LINQ
Consulta Integrada à Linguagem


TM
LINQ
Microsoft®
.NET

O que é LINQ?

- LINQ, abreviação de (**Language Integrated Query**)

- O LINQ é um modelo e uma metodologia de programação da Microsoft que basicamente adiciona recursos de consulta formal às linguagens de programação baseadas no Microsoft .Net. O LINQ oferece uma sintaxe compacta, expressiva e inteligível para manipular dados. O valor real do LINQ vem de sua capacidade de aplicar a mesma consulta a um banco de dados SQL, a um DataSet, a uma matriz de objetos na memória e a muitos outros tipos de dados também.

O que é LINQ?

- LINQ fornece funcionalidades de consulta no nível da linguagem e uma API de função de ordem superior para C# e VB como uma maneira de escrever um código expressivo e declarativo.

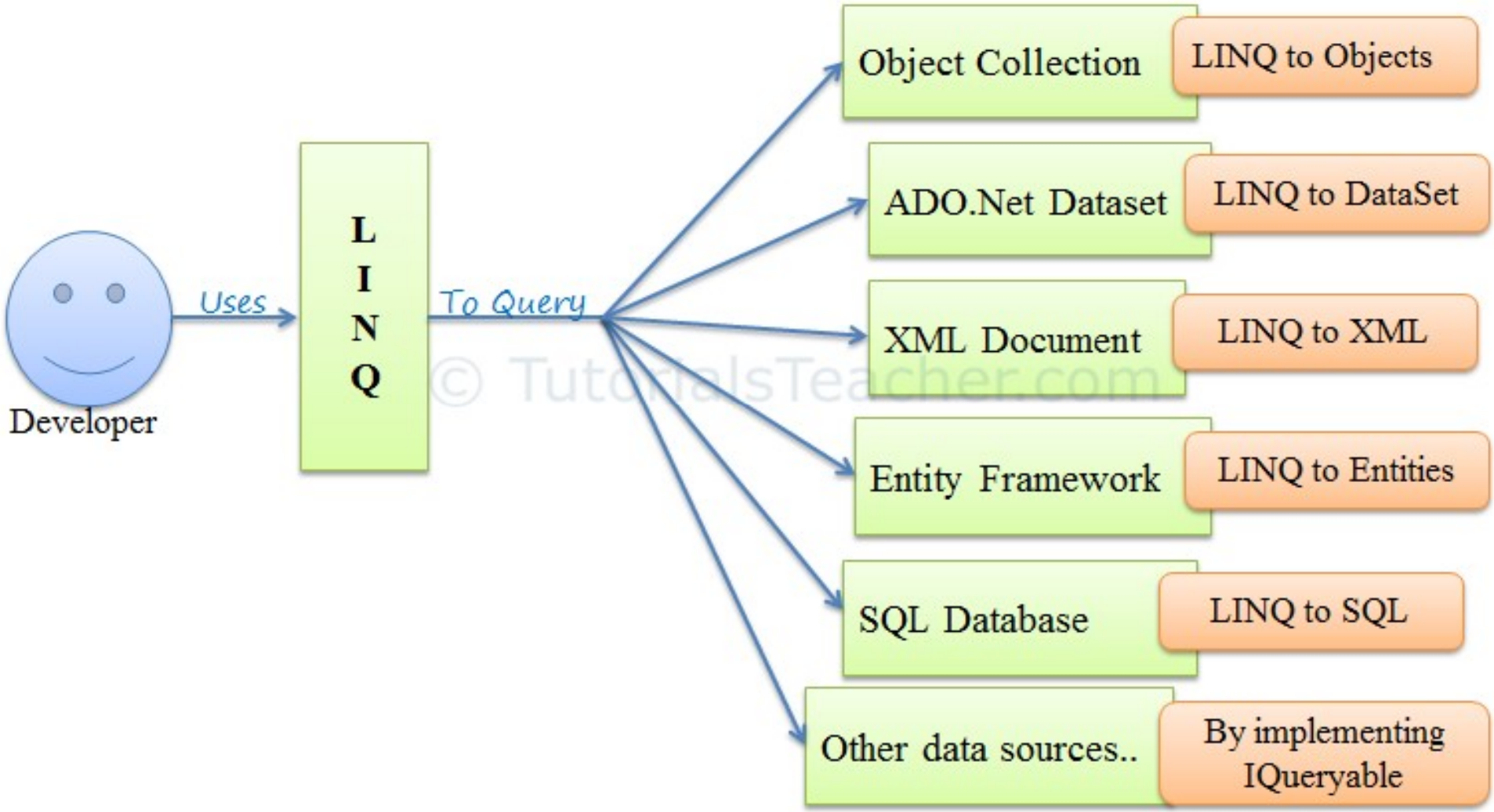
O que é LINQ?

- O LINQ usa uma sintaxe semelhante a SQL para tornar as expressões de consulta muito além dos recursos do SQL incorporado, conforme implementado nas linguagens de programação.

- O SQL incorporado usa uma sintaxe simplificada para adicionar instruções SQL a outras linguagens de programação, onde não há nenhuma tentativa de integrar essas instruções à sintaxe nativa e aos mecanismos de digitação.

O que é LINQ?

- Portanto, não é possível invocar estruturas de idioma nativo, como funções em instruções SQL incorporadas, como você pode usar o LINQ, porque ele é implementado para usar a sintaxe nativa, estruturas e mecanismos de digitação. Além disso, o LINQ pode ser usado para acessar todos os tipos de dados, enquanto o SQL incorporado é limitado a endereçar somente bancos de dados que podem manipular consultas SQL.



© TutorialsTeacher.com

VANTAGENS DO LINQ

- Trabalhar com o LINQ fácil, pois é semelhante à consulta de dados usando palavras inglesas normais (assim como a sintaxe SQL). O LINQ reduz drasticamente o número de linhas de código tornando o código limpo, fácil de ler e manter.
- Possui duas sintaxes principais para a consulta de dados. A sintaxe do SQL e a sintaxe dos métodos de extensão.

VANTAGENS DO LINQ

- No início do .NET, se seus dados estivessem armazenados em um banco de dados do SQL Server, você teria que usar os métodos de consulta fornecidos pelo ADO.NET para acessar dados com comandos SQL. Se Seus dados estivessem em um arquivo XML, você teria que usar outros métodos fornecidos por uma API diferente para consultar dados do documento XML. O LINQ forneceu uma API consistente para consultar dados dessas fontes e muitas outras fontes de dados.

VANTAGENS DO LINQ

- Com as Extensões Reativas (RX) do .Net, você tem uma maneira conveniente de ouvir eventos, lidar com erros, definir tempos limite e muito mais. Ele fornece uma maneira conveniente de declarar retornos de chamada e gerenciar a execução de código assíncrono. O RX estende o LINQ e, consequentemente, você tem a capacidade de transformar uma Consulta LINQ em uma sequencia observável e, em seguida, inscrever uma função como um observador dessa sequencia.

VANTAGENS DO LINQ

- Com as Extensões Reativas (RX) do .Net, você tem uma maneira conveniente de ouvir eventos, lidar com erros, definir tempos limite e muito mais. Ele fornece modos de declarar retornos de chamada e gerenciar a execução de código assíncrono. O RX estende o LINQ e, consequentemente, você tem a capacidade de transformar uma Consulta LINQ em uma sequencia observável e, em seguida, inscrever uma função como um observador dessa sequência.

VANTAGENS DO LINQ

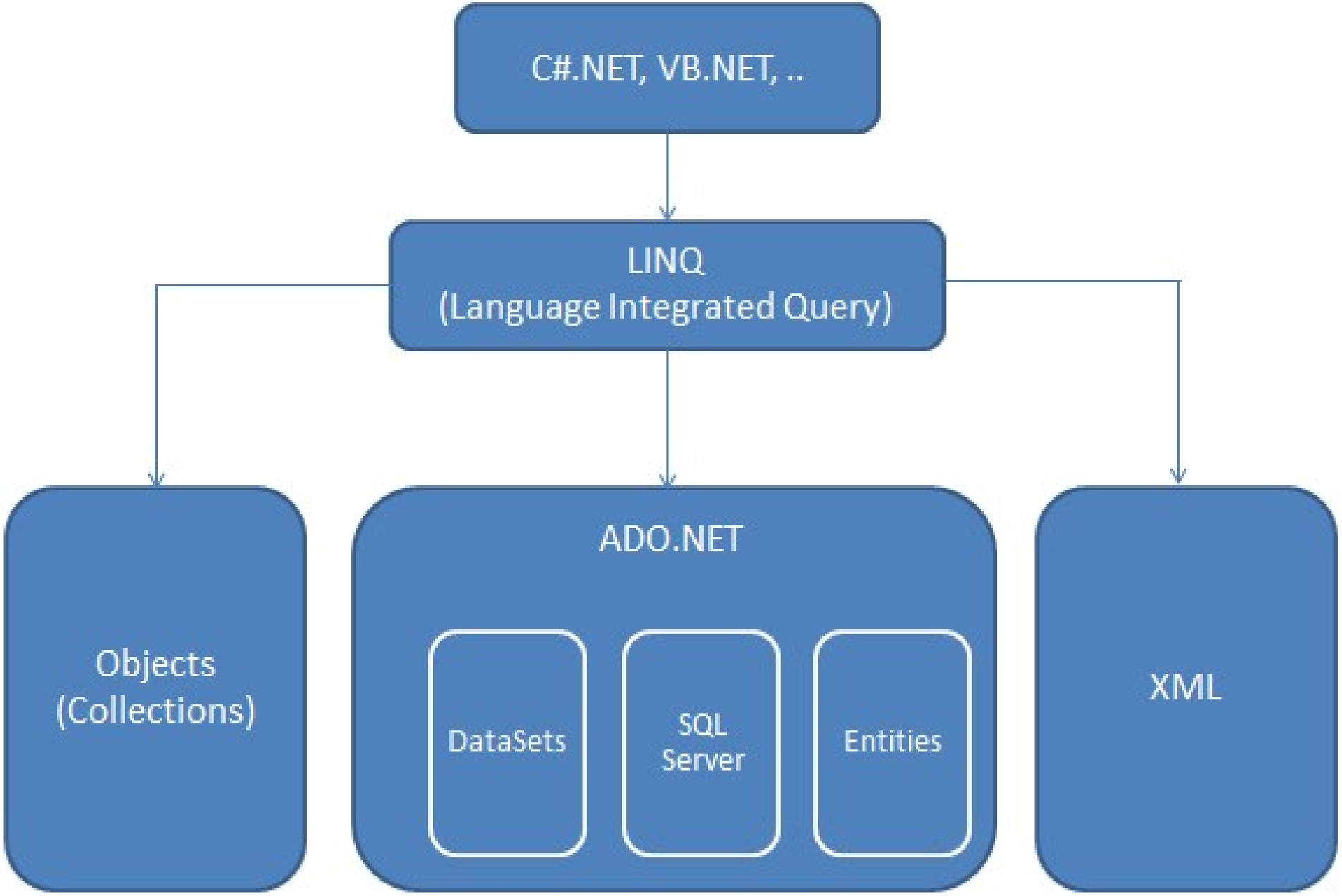
- Ao trabalhar com dados, você realizará operações complexas. Por exemplo, classificar os dados, filtrá-los ou projetá-los em uma estrutura de dados diferente. Fazer operações complexas que requerem junções. Realizar estas tarefas com o código normal pode ser um pouco complicado. Imagine que você estivesse escrevendo código para executar cada uma dessas tarefas explicitamente, não apenas desperdiçaria muito tempo, mas ler esse código mais tarde não seria nada fácil.

```
Listar <Man> men = new List <Man> {
    new Man {Name = "John", Age = 30},
    new Man {Name = "Peter", Age = 21},
    new Man {Name = "George", Age = 40}
};

// Retorna o nome do homem mais velho
var theNameOfTheOldestMan = men
    .OrderByDescending (man => man.Age)
    .Selecione (man => man.Name)
    .First ();
```

VANTAGENS DO LINQ

- Durante a programação, existem tarefas simples que você pode querer executar. Como somar todos os valores em uma lista, iterando através de uma lista e várias outras operações. Executar tarefas tão simples normalmente pode exigir várias linhas de código. Embora isso nem sempre seja aconselhado, o LINQ pode ser usado para executar essas operações simples.

```
// Percorrendo itens com FOREACH
List<int> myListOfNumbers = new List<int>();
myListOfNumbers.AddRange(
   new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }
);
myListOfNumbers.ToList()
   .ForEach(c => {
        //Perform apropriate action
   });

// Somando todos os itens de uma lista
var sum = myListOfNumbers.Sum();
```


C#.NET, VB.NET, ..
LINQ
(Language Integrated Query)
Objects
(Collections)
ADO.NET
DataSets
SQL
Server
Entities
XML

Consulta Integrada à Linguagem

```
// Sintaxe de consulta de nível de linguagem:
var linqExperts = from p in programmers
                  where p.IsNewToLINQ
                  select new LINQExpert(p);

//Mesmo exemplo usando a API IEnumerable<T>:
var linqExperts = programmers
                  .Where(p => p.IsNewToLINQ)
                  .Select(p => new LINQExpert(p));
```

LINQ EXPRESSIVA

- Imagine que você tem uma lista de animais de estimação, mas deseja convertê-la em um dicionário em que você pode acessar um animal de estimação diretamente por seu valor RFID.

LINQ EXPRESSIVA

```
var petLookup = new Dictionary<int, Pet>();
foreach (var pet in pets)
{
   petLookup.Add(pet.RFID, pet);
}
```

- A intenção do código não é criar um novo Dictionary<int, Pet> e adicionar a ele por meio de um loop, é converter uma lista existente em um dicionário. A LINQ preserva a intenção enquanto o código imperativo não.

```
var petLookup = pets.ToDictionary(pet => pet.RFID);
```

SINTAXE DE API x SINTAXE DE CONSULTA

```
var filteredItems = myItems.Where(item =>
    item.Foo);
```

```
var filteredItems = from item in myItems
                    where item.Foo
                    select item;
```

SINTAXE DE API x SINTAXE DE CONSULTA

- A sintaxe da API não é apenas uma maneira mais concisa de fazer a sintaxe de consulta?

- A sintaxe de consulta permite o uso da cláusula **let**, que permite que você introduza e associe uma variável no escopo da expressão, usando-a em partes subsequentes da expressão. É possível reproduzir o mesmo código com apenas a sintaxe da API, mas mais provavelmente levará a um código que é difícil de ler.

SINTAXE DE API x SINTAXE DE CONSULTA

- Portanto, isso levanta a questão, **você deve usar apenas a sintaxe de consulta?**

SINTAXE DE API x SINTAXE DE CONSULTA

- A resposta para essa pergunta será **sim** se...
  - Sua base de código já usar a sintaxe de consulta
  - Você precisar definir o escopo de variáveis em suas consultas devido à complexidade
  - Você preferir a sintaxe de consulta e ela não for distraí-lo da base de código

SINTAXE DE API x SINTAXE DE CONSULTA

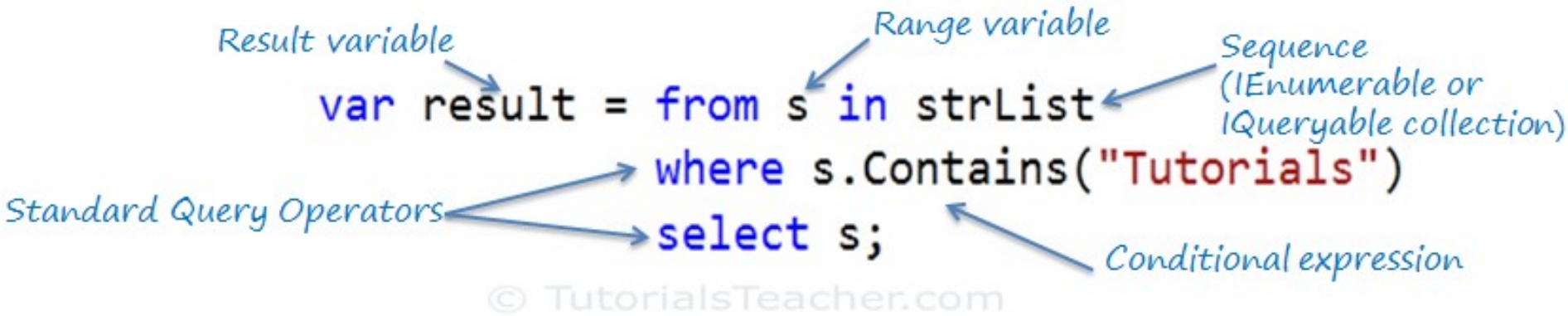
- A resposta para essa pergunta será **não** se...
  - Sua base de código já usar a sintaxe de API
  - Você não precisar definir o escopo de variáveis em suas consultas
  - Você preferir a sintaxe de API e ela não for distraí-lo da base de código

Query Syntax

```
var vendorQuery = from v in vendors
                  where v.CompanyName.Contains("Toy")
                  orderby v.CompanyName
                  select v;
```

Method Syntax

```
var vendorQuery = vendors
                  .Where(v => v.CompanyName.Contains("Toy"))
                  .OrderBy(v=> v.CompanyName);
```


Result variable
Range variable
Sequence
(IEnumerable or
IQueryable collection)
var result = from s in strList
where s.Contains("Tutorials")
select s;
Standard Query Operators
Conditional expression
© TutorialsTeacher.com

EXEMPLOS WHERE, SELECT, AGGREGATE

```
// Filtering a list
var germanShepards = dogs.Where(
    dog => dog.Breed == DogBreed.GermanShepard
);

// Using the query syntax
var queryGermanShepards =
    from dog in dogs
    where dog.Breed == DogBreed.GermanShepard
    select dog;
```

EXEMPLOS WHERE, SELECT, AGGREGATE

```
// Mapping a list from type A to type B
var cats = dogs.Select(
   dog => dog.TurnIntoACat()
);

// Using the query syntax
var queryCats = from dog in dogs
                select dog.TurnIntoACat();
```

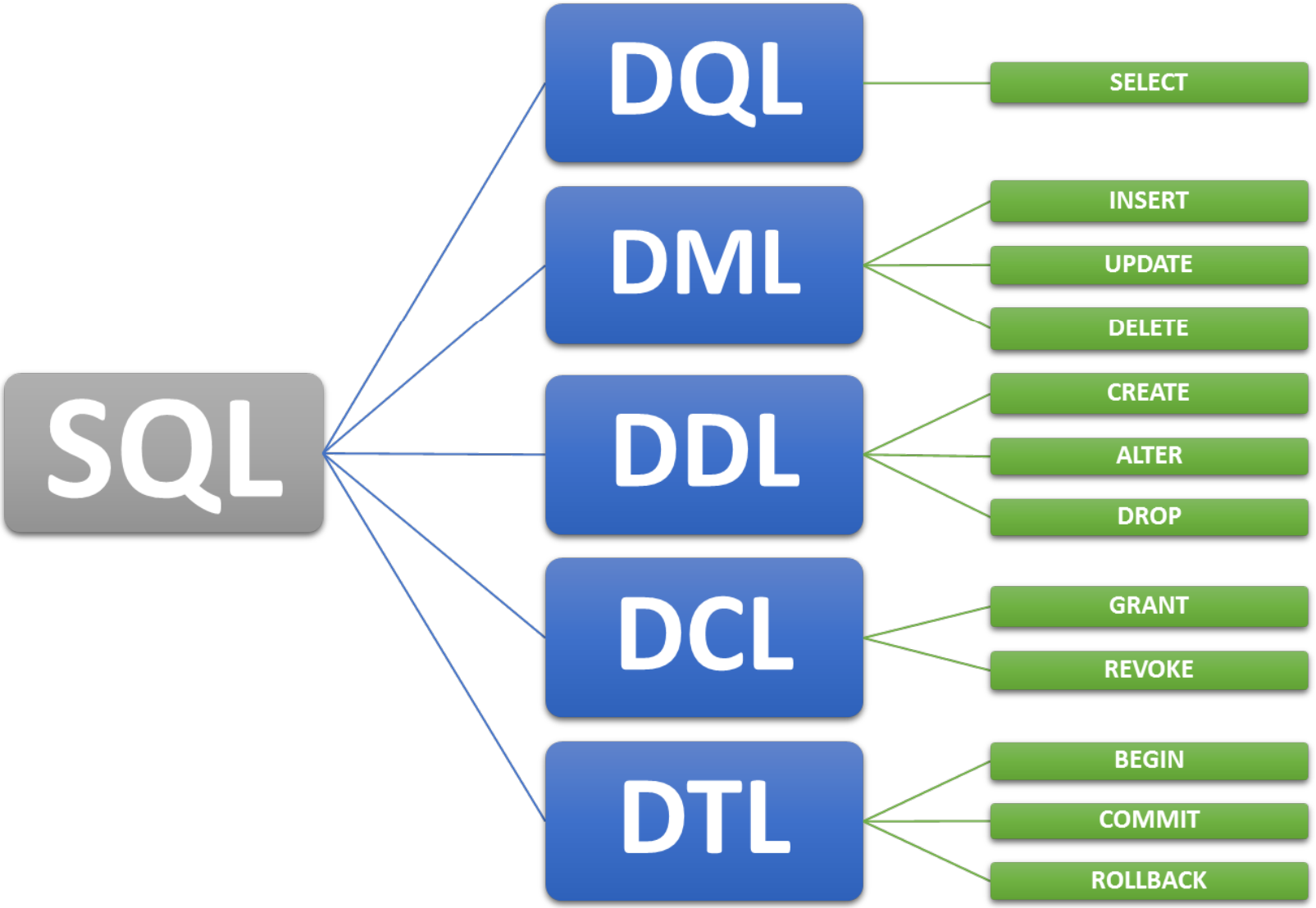
EXEMPLOS WHERE, SELECT, AGGREGATE

```
// Summing the lengths of a set of strings
int seed = 0;
int sumOfStrings =
    strings.Aggregate(
        seed, (s1, s2) => s1.Length + s2.Length
    );
```

INTRODUÇÃO AO SQL

- **A linguagem SQL é o recurso mais conhecido por DBAs e programadores para a execução de comandos em bancos de dados relacionais**. É por meio dela que criamos tabelas, colunas, índices, atribuímos permissões a usuários, bem como realizamos consultas a dados. Enfim, é utilizando a SQL que “conversamos” com o banco de dados.

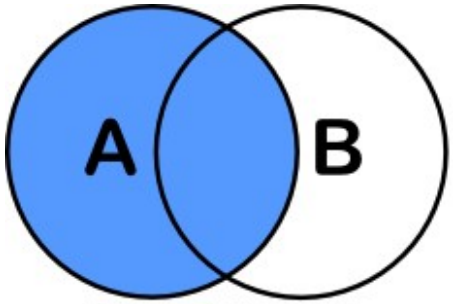
INTRODUÇÃO AO SQL



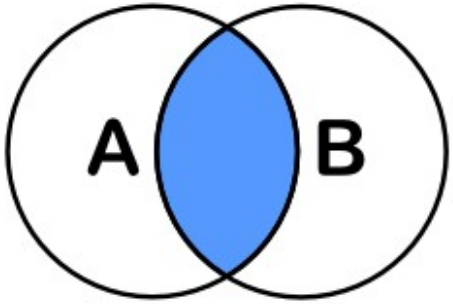
INTRODUÇÃO AO SQL

- **DQL - Linguagem de Consulta de Dados -** Define o comando utilizado para que possamos consultar (SELECT) os dados armazenados no banco;
- **DML - Linguagem de Manipulação de Dados -** Define os comandos utilizados para manipulação de dados no banco (INSERT, UPDATE e DELETE);
- **DDL - Linguagem de Definição de Dados -** Define os comandos utilizados para criação (CREATE) de tabelas, views, índices, atualização dessas estruturas (ALTER), assim como a remoção (DROP);
- **DCL - Linguagem de Controle de Dados -** Define os comandos utilizados para controlar o acesso aos dados do banco, adicionando (GRANT) e removendo (REVOKE) permissões de acesso;
- **DTL - Linguagem de Transação de Dados -** Define os comandos utilizados para gerenciar as transações executadas no banco de dados, como iniciar (BEGIN) uma transação, confirmá-la (COMMIT) ou desfazê-la (ROLLBACK).

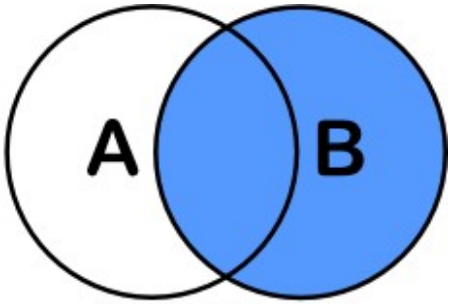
CHEATSHEET SQL JOINS



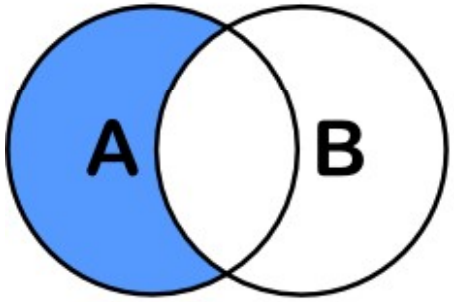

```
SELECT <auswahl>
FROM tabelleA A
LEFT JOIN tabelleB B
ON A.key = B.key
```



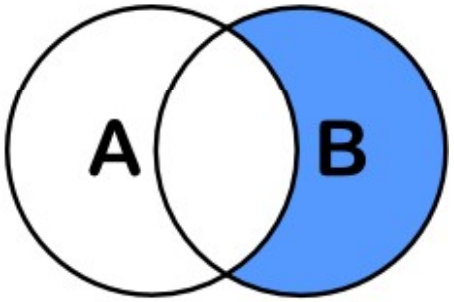

```
SELECT <auswahl>
FROM tabelleA A
INNER JOIN tabelleB B
ON A.key = B.key
```



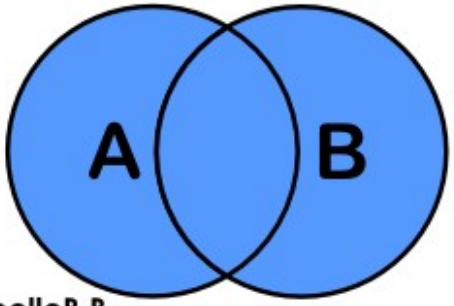

```
SELECT <auswahl>
FROM tabelleA A
RIGHT JOIN tabelleB B
ON A.key = B.key
```



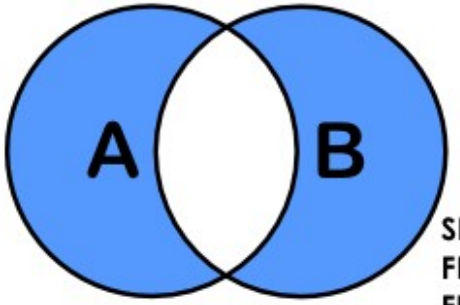

```
SELECT <auswahl>
FROM tabelleA A
LEFT JOIN tabelleB B
ON A.key = B.key
WHERE B.key IS NULL
```




```
SELECT <auswahl>
FROM tabelleA A
RIGHT JOIN tabelleB B
ON A.key = B.key
WHERE A.key IS NULL
```




```
SELECT <auswahl>
FROM tabelleA A
FULL OUTER JOIN tabelleB B
ON A.key = B.key
```




```
SELECT <auswahl>
FROM tabelleA A
FULL OUTER JOIN tabelleB B
ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL
```

ATIVIDADE

- Apresentação em dupla sobre Capítulo 12 do livro Professional C# 7 and .NET Core 2.0 – LINQ – Language Integrated Query.

Referências auxiliares

- https://doumer.me/5-reasons-why-linq-is-addictive-for-every-net-developer/

- 101 LINQ Samples
  - https://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b

- Introdução ao SQL
  - https://www.devmedia.com.br/guia/guia-completo-de-sql/38314