01

1 - Comprou tambem

Transcrição

O próximo pedido do cliente é para criarmos uma consulta que será utilizada pela área de marketing da empresa. A consulta que produzirmos será aplicada para trabalhar na técnica de análise de afinidade.

Lembrando que análise de afinidade diz respeito aos produtos similares que um site recomenda quando compramos determinado item. Um site de vendas normalmente mostra que para determinada compra existem outros produtos que são comumente adquiridos. Por exemplo, ao comprarmos uma cafeteira, são mostrados filtros, cápsulas de café etc.

Será preciso criar uma consulta para o cliente aplicar no site. Vamos começar introduzindo uma variável que armazene o produto que desejamos analisar, no caso, as faixas de música. Faremos a análise de afinidade a partir da música *Smells Like Teen Spirit* e dentro da variável que criarmos, vamos inserir o contexto *Entity Framework*, um filtro para buscar a música - e dentro deste, adicionaremos uma expressão `lambda`. Como queremos que o resultado da consulta busque os `Ids` dos itens, utilizaremos o `Select e f.FaixaId`. O filtro será englobado em na variável `faixaIds`:

```
static void Main(String[] args)
{
    var nomeDaMusica = "Smells Like Teen Spirit"

    using (var contexto = new AluraTunesForEntities())
    {
        var faixaIds =
        contexto.Faixas.Where(f => f.Nome == nomeDaMusica).Select(f => f.FaixaId);
    }
}
```

Falta trazer os itens de nota fiscal que contem a faixa de música *Smells Like Teen Spirit*. Então, adicionamos outra variável `query` e adicionaremos a seguinte linha:

```
from comprouItem in contexto.ItemNotaFiscals
```

Também adicionaremos o `where` e incluir o `faixaId` e o `where`. No fim, vamos adicionar o `select comprouItem`:

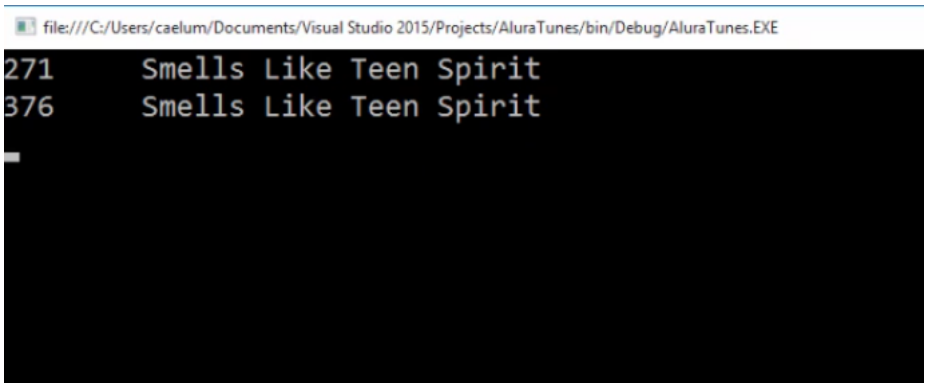
```
static void Main(String[] args)
{
    var nomeDaMusica = "Smells Like Teen Spirit"

    using (var contexto = new AluraTunesForEntities())
    {
        var faixaIds =
        contexto.Faixas.Where(f => f.Nome == nomeDaMusica).Select(f => f.FaixaId);

        var query =
            from comprouItem in contexto.ItemNotaFiscals
            where faixasId.Contains(comprouItem.FaixaId)
            select comprouItem;
    }
```

```
    Console.ReadKey();
}
```

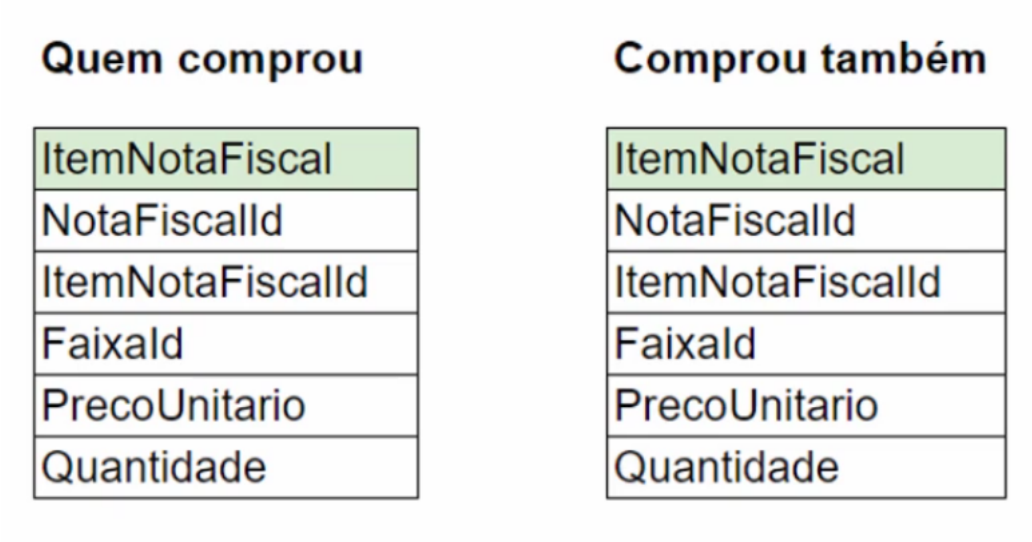
Falta acrescentar o `foreach ()` e o `Console.WriteLine()` para imprimir o resultado - no último, nós vamos inserir a formatação das colunas e os itens, `"{0}\t{1}"` e `item.NotaFiscalid, item.Faixa.Nome` , respectivamente:

```
static void Main(String[] args)
{
    var nomeDaMusica = "Smells Like Teen Spirit"

    using (var contexto = new AluraTunesForEntities())
    {
        var faixaIds =
        contexto.Faixas.Where(f => f.Nome == nomeDaMusica).Select(f => f.FaixaId);

        var query =
            from comprouItem in contexto.ItemNotaFiscals
            where faixasId.Contains(comprouItem.FaixaId)
            select comprouItem;

    foreach (var item in query)
    {
        Console.WriteLine("{0}\t{1}", item.NotaFiscalid, item.Faixa.Nome)
    }

    Console.ReadKey();
}
```

Teremos o seguinte resultado quando rodarmos a aplicação:



Temos o nome da música e também o seu `Id` ! O próximo passo é trazer as faixas comumente adquiridas com a música "Smells Like Teen Spirit"! Para compreender o que vamos fazer, observe o seguinte diagrama:



Temos que criar um tipo de relacionamento que ligue entre si duas entidades similares. Esse tipo de relação chama-se **self join** ou "auto relacionamento". Assim, precisamos acrescentar ao código um `join` que deve ser utilizado para ligar itens de nota fiscal. Portanto, vamos ter a seguinte linha:

```
join comprouTambem in contexto.ItemNotasFiscals
```

E como estamos falando de `itemNotaFiscal` , teremos:

```
on comprou.Item.NotaFiscalId equals comprouTambem.NotaFiscalId
```

Com as alterações, o código ficará assim:

```
var query =
    from comprouItem in contexto.ItemNotaFiscals
    join comprouTambem in contexto.ItemNotaFiscals
        on comprou.Item.NotaFiscalId equals comprouTambem.NotaFiscalId
    where faixasId.Contains(comprouItem.FaixaId)
    contexto.ItemNotasFiscals
    select comprouItem;
```

Ao rodar a consulta, o resultado terá vários itens com o mesmo nome da música, `Smells Like Teen Spirit` ! Para resolvermos isto, vamos trocar o `comprouItem` do `select` por `comprouTambem` :

```
var query =
            from comprouItem in contexto.ItemNotaFiscals
            join comprouTambem in contexto.ItemNotaFiscals
                on comprou.Item.NotaFiscalId equals comprouTambem.NotaFiscalId
            where faixasId.Contains(comprouItem.FaixaId)
            contexto.ItemNotasFiscals
            select comprouTambem;
```

Assim, quando executarmos o código, o resultado são diversas músicas distintas! O problema do relatório é que ele traz a própria música para a qual queremos fazer análise de afinidade, a "Smells Like Teen Spirit":

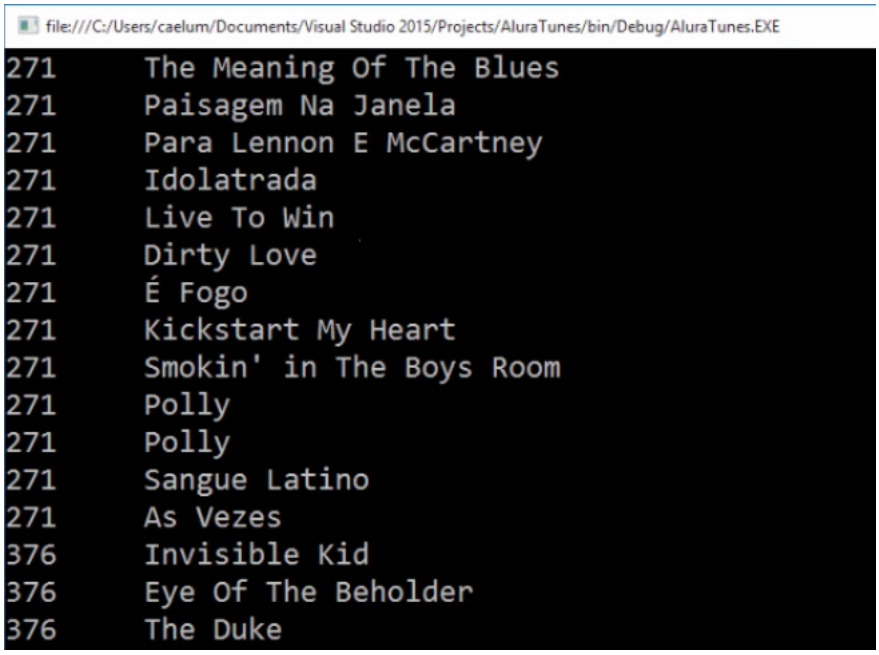
Para solucionar a questão, incluiremos uma cláusula. É preciso definir que o `comprouTambem` deve ser diferente do produto `comprouItem` . Assim, abaixo do `where` , nós vamos inserir o operador lógico `&&` e logo em seguida colocaremos a regra a seguinte regra:

```
comprouItem.FaixaId !=comprouTambem.FaixaId
```

Com as alterações, o trecho ficou assim:

```
var query =
            from comprouItem in contexto.ItemNotaFiscals
            join comprouTambem in contexto.ItemNotaFiscals
                on comprou.Item.NotaFiscalId equals comprouTambem.NotaFiscalId
            where faixasId.Contains(comprouItem.FaixaId)
                && comprouItem.FaixaId !=comprouTambem.FaixaId
            contexto.ItemNotasFiscals
            select comprouTambem;
```

Ao rodarmos a aplicação teremos o seguinte:



O resultado serão diversas músicas que foram compradas concomitantes à aquisição de "Smells Like Teen Spirit", mas sem trazer a própria música!

Nesta aula vimos como criar uma consulta, que será utilizado pela área de Marketing para fazer análise de afinidade, utilizando `self join` ou `auto-relacionamento`.