Classes para utilizar SQL Server no C#

1) SqlConnection
Namespace: System.Data.SqlClient
Para que serve: Representa uma conexão com o SQL Server. É responsável por abrir/fechar a conexão e gerenciar o *connection pool*.
Principais pontos:

- Use **using** para garantir fechamento/Dispose.
- Strings de conexão podem vir de appsettings.json/ConfigurationManager.
- Conexão só deve ficar aberta o tempo necessário.

Membros úteis:

- Open(), Close()
- State
- ConnectionString
- BeginTransaction()

```
using (var conn = new SqlConnection(connString))
{
   conn.Open();
   // ... comandos aqui
} // fecha/Dispose automaticamente
```

2) SqlCommand
Namespace: System.Data.SqlClient
Para que serve: Executa instruções SQL ou procedures no SQL Server, usando uma SqlConnection.
Principais pontos:

- Sempre **parametrize** comandos para evitar SQL Injection.
- Defina CommandType (Text ou StoredProcedure).
- Combine com ExecuteNonQuery, ExecuteScalar, ExecuteReader.

Membros úteis:

- CommandText, CommandType, Parameters
- ExecuteNonQuery() → DML (INSERT/UPDATE/DELETE)
- ExecuteScalar() → retorna o **primeiro valor da primeira linha**
- ExecuteReader() → retorna um SqlDataReader para leitura *forward-only*

```
using (var conn = new SqlConnection(connString))
using (var cmd = new SqlCommand(@"UPDATE Produtos SET Preco = @preco WHERE
Id = @id", conn))
{
   cmd.Parameters.AddWithValue("@preco", 19.90m);
   cmd.Parameters.AddWithValue("@id", 42);

   conn.Open();
   int afetados = cmd.ExecuteNonQuery();
}
```

3) StringBuilder
Namespace: System.Text
Para que serve: Montar strings de forma eficiente quando há muitas concatenações (evita criar várias strings temporárias).
Principais pontos:

- Use quando houver concatenações em loops ou textos longos.
- Métodos: Append, AppendLine, Clear, ToString.

Exemplo:
C#

```
var sb = new StringBuilder();
sb.AppendLine("Relatório");
foreach (var item in itens)
{
sb.AppendLine($"{item.Id} - {item.Nome}");
}

var sb = new StringBuilder();
sb.AppendLine("Relatório");
foreach (var item in itens)
{
   sb.AppendLine($"{item.Id} - {item.Nome}");
}
string relatorio = sb.ToString();
```

4) DataTable
Namespace: System.Data
Para que serve: Tabela de dados em memória (linhas/colunas). Útil para resultados tabulares desconectados, *binding* em UI, interoperar com APIs que esperam DataTable.
Principais pontos:

- Trabalha com DataColumn, DataRow.
- Pode ser preenchida por SqlDataAdapter ou manualmente.
- Ótima para cenários sem *ORM* ou para *bulk operations* (com SqlBulkCopy).

Membros úteis:

- Columns, Rows, Rows.Add(), Select(), DefaultView

Exemplo preenchendo via SqlDataAdapter:

```
var dt = new DataTable();
dt.Columns.Add("Id", typeof(int));
dt.Columns.Add("Nome", typeof(string));

dt.Rows.Add(1, "Teclado");
dt.Rows.Add(2, "Mouse");
```

ExecuteReader (método do SqlCommand)
Para que serve: Executa o comando e retorna um SqlDataReader para **leitura adiantada, somente leitura**, ideal para streams de dados grandes.

Principais pontos:

- Leia com while (reader.Read()).
- Acesse colunas por índice ou nome (reader.GetInt32(0), reader["Nome"]).
- Use CommandBehavior para otimizações (ex.: CommandBehavior.CloseConnection fecha a conexão ao fechar o reader).

Exemplo:

```
using (var conn = new SqlConnection(connString))
using (var cmd = new SqlCommand("SELECT Id, Nome, Preco FROM Produtos WHERE
Preco > @min", conn))
{
   cmd.Parameters.Add("@min", SqlDbType.Decimal).Value = 10.0m;

   conn.Open();
   using (var reader = cmd.ExecuteReader(CommandBehavior.CloseConnection))
   {
      while (reader.Read())
      {
         int id = reader.GetInt32(0);
         string nome = reader.GetString(1);
         decimal preco = reader.GetDecimal(2);
```