Data** e clique em **Instalar**.
3. Aceite as licenças. Ao final, o Visual Studio adicionará automaticamente a referência `MySql.Data.dll` ao seu projeto.

B) Instalando manualmente via instalador (caso o NuGet não seja possível)

Se você não tiver acesso à internet ou não puder usar o NuGet, baixe o conector manualmente. O manual da Oracle descreve que existe um instalador `.msi` chamado **`mysql-connector-net-versão.msi`**【830068984827776†L235-L239】. Siga estes passos:

1. Visite https://dev.mysql.com/downloads/connector/net/ e faça download da versão desejada.

2. Execute o instalador e clique em **Next** nas telas iniciais. Escolha o tipo de instalação (Typical, Custom ou Complete) conforme sua necessidade.
3. Finalize a instalação. O instalador adicionará o conector ao **Global Assembly Cache (GAC)** e instalará os arquivos em `C:\Program Files (x86)\MySQL\MySQL Connector Net <versão>.`
4. No Visual Studio, clique com o botão direito em **Dependências**, selecione **Adicionar referência...** e, na guia **Procurar**, localize a pasta de instalação. Selecione `MySql.Data.dll` e clique em **OK**.

> **Observação:** Use sempre a versão do conector compatível com a versão do seu .NET (por exemplo, .NET 6). Versões incompatíveis podem gerar erros ao conectar.

3. Criando seu primeiro projeto no Visual Studio

Com o ambiente pronto, vamos criar um projeto de **Aplicativo de Console (.NET)**, que exibe texto no terminal e aceita entrada do usuário.

1. Abra o Visual Studio e clique em **Criar um novo projeto**.
2. Selecione a opção **Aplicativo de Console** (Certifique-se de escolher a plataforma .NET adequada) e clique em **Avançar**.
3. Defina um nome para o projeto, por exemplo **`GestaoAlunos`**, e selecione um local para salvá-lo. Clique em **Criar**.
4. Assim que o projeto for criado, instale o **MySql.Data** conforme explicado na seção 2.3.
5. Abra o arquivo `Program.cs`. No topo do arquivo, adicione o seguinte `using` para importar as classes do conector:

```
using MySql.Data.MySqlClient;
using System;
```

> **Entendendo os `using`:** A primeira linha informa ao compilador onde encontrar as classes `MySqlConnection` e `MySqlCommand`. A segunda linha permite usar classes básicas do C# como `Console` e `Convert`.

4. Conectando ao banco de dados

4.1 Montando a string de conexão

Para iniciar uma conversa com o MySQL, precisamos de uma **string de conexão**. Pense nela como um **caminho** contendo as informações necessárias para localizar e autenticar no banco. A documentação oficial mostra um exemplo de string de conexão:

```
"server=127.0.0.1;uid=root;pwd=12345;database=test"
```

Cada segmento é um par `chave=valor` separado por ponto e vírgula. Veja o que cada parte significa:

- **server**: endereço ou IP onde o MySQL está rodando (`localhost` ou `127.0.0.1` se for no mesmo computador). Você pode informar a porta caso não seja a padrão (`port=3306`).
- **uid** (User ID): nome do usuário (ex.: `root`).
- **pwd** (Password): senha do usuário.
- **database**: nome do banco que você quer acessar (por exemplo, `escola`).

Monte sua string adaptando para os seus dados de acesso. Exemplo:

```
string conexaoString =
"server=localhost;uid=root;pwd=SuaSenhaAqui;database=escola;port=5432";
```

> **Dica de segurança:** Nunca divulgue senhas em código público. Em aplicações reais, utilize arquivos de configuração ou variáveis de ambiente para guardar dados sensíveis.

4.2 Abrindo e fechando a conexão

Para se comunicar com o MySQL, utilizamos a classe **`MySqlConnection`**. O manual mostra um exemplo de como instanciar a conexão, definir a string e abrir/fechar. Siga estes passos:

1. Crie um objeto `MySqlConnection` passando a sua string de conexão.
2. Chame o método `Open()` para iniciar a conexão.
3. Execute suas operações (inserir, ler, etc.).
4. Feche a conexão ao final (acontece automaticamente se usar `using`).

Sempre coloque o código dentro de um bloco `try/catch` para capturar falhas, como senha incorreta ou servidor indisponível. Exemplo:

```
try
{
    using MySqlConnection conexao = new MySqlConnection(conexaoString);
    conexao.Open();
    Console.WriteLine("Conexão aberta com sucesso!");

}
catch (MySqlException ex)
{
    Console.WriteLine($"Erro ao conectar: {ex.Message}");
}
```

> **Por que usar `using`?** O bloco `using` garante que, ao sair do bloco, o objeto será descartado e a conexão será fechada, mesmo que ocorra uma exceção. Isso evita vazamentos de conexão.

5. Executando operações CRUD

Para manipular dados no MySQL, usaremos a classe **`MySqlCommand`**, cujos métodos principais são:

- **`ExecuteReader`** – executa consultas (`SELECT`) e retorna um **MySqlDataReader** que lê múltiplos registros.
- **`ExecuteNonQuery`** – executa comandos que não retornam dados (como `INSERT`, `UPDATE` e `DELETE`).
- **`ExecuteScalar`** – retorna um único valor (por exemplo, o resultado de `SELECT COUNT(*)`).

Em todas as operações a seguir, **abra a conexão antes** e **feche ao final** (usando `using`).

5.1 Lendo dados (SELECT)

Vamos consultar a tabela `alunos` e mostrar cada registro no console. Siga o roteiro:

1. Abra a conexão (`conexao.Open()`).
2. Defina a instrução SQL. Use apenas as colunas que você precisa.
3. Crie um `MySqlCommand` passando a instrução e a conexão.
4. Chame `ExecuteReader()` para obter um `MySqlDataReader`.
5. Percorra os resultados com `while (reader.Read())`.
6. Use `reader["NomeDaColuna"]` ou métodos como `GetString()` para extrair valores. Neste exemplo, convertemos o campo `Id` para inteiro com `Convert.ToInt32(...)`.

Código completo:

```
string sqlSelect = "SELECT Id, Nome, Idade FROM alunos";
using MySqlCommand cmd = new MySqlCommand(sqlSelect, conexao);
using MySqlDataReader reader = cmd.ExecuteReader();
while (reader.Read())
{
    int id = Convert.ToInt32(reader["Id"]);
    string nome = reader.GetString("Nome");

    string idade = reader.GetString("Idade");
    Console.WriteLine($"ID: {id}  Nome: {nome}  Idade: {idade}");
}
```

Explicação: O `MySqlDataReader` lê uma linha por vez, por isso usamos um `while`. A conversão do campo `Id` evita erros de tipo, pois `reader["Id"]` retorna um `object`.

5.2 Inserindo dados (INSERT)

Para adicionar um novo aluno, utilizamos o comando `INSERT INTO`. Para evitar **SQL Injection**, não concatenamos valores diretamente na SQL; usamos **parâmetros**. A

documentação do conector explica como adicionar parâmetros com `AddWithValue`. Siga os passos:

1. Escreva a instrução SQL usando placeholders (`@nome`, `@idade`).
2. Crie o `MySqlCommand` com a SQL e a conexão.
3. Use `cmd.Parameters.AddWithValue()` para cada valor.
4. Chame `ExecuteNonQuery()` para executar e receba o número de linhas afetadas.

Exemplo:

```
string sqlInsert = "INSERT INTO alunos (Nome, Idade) VALUES (@nome, @idade)";
using MySqlCommand cmd = new MySqlCommand(sqlInsert, conexao);
cmd.Parameters.AddWithValue("@nome", idadeDigitada);
cmd.Parameters.AddWithValue("@idade", nomeDigitado);
int linhas = cmd.ExecuteNonQuery();
Console.WriteLine($"{linhas} registro(s) inserido(s).");
```

> **Pergunta comum:** "Preciso informar a coluna `Id`?" Não. Se a coluna `Id` for auto-incrementada (como veremos na atividade prática), o MySQL gera o valor automaticamente.

5.3 Atualizando dados (UPDATE)

Para atualizar um registro existente, utilize a instrução `UPDATE` com uma cláusula `WHERE` para localizar o registro. O procedimento é o mesmo do INSERT:

```
string sqlUpdate = "UPDATE alunos SET Nome=@nome, Idade=@idade";
using MySqlCommand cmd = new MySqlCommand(sqlUpdate, conexao);
cmd.Parameters.AddWithValue("@nome", novoNome);
cmd.Parameters.AddWithValue("@idade", novaIdade);
cmd.Parameters.AddWithValue("@id", idParaAtualizar);
int alterados = cmd.ExecuteNonQuery();
Console.WriteLine($"{alterados} registro(s) atualizado(s).");
```

Sempre utilize o `WHERE` para não atualizar todos os registros inadvertidamente.

5.4 Excluindo dados (DELETE)

Para remover registros, use `DELETE FROM` com `WHERE Id=@id`. Os passos são idênticos: definir a SQL, adicionar o parâmetro e chamar `ExecuteNonQuery()`.

```
string sqlDelete = "DELETE FROM alunos WHERE Id=@id";
using MySqlCommand cmd = new MySqlCommand(sqlDelete, conexao);
cmd.Parameters.AddWithValue("@id", idParaExcluir);
int removidos = cmd.ExecuteNonQuery();
Console.WriteLine($"{removidos} registro(s) excluído(s).");
```

5.5 Obtendo valores agregados (COUNT, SUM)

O método `ExecuteScalar` retorna o valor da primeira coluna da primeira linha do resultado【927980072184630†L352-L399】. Isso é útil para obter contagens ou somatórios.

Exemplo: contar quantos alunos existem na tabela:

```
string sqlCount = "SELECT COUNT(*) FROM alunos";
using MySqlCommand cmd = new MySqlCommand(sqlCount, conexao);
object resposta = cmd.ExecuteScalar();
int total = Convert.ToInt32(resposta);
Console.WriteLine($"Total de alunos cadastrados: {total}");
```

> **Curiosidade:** Você pode utilizar `SUM(idade)` ou `AVG(idade)` para somar ou calcular a média de idades e acessar o resultado com `ExecuteScalar()`.

6. Boas práticas

Para escrever código de qualidade, siga estas orientações:

- **`using` sempre**: encapsule `MySqlConnection`, `MySqlCommand` e `MySqlDataReader` dentro de blocos `using` para liberar recursos corretamente.
- **Utilize parâmetros**: nunca concatene valores diretamente na string SQL; use `AddWithValue` ou `Add` para passar valores. Isso evita injeção de código e converte os tipos automaticamente【743827470201952†L270-L330】.
- **Trate exceções**: envolva operações de banco em `try/catch` para capturar mensagens de erro e informar o usuário de forma amigável.
- **Valide entradas**: verifique se o usuário digitou números onde necessário; utilize `int.TryParse` para converter strings em inteiros.
- **Comente o código**: escreva comentários explicando o que faz cada bloco de código. Isso facilita seu próprio entendimento e a manutenção futura.
- **Separe responsabilidades**: crie métodos diferentes para cada operação (cadastrar, listar, atualizar, excluir). Isso torna o código mais organizado e reutilizável.

7. Atividade prática – Gerenciador de Alunos

Agora que você aprendeu os conceitos fundamentais, é hora de praticar! O objetivo é criar uma aplicação de console para gerenciar alunos armazenados em um banco de dados.

7.1 Preparando o banco de dados

Execute os comandos abaixo em seu servidor MySQL (usando MySQL Workbench ou o terminal) para criar a base de dados `escola` e a tabela `alunos`.

```
CREATE DATABASE IF NOT EXISTS escola;
USE escola;
CREATE TABLE IF NOT EXISTS alunos (
    Id INT AUTO_INCREMENT PRIMARY KEY,
    Nome INT NOT NULL,
    Idade INT NOT NULL,
    Curso VARCHAR(50) NOT NULL
);
```

7.2 Estrutura sugerida da aplicação

Sua aplicação deve mostrar um **menu interativo** com opções. Uma sugestão:

1. **Cadastrar aluno** – solicita nome, idade e curso. Chame um método `CadastrarAluno` que executa o `INSERT`.
2. **Listar todos os alunos** – chama `ListarAlunos` e utiliza `SELECT`.
3. **Buscar aluno por nome** – pede um termo de busca e filtra `SELECT Nome LIKE`.
4. **Atualizar aluno** – solicita o ID do aluno, novos valores e chama `AtualizarAluno`.
5. **Excluir aluno** – solicita o ID e chama `ExcluirAluno`.
6. **Exibir total de alunos** – chama um método que usa `ExecuteScalar` para exibir o total.
7. **Sair** – encerra o programa.

Cada opção deve ser implementada em um método separado. Por exemplo, `static void CadastrarAluno(MySqlConnection conexao)` deve:

- Abrir a conexão.
- Ler dados do console (`Console.ReadLine()`).
- Criar o comando `INSERT` com parâmetros.
- Executar `ExecuteNonQuery()`.
- Exibir mensagem de sucesso ou erro.

Exercício: Adapte o código dos exemplos das seções 5.1–5.5 para preencher cada uma das funcionalidades acima.

7.3 Entregáveis

1. **Código-fonte (`Program.cs`)** – com comentários explicando cada etapa.
2. **Relatório reflexivo** – escreva um pequeno texto relatando as principais dificuldades encontradas e como as superou. Esse exercício de reflexão ajuda na fixação do conteúdo.