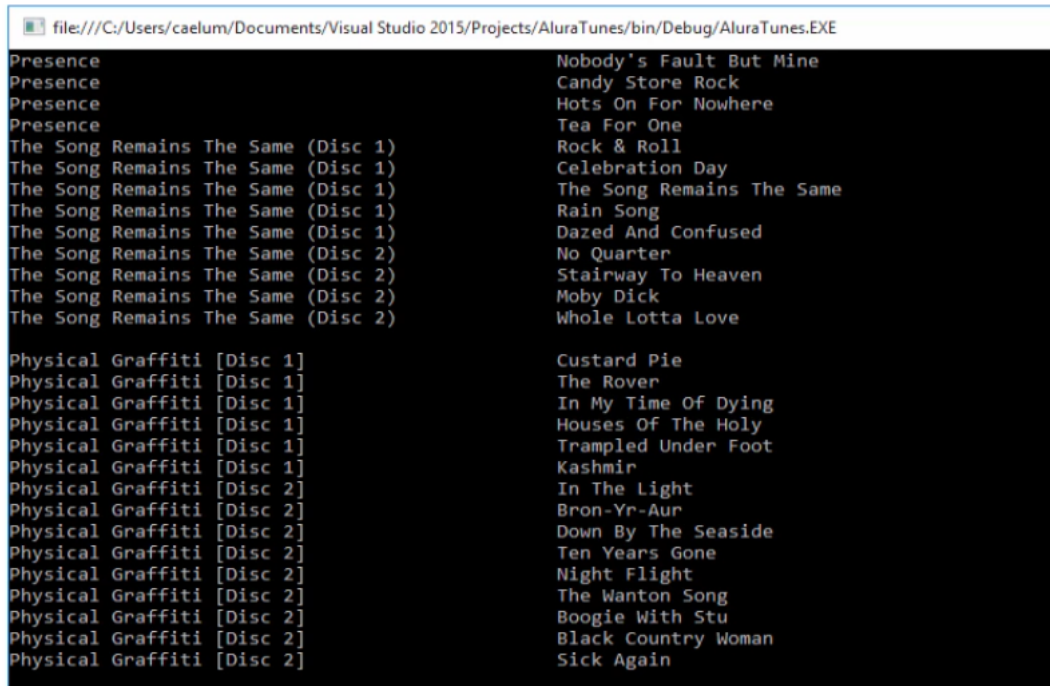


## 5 - Linq to entities orderby

## Transcrição

Na última aula, criamos uma função para trazer as faixas de músicas filtradas por nome de banda e nome do álbum. Apesar do sucesso da consulta, o cliente achou um pouco estranho a falta de ordenamento dos resultados, seu questionamento foi: por que as faixas de música não estão distribuídas em ordem alfabética?



Por enquanto os objetos estão organizados por ordem de armazenamento. Assim, é preciso modificar a consulta para que ela traga os elementos por ordem alfabética. Para fazer isso vamos adicionar uma instrução na query. Como queremos ordenar primeiro por nome de álbum e depois por faixa de música, vamos modificar a query utilizando o método `orderBy()` .

Passaremos para ele a seguinte instrução:

```
q => q.Album.Titulo
```

Teremos o seguinte:

```
if (!string.IsNullOrEmpty(buscaAlbum))
{
    query = query.where(q => q.Album.Titulo.Contains(buscaAlbum));
}

query = query.OrderBy(q => q.Album.Titulo);
```

Assim, teremos uma ordem estabelecida por nome de álbum! Para que a ordem se estabeleça também entre as faixas é preciso modificar mais de uma vez a `query`. Portanto, além do `OrderBy()` por título, colocaremos a ordenação por nome de faixa, dessa maneira, acrescentaremos o `.ThenBy` que estabelece uma distribuição secundária. Isto significa que ele só funcionará quando uma ordenação primária for estabelecida. Por isso, adicionaremos o `ThenBy()` e passaremos para ele a outra propriedade que deve ser ordenada, a `q => q.Nome`. Teremos:

```
if (!string.IsNullOrEmpty(buscaAlbum))
{
    query = query.where(q => q.Album.Titulo.Contains(buscaAlbum));
}
}
```

```
query = query.OrderBy(q => q. Album.Titulo).ThenBy(q => q.Nome);
```

Rodando temos:

```
file:///C:/Users/caelum/Documents/Visual Studio 2015/Projects/AluraTunes/bin/Debug/AluraTunes.EXE
Presence           Hots On For Nowhere
Presence           Nobody's Fault But Mine
Presence           Royal Orleans
Presence           Tea For One
The Song Remains The Same (Disc 1)  Celebration Day
The Song Remains The Same (Disc 1)  Dazed And Confused
The Song Remains The Same (Disc 1)  Rain Song
The Song Remains The Same (Disc 1)  Rock & Roll
The Song Remains The Same (Disc 1)  The Song Remains The Same
The Song Remains The Same (Disc 2)  Moby Dick
The Song Remains The Same (Disc 2)  No Quarter
The Song Remains The Same (Disc 2)  Stairway To Heaven
The Song Remains The Same (Disc 2)  Whole Lotta Love
Physical Graffiti [Disc 1]          Custard Pie
Physical Graffiti [Disc 1]          Houses Of The Holy
Physical Graffiti [Disc 1]          In My Time Of Dying
Physical Graffiti [Disc 1]          Kashmir
Physical Graffiti [Disc 1]          The Rover
Physical Graffiti [Disc 1]          Trampled Under Foot
Physical Graffiti [Disc 2]          Black Country Woman
Physical Graffiti [Disc 2]          Boogie With Stu
Physical Graffiti [Disc 2]          Bron-Yr-Aur
Physical Graffiti [Disc 2]          Down By The Seaside
Physical Graffiti [Disc 2]          In The Light
Physical Graffiti [Disc 2]          Night Flight
Physical Graffiti [Disc 2]          Sick Again
Physical Graffiti [Disc 2]          Ten Years Gone
Physical Graffiti [Disc 2]          The Wanton Song
```

O resultado mostra que o ordenamento de álbuns se mantém! Repare que na segunda coluna, ocorre uma distribuição da letra **c** até a **t** no primeiro disco. Depois, da letra **b** até **t**, no segundo disco. Ou seja, realizaremos o desejo cliente!

E se, em vez de uma ordem estabelecida em ordem alfabética pudéssemos realizar uma ordem de maneira decrescente?

Para isto, vamos manter a `query` igual até:

```
OrderBy(q => q.Album.Titulo)
```

A partir disso, vamos alterar o `ThenBy()` por:

```
ThenByDescending(q => q.Nome)
```

Com a alteração, o código ficará da seguinte maneira:

```
query = query.OrderBy(q => q.Album.Titulo).ThenByDescending(q => q.Nome);
```

Teremos um resultado no qual a primeira coluna exibirá os nomes dos álbuns ordenados e na segunda, teremos um ordenamento secundário, o nome das faixas estará de maneira decrescente!

```
Physical Graffiti [Disc 1]    Trampled Under Foot
Physical Graffiti [Disc 1]    The Rover
Physical Graffiti [Disc 1]    Kashmir
Physical Graffiti [Disc 1]    In My Time Of Dying
Physical Graffiti [Disc 1]    Houses Of The Holy
Physical Graffiti [Disc 1]    Custard Pie
Physical Graffiti [Disc 2]    The Wanton Song
Physical Graffiti [Disc 2]    Ten Years Gone
Physical Graffiti [Disc 2]    Sick Again
Physical Graffiti [Disc 2]    Night Flight
Physical Graffiti [Disc 2]    In The Light
Physical Graffiti [Disc 2]    Down By The Seaside
Physical Graffiti [Disc 2]    Bron-Yr-Aur
Physical Graffiti [Disc 2]    Boogie With Stu
Physical Graffiti [Disc 2]    Black Country Woman
```

Assim, o problema do cliente está resolvido!

Retornando ao código, podemos fazer uma pergunta: Por que colocar a ordenação em uma `query`, uma linha abaixo da `var query`? Por que não colocar essas informações junto do `where`?

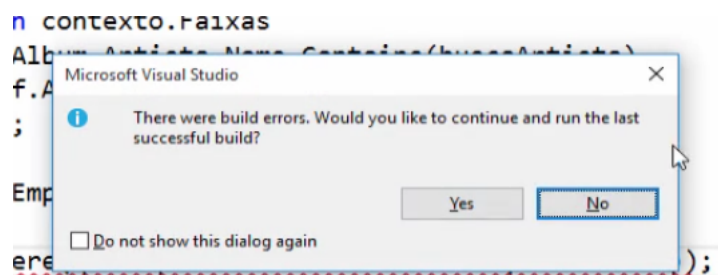
Utilizaremos a sintaxe de método e, acima, a da consulta. Caso movêssemos o `OrderBy` para a Sintaxe de Consulta ficaríamos com o seguinte código:

```
var query = from f in contexto.Faixas
            where f.Album.Artista.Nome.Contains(buscaArtista)
            orderby f.Album.Titulo, f.Nome
            select f;
```

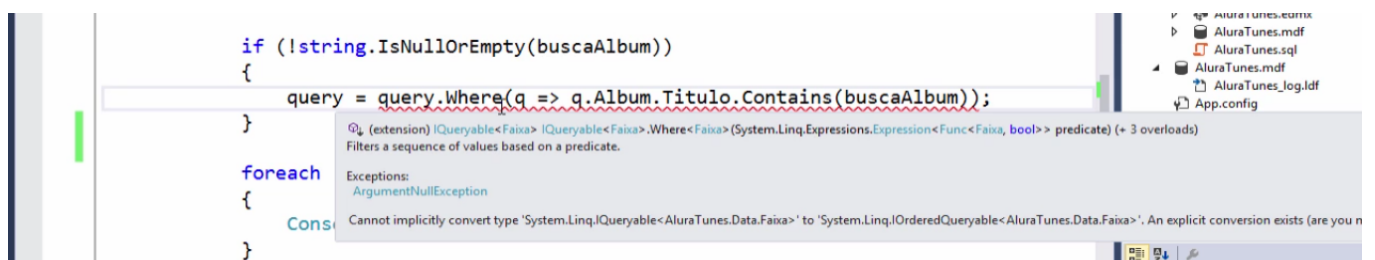
Antes de rodar, é preciso excluir a linha da `query` que contem o `OrderBy`:

```
query = query.OrderBy(q => q.Album.Titulo).ThenByDescending(q => q.Nome);
```

E ao executarmos o código, teremos a seguinte mensagem:



Vamos observar a mensagem que aparece para entender o que fizemos de errado:



A mensagem diz que o filtro `where` não pode ser colocado após uma consulta ordenada. Pensando em uma consulta SQL, é exatamente isso que ocorre! Se fizermos uma consulta SQL e colocarmos o `OrderBy`, não é possível inserir depois o `where`.

O LINQ é inteligente a ponto de saber que a consulta SQL gerada não aceita esse tipo de sintaxe. Para resolver a situação é preciso mover a seguinte linha para antes do `OrderBy` :

```
query. Where(q => q.Album.Titulo.Contains(buscaAlbum))
```

O problema é que a busca pelo `Album` só será feita se passarmos o nome do álbum para a função. Caso isso não seja feito, o filtro será ignorado e a consulta permanecerá. Assim, adicionaremos o `&&` depois de:

```
where f.Album.Artista.Nome.Contains(buscaArtista)
```

Acrescentaremos junto disso a seguinte condição:

```
q => q.Album.Titulo.Contains(buscaAlbum)
```

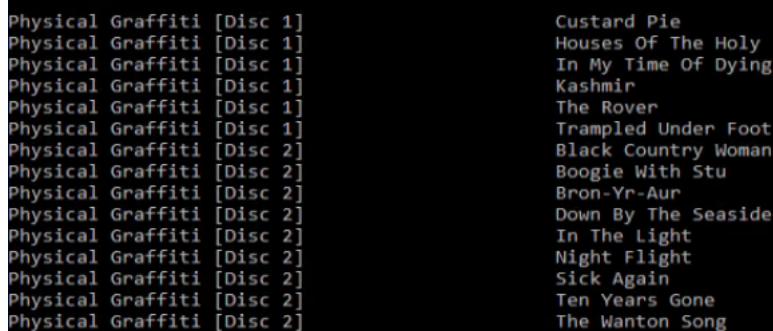
Como não podemos apenas apagar a condição, vamos inserir ela inline:

```
&& !string.IsNullOrEmpty(buscaAlbum)
```

No fim, acrescentaremos o `: true` , removemos o `if` e o `query.where` . Lembrando que a `string` deve ficar entre parênteses. Teremos:

```
var query = from f in contexto.Faixas
             where f.Album.Artista.Nome.Contains(buscaArtista)
             && !(string.IsNullOrEmpty(buscaAlbum) ?
                 q.Album.Titulo.Contains(buscaAlbum) : true)
             orderby f.Album.Titulo, f.Nome
             select f;
```

Ao rodarmos isso, teremos a consulta:



Physical Graffiti [Disc 1]	Custard Pie
Physical Graffiti [Disc 1]	Houses Of The Holy
Physical Graffiti [Disc 1]	In My Time Of Dying
Physical Graffiti [Disc 1]	Kashmir
Physical Graffiti [Disc 1]	The Rover
Physical Graffiti [Disc 1]	Trampled Under Foot
Physical Graffiti [Disc 2]	Black Country Woman
Physical Graffiti [Disc 2]	Boogie With Stu
Physical Graffiti [Disc 2]	Bron-Yr-Aur
Physical Graffiti [Disc 2]	Down By The Seaside
Physical Graffiti [Disc 2]	In The Light
Physical Graffiti [Disc 2]	Night Flight
Physical Graffiti [Disc 2]	Sick Again
Physical Graffiti [Disc 2]	Ten Years Gone
Physical Graffiti [Disc 2]	The Wanton Song

O resultado mostra uma ordem por nome de álbum e na segunda coluna, por nome de faixa!