01

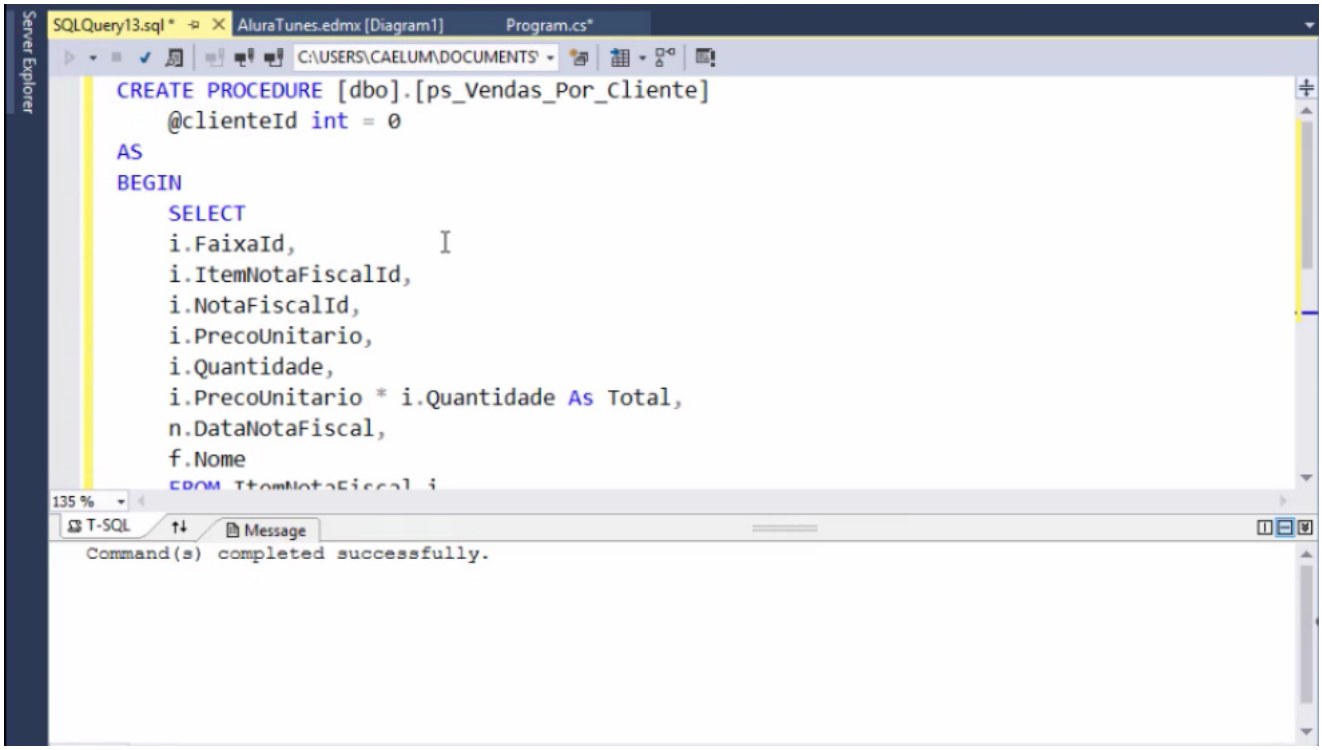
1 - Linq to entities stores procedure

Transcrição

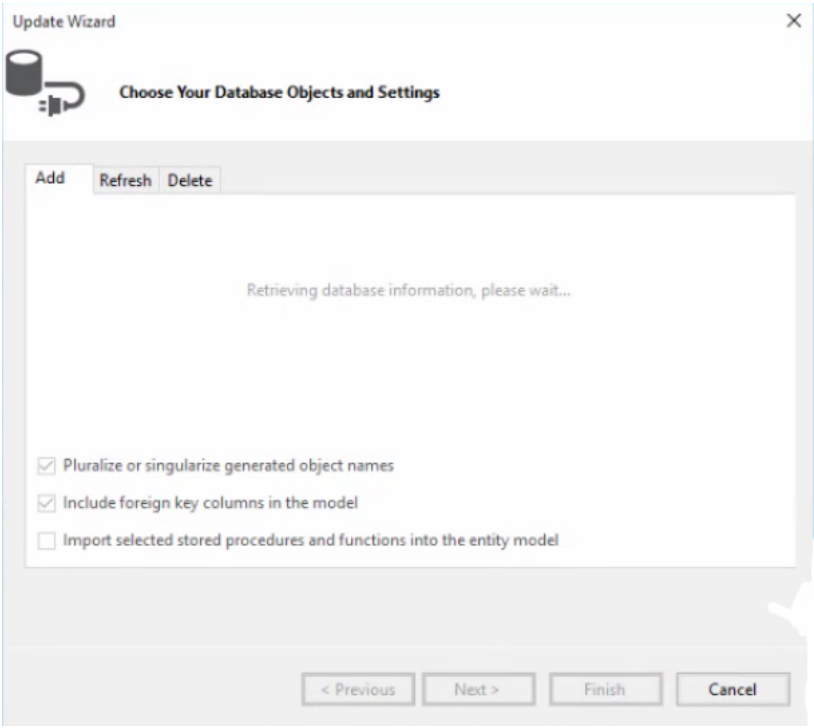
A tarefa proposta para esta aula é a de criar um relatório que traga as vendas da **Alura Tunes** agrupadas por ano e mês. Para alcançar esse objetivo, utilizaremos os dados da `stored procedure` que é criada e mantida pelo *Database Administrator* (DBA, ou o Administrador de Banco de dados).

No código existe uma `procedure` , a `ps_Vendas_Por_Clientes` . Ela calcula o valor `Total` do `Item` multiplicando o valor unitário pela quantidade:

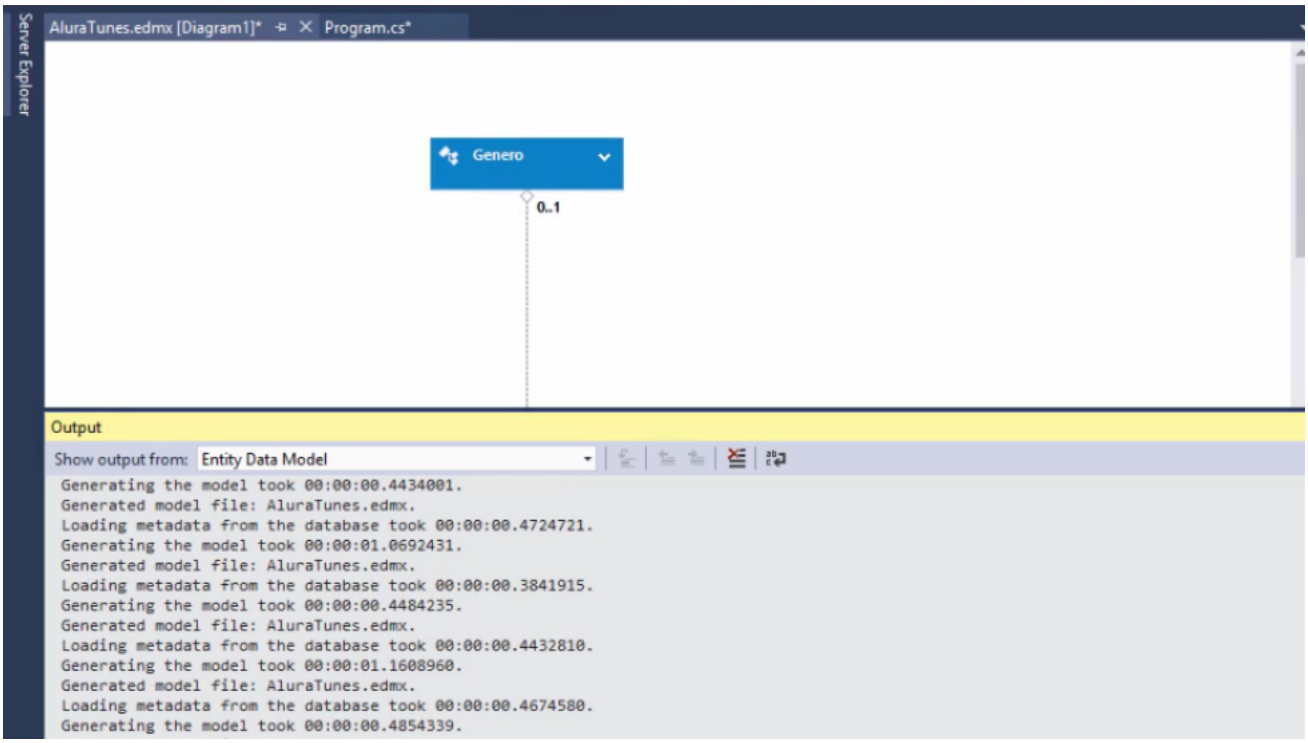
O próximo passo é criar uma `procedure` ! Para fazer isso clicaremos com o botão esquerdo em cima de `AluraTunes.mdf` e escolheremos o "New Query". Assim, terminamos de criar a `stored procedure` !



Agora, é preciso atualizar o modelo do `Entity Framework` para sinalizar que existe uma `procedure` com a qual, a partir desse momento, deve-se trabalhar. Abriremos o `AluraTunes.edmx` , em seguida, vamos clicar com o botão esquerdo e escolheremos a opção "Update Model from Database...". Assim, aparecerá a seguinte janela:



Ao selecionarmos o "Finish", a atualização será concluída:



Na aplicação, poderemos começar definindo o `Id` do cliente que deve ser acessado pela `stored procedure`. Assim, colocaremos o `int clienteId = 17` e vamos adicionar também o `using` e junto disso, acrescentaremos o contexto:

```
(var contexto = new AluraTunesEntities)
```

Também vamos incluir `stored procedure` através de `contexto.ps_Vendas_Por_Clientes`.

A `stored procedure` necessitará do `Id` do cliente. Assim, vamos acrescentar o `clienteId` e o `select`. Colocaremos tudo isso dentro da variável `vendasPorCliente`. Nosso código ficará da seguinte forma:

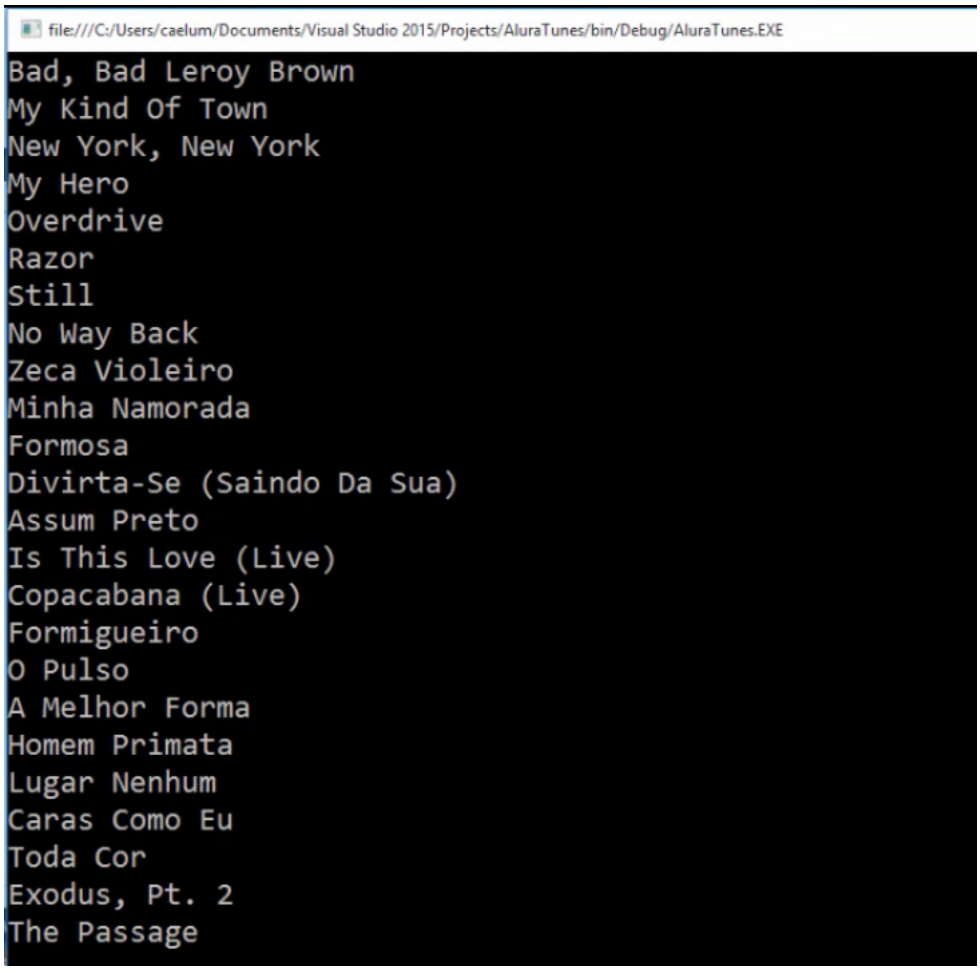
```
using (var contexto = new AluraTunesEntities())
{
    var vendasPorCliente =
    from v in contexto.ps_Vendas_Por_Cliente(clienteId)
    select v;
}
```

Agora, é preciso acrescentar um `foreach(var item in vendasPorCliente)` e também o `Console.WriteLine(item.Nome)` . O código fica da seguinte maneira:

```
using (var contexto = new AluraTunesEntities())
{
    var vendasPorCliente =
    from v in contexto.ps_Vendas_Por_Cliente(clienteId)
    select v;

    foreach (var item in vendasPorCliente)
    {
        Console.WriteLine(item.Nome);
    }
}
```

Desta forma, vamos rodar a aplicação para verificar o resultado:



O resultado mostra todas as faixas de música adquiridas pelo consumidor de id `17` . Falta acrescentar o `Console.ReadKey()` !

O próximo passo é agrupar os valores por ano e mês. Para fazer isso nós utilizaremos o `group by` que ajuda a aglutinar os elementos, assim, adicionaremos `group v by v.DataNotaFiscal.Year`. Agora, estabelecemos que o ajuntamento de itens deve ser feito por ano e por mês (`Month`). O problema é que o `select v` retornará um erro, pois, não é possível acessar o `v`, assim, é preciso agrupar outra variável. Portanto, vamos alterar para `group v by new` e abaixo disso, teremos o `into agrupado`. O problema é que o `v` não está mais disponível, é preciso seguir utilizando o `agrupado`. Portanto, vamos trazer no `select` algumas propriedades - e não mais as que vêm diretamente da `procedure`. Vamos adicionar abaixo do `into` o `orderby agrupado.Key.Year` e o `agrupado.Key.Month`. O `Key` equivale a chave que irá acessar o agrupamento. Falta trazer no `select` um objeto de anônimo. Teremos o seguinte:

```
using (var contexto = new AluraTunesEntities())
{
    var vendasPorCliente =
    from v in contexto.ps_Vendas_Por_Cliente(clienteId)
    group v by new { v.DataNotaFiscal.Year, v.DataNotaFiscal.Month }
        into agrupado
    orderby agrupado.Key.Year, agrupado.Key.Month
    select new
    {

    };

    select v;

    foreach (var item in vendasPorCliente)
    {
        Console.WriteLine(item.Nome);
    }
}
```

Agora, vamos acrescentar as propriedades que desejamos obter, no caso, ano e mês. Vamos adicionar `Ano = agrupado.Key.Year` e `Mes = agrupado.Key.Month`. Assim, é preciso passar a propriedade `Total = agrupado.Sum()`, referente ao total, e a função da soma para a qual devemos inscrever os campos que devem ser somados. Faremos isso por meio da expressão lambda: `a => a.Total`.

Teremos o seguinte:

```
orderby agrupado.Key.Year, agrupado.Key.Month
select new
{
    Ano = agrupado.Key.Year,
    Mes = agrupado.Key.Month,
    Total = agrupado.Sum(a => a .Total)
};
```

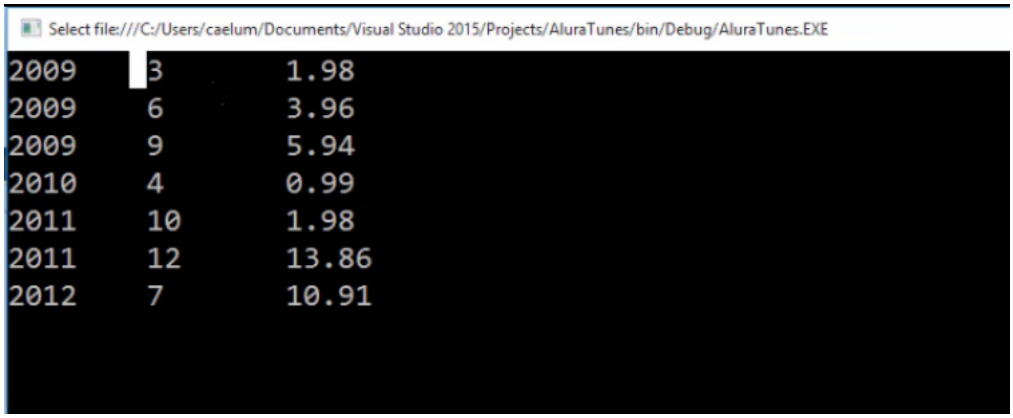
Feito isso, o `Console.WriteLine` afirma que o `item.Nome` é inválido, pois não temos mais o nome da faixa da música no presente relatório. O que existem são informações referentes a `Ano`, `Mes` e `Total`. Portanto, vamos imprimir essas três informações passando para o `Console.WriteLine()` o `item.Ano`, `item.Mes` e `item.Total`. Adicionamos também uma `string` de formatação: `"{0}\t{1}\t{2}"`:

```
orderby agrupado.Key.Year, agrupado.Key.Month
select new
{
```

```
    {
        Ano = agrupado.Key.Year,
        Mes = agrupado.Key.Month,
        Total = agrupado.Sum(a => a .Total)
    };

    foreach (var item in vendasPorCliente)
    {
        Console.WriteLine("{0}\t{1}\t{2}", item.Ano, item.Mes, item.Total);
    }
```

Agora, podemos rodar a aplicação para verificar o resultado:



O resultado mostra uma lista que ordena na primeira coluna as vendas por mês, na segunda por ano e na terceira por total de vendas.

Nesta aula aprendemos a utilizar uma `stored procedure` como fonte de dados para uma consulta `LINQ` do `Entities` !