A Lixeira do Dri

Se eu não apertei Shift + Del...

DELETADO EM 24 DE NOVEMBRO DE 2009

Gerar XMLs pelo SQL Server (FOR XML)

Depois de 3 meses de muita quebradeira em casa, agora tenho meu espaço de volta e talvez consiga postar mais facilmente hahahahahaha.

Hoje vou falar, mais uma vez, sobre XML... É não tem como escapar.

Vamos ao seguinte cenário, temos 3 tabelas em nosso banco de dados, são elas: Conta, Usuario, Endereço. Meu cliente tem um sistema interno que recebe os novos usuários de seu site, ele quer receber diariamente todos os usuário do site em formato XML. Porém não só os usuários novos, ele quer todos os dias TODOS os usuário (É isso existe).

Meu cliente (Bão de venda), tem pré-cadastrados 100.000 usuários, ele pretende ter cerca de 500 novos por dia =O. Se minha calculadora estiver certa 182.500 por ano, pouco até, mas já precisa-se pensar sobre a performance.

O XML deve ter o seguinte formato:

```
<Usuario>
        <Nome>Teste</Nome>
      ... *
</Usuario>
```

* São os campos: Nome, Sobrenome, Login, E-mail, Sexo, DataCadastro, Tipo, CPF, CNPJ, Razão Social, RG, DataNascimento, Logadouro, Numero, Complemento, Referencia, Bairro, Cidade, Estado, Pais, Telefone1, Telefone2, Celular, Fax... E assim vai porque isso é só para exemplificar que é "campo pra caramba"!!!

Então, podemos fazer de 2 modos:

O 1º é fazer uma view e após uma procedure que retorna todos esses carinhas para o meu objeto (1 objeto? Sei lá acho que vão ser 3 objetos ^^, ou um DataSet talvez, não importa) no C#, lá eu varro a lista e vou escrevendo o XML de uma das N maneiras que o C# tem de escrever um XML. Pronto...

Ou podemos deixar 60% mais rápido.

Podemos até criar a view, porém é a procedure que vai retornar para o C# um XML formatado, o C# vai jogar em um arquivo e vai salvar. Joiá! Mas como?

Para exemplificar vamos trabalhar com Nome, Sobrenome e E-mail. Assim pulamos a parte da View e vamos direto para a procedure:

```
CREATE PROCEDURE GerarXMLUsuarios

AS

BEGIN

        SELECT [UsuNome]

              ,[UsuSobrenome]

              ,[UsuEmail]

          FROM [tbUsuarios]

END
```

Agora utilizaremos a cláusula FOR XML.

O FOR XML simplesmente organiza e retorna as informações em formato XML.

Temos as seguintes opções no FOR XML:

1 - FOR XML AUTO:

O modo AUTO retorna os resultados da consulta como elementos XML simples e aninhados, ou seja, cada tabela indicada depois da cláusula FROM é um elemento do XML, as colunas indicadas

Sobre o dono da lix

Adriano Galesso Alves

Sobre: 23 anos, técnico em informática e tecnólogo em informática para gestão de negócios, trabalhando com a plataforma .NET e Visual C# a 3 anos. Atualmente na área desenvolvimento de plataformas de e-Commerce.

no SELECT são atributos. Utilizamos o AS para nomear os elementos:

```
SELECT [UsuNome] AS Nome
        ,[UsuSobrenome] AS Sobrenome
        ,[UsuEmail] AS Email
  FROM [tbUsuarios] AS UsuariosGerais
FOR XML AUTO
```

Resultado:

```
<UsuariosGerais Nome="Adriano" Sobrenome="Teste"
Email="adriano.galesso@teste.com.br" />

<UsuariosGerais Nome="renata" Sobrenome="Teste"
Email="renata@teste.com.br" />

<UsuariosGerais Nome="Christiano" Sobrenome="Teste"
Email="web@teste.com.br" />

<UsuariosGerais Nome="suporte" Sobrenome="Teste"
Email="suporte@teste.com.br" />
```

Se alguma coluna do SELECT for de uma tabela já identificada ela é colocada como atributo (conforme visto acima), se for de uma tabela diferente será adicionada como elemento dentro do elemento pai, que é a 1° tabela especificada:

```
SELECT UsuariosGerais.[UsuNome] AS Nome
        ,UsuariosGeraisIntro.[UsuSobrenome] AS Sobrenome
        ,UsuariosGerais.[UsuEmail] AS Email
  FROM [tbUsuarios] AS UsuariosGerais INNER JOIN
                [tbUsuarios] AS UsuariosGeraisIntro
                        ON UsuariosGerais.UsuCodigo =
UsuariosGeraisIntro.UsuCodigo
FOR XML AUTO
```

Resultado:

```
<UsuariosGerais Nome="Adriano" Email="adriano.galesso@teste.com.br">
  <UsuariosGeraisIntro Sobrenome="Teste" />
</UsuariosGerais>
<UsuariosGerais Nome="renata" Email="renata@teste.com.br">
  <UsuariosGeraisIntro Sobrenome=" Teste " />
</UsuariosGerais>
<UsuariosGerais Nome="Christiano" Email="web@teste.com.br">
  <UsuariosGeraisIntro Sobrenome=" Teste " />
</UsuariosGerais>
<UsuariosGerais Nome="suporte" Email="suporte@teste.com.br">
  <UsuariosGeraisIntro Sobrenome=" Teste " />
</UsuariosGerais>
```

Vejam que, como estou pegando o Sobrenome de uma “tabela diferente”, ele criou uma nova Tag dentro da principal e aplicou o atributo Sobrenome a ela.

#CSharp BR
Ikeda

Alguns Comentários

Anonymous wrote...
Olá, gostaria de saber como fazer isso usando EF, tenho uma procedure que gera esse xml no sql server, só que eu queria pegar esse xml em C#. como fazer isso?

Jonathan wrote...
Muito legal, estava com exatamente este problema! Valew

Manuela Bognar wrote...
Nooooossssaaaaaaaaa, você voltou!Só porque tava escrevendo artigos pra revista, esqueceu da Lixeira? hahahaOlha, vê se não para mais de escrever aqui ;*

Assim funciona o FOR XML AUTO.

2 - FOR XML RAW:

No modo RAW cada resultado da instrução SELECT vem em linha de acordo com o nome fornecido ao RAW (que se não for fornecido é raw mesmo):

```
SELECT [UsuNome] AS Nome
        ,[UsuSobrenome] AS Sobrenome
        ,[UsuEmail] AS Email
  FROM [tbUsuarios] AS UsuariosGerais
FOR XML RAW
```

```
SELECT [UsuNome] AS Nome
        ,[UsuSobrenome] AS Sobrenome
        ,[UsuEmail] AS Email
  FROM [tbUsuarios] AS UsuariosGerais
FOR XML RAW('Teste')
```

Resultado:

```
<row Nome="Adriano" Sobrenome="Teste"
Email="adriano.galesso@teste.com.br" />
<row Nome="renata" Sobrenome="Teste" Email="renata@teste.com.br" />
<row Nome="Christiano" Sobrenome="Teste" Email="web@teste.com.br" />
<row Nome="suporte" Sobrenome="Teste" Email="suporte@teste.com.br" />
```

```
<Teste Nome="Adriano" Sobrenome="Teste"
Email="adriano.galesso@teste.com.br" />
<Teste Nome="renata" Sobrenome="Teste" Email="renata@teste.com.br" />
<Teste Nome="Christiano" Sobrenome="Teste" Email="web@teste.com.br" />
<Teste Nome="suporte" Sobrenome="Teste" Email="suporte@teste.com.br" />
```

Nota-se que diferente do AUTO, podemos nomear nossas elementos de acordo com o que colocamos no RAW e tabelas diferentes não são criadas como novos elementos filhos.

Assim funciona o FOR XML RAW

3 - FOR XML PATH:

O modo PATH tem uma maneira simples de misturar nós de elementos e atributos, e para propriedades mais complexas o PATH pode ser utilizado para trazer resultados com mais facilidade. Os nomes ou alias de colunas são tratados como expressões XPath.

Vamos ao teste com nome de colunas:

```
SELECT [UsuNome] AS '@Nome'
          ,[UsuSobrenome] AS Sobrenome
          ,[UsuEmail] AS Email
  FROM [tbUsuarios] AS Usuarios
FOR XML PATH
```

Resultado:

```
<row Nome="Adriano">
```

```
    <Sobrenome>Teste</Sobrenome>
    <Email>adriano.galesso@teste.com.br</Email>
  </row>
   <row Nome="renata">
    <Sobrenome>barbosa</Sobrenome>
    <Email>renata@teste.com.br</Email>
  </row>
  <row Nome="Christiano">
    <Sobrenome>Teste</Sobrenome>
    <Email>web@teste.com.br</Email>
  </row>
  <row Nome="suporte">
    <Sobrenome>Teste</Sobrenome>
    <Email>suporte@teste.com.br</Email>
  </row>
```

Nota-se que onde colocamos o @ temos um Atributo, os que estão sem o @ são Elementos.

Outro teste, pra mim o mais legal, é o seguinte:

```
      SELECT [UsuCodigo] AS '@ID'
                ,[UsuNome] AS 'Nome/Nome'
                ,[UsuSobrenome] AS 'Nome/Sobrenome'
                ,[UsuEmail] AS 'Email'
        FROM [tbUsuarios] AS Usuarios
      FOR XML PATH('Usuarios')
```

Resultado:

```
<Usuarios ID="216562">
  <Nome>
    <Nome>Adriano</Nome>
    <Sobrenome>Teste</Sobrenome>
  </Nome>
  <Email>adriano.galesso@teste.com.br</Email>
</Usuarios>
<Usuarios ID="216562">
  <Nome>
    <Nome>renata</Nome>
    <Sobrenome>Teste</Sobrenome>
  </Nome>
  <Email>renata@teste.com.br</Email>
</Usuarios>
<Usuarios ID="216563">
  <Nome>
    <Nome>Christiano</Nome>
```

```
    <Sobrenome>Teste</Sobrenome>
  </Nome>
  <Email>web@teste.com.br</Email>
</Usuarios>
<Usuarios ID="216566">
  <Nome>
    <Nome>suporte</Nome>
    <Sobrenome>Teste</Sobrenome>
  </Nome>
  <Email>suporte@teste.com.br</Email>
</Usuarios>
```

Nota-se que quando colocado "/" entre os nomes dos elementos, o elemento anterior a "/" é o pai e o posterior é o elemento filho.

Outro ponto é o @, ele só pode ser usado se for o 1° elemento, ou se o elemento anterior for @ também. No caso se colocássemos o @ no Email daria um erro.

Por fim, o PATH pode ser nomeado assim como o RAW.

Assim funciona o FOR XML PATH.

4 - FOR XML EXPLICIT:

O mais avançado das 4 opções, o EXPLICIT especifica a forma que terá a arvore XML, assim você consegue manipular e ter mais controle em qualquer nó que será criado, no nosso caso por exemplo, o Sobrenome pode ser um nó filho de Nome sem precisarmos manipular tabelas conforme fiz em exemplos anteriores.

Porém o EXPLICIT tem algumas obrigações:

- "A primeira coluna deve fornecer o número da marca, o tipo de inteiro, do elemento atual, e o nome da coluna deve ser TAG."

- "A segunda coluna deve fornecer um número de marca do elemento pai e esse nome de coluna deve ser PARENT. Assim criando uma hierarquia."

Também precisamos especificar os nomes das colunas em um formato geral:

Nome do Elemento ! Numero da TAG ! Nome do Atributo ! Diretiva

Onde:

Nome do Elemento: Identificador da Tag, exemplo Usuario = <Usuario>

Numero da TAG: Assim como Tag e o Parent, ajuda a determinar a aninhamento do XML.

Nome do Atributo: Fornece o nome do atributo a ser construído no Nome do Elemento especificado, se existir a diretiva, ai esse se torna um elemento filho.

Diretiva: É opcional e você pode usá-la para fornecer informações adicionais para construção do XML. Com o propósito de codificar valores e mapear os dados de cadeia de caracteres para XML.

Não vou me aprofundar muito no EXPLICIT, pois o mesmo tem muitas informações. Uma boa leitura está aqui para quem quiser ou precisar se aprofundar no assunto. Entretanto vamos ver alguns exemplos:

```
      SELECT 1 AS TAG
                ,NULL AS PARENT
                ,[UsuNome] as [Usuario!1!Nome!Element]
                ,[UsuSobrenome] as [Usuario!1!Sobrenome!Element]
                ,[UsuEmail] as [Usuario!1!Email!Element]
        FROM [tbUsuarios]
      FOR XML EXPLICIT
```

Resultado:

```
<Usuario>
  <Nome>Adriano</Nome>
  <Sobrenome>Teste</Sobrenome>
  <Email>adriano.galesso@teste.com.br</Email>
</ Usuario >
< Usuario >
  <Nome>renata</Nome>
  <Sobrenome>Teste</Sobrenome>
  <Email>renata@teste.com.br</Email>
</ Usuario >
< Usuario >
  <Nome>Christiano</Nome>
  <Sobrenome>Teste</Sobrenome>
  <Email>web@teste.com.br</Email>
</ Usuario >
< Usuario >
  <Nome>suporte</Nome>
  <Sobrenome>Teste</Sobrenome>
  <Email>suporte@teste.com.br</Email>
</Usuario >
```

Nota-se que obriguei todas as TAG serem 1 e todas as PARENT serem NULL, e Order é o nome do Elemento, 1 é o número da Tag (que só temos 1 mesmo), depois temos os nomes dos atributos e por fim digo que esses atributos são Elementos (se não tivesse especificado isso, eles viriam em linha como se fosse o RAW).

Assim até que é fácil, mas e se quisermos fazer o que eu disse acima? Colocar o Sobrenome como elemento filho de Nome?

Temos uma tabela (retirada do site do MSDN) ela mostra como trabalha o TAG e o PARENT.

Tag	Parent	Customer!1!cid	Customer!1!name	Order!2!id	Order!2!date	OrderDetail!3!id!id	OrderDetail!3!pid!idref
1	NULL	C1	"Janine"	NULL	NULL	NULL	NULL
2	1	C1	NULL	01	1/20/1996	NULL	NULL
3	2	C1	NULL	01	NULL	OD1	P1
3	2	C1	NULL	01	NULL	OD2	P2
2	1	C1	NULL	02	3/29/1997	NULL	NULL

Vendo assim parece simples, mas ficar manipulando essas duas colunas é bem complicado.

Então podemos usar o Campo chave para organizar meu XML:

```
      SELECT 1 AS TAG
                ,NULL AS PARENT
                ,[UsuCodigo] * 100 AS [Usuario!1!UsuCodigo]
                ,NULL as [Nome!2!Nome!Element]
                ,NULL as [Nome!2!Sobrenome!Element]
                ,[UsuEmail] as [Usuario!1!Email!Element]
        FROM [tbUsuarios]
      UNION
      SELECT 2 AS TAG
                ,1 AS PARENT
```

```
                ,[UsuCodigo] * 100 + 1
                ,[UsuNome]
                ,[UsuSobrenome]
                ,[UsuEmail]
          FROM [tbUsuarios]
       ORDER BY [Usuario!1!UsuCodigo]
       FOR XML EXPLICIT
```

Resultado:

```
<Usuario UsuCodigo="21656000">
  <Email>renata @teste.com.br</Email>
  <Nome>
    <Nome>renata</Nome>
    <Sobrenome>Teste</Sobrenome>
  </Nome>
</Usuario>
<Usuario UsuCodigo="21656200">
  <Email>renata@teste.com.br</Email>
  <Nome>
    <Nome>renata</Nome>
    <Sobrenome>Teste</Sobrenome>
  </Nome>
</Usuario>
<Usuario UsuCodigo="21656300">
  <Email>web@teste.com.br</Email>
  <Nome>
    <Nome>Christiano</Nome>
    <Sobrenome>Teste</Sobrenome>
  </Nome>
</Usuario>
```

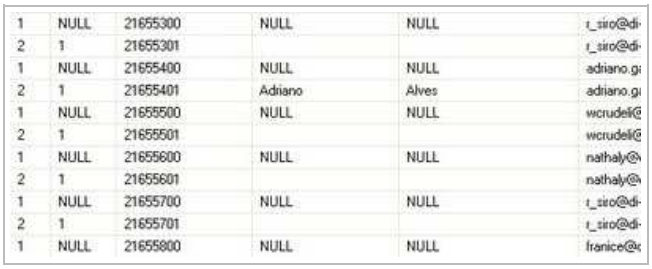
Agora nota-se muita coisa =)

Bom vemos que apareceu o UsuCodigo, ele vai ajudar a ordenação dos resultados, e mostro o porquê:

Sem ele os resultados viriam ordenados do 1º SELECT e depois os do 2º SELECT:

1	NULL	NULL	NULL
1	NULL	NULL	NULL
1	NULL	NULL	NULL
1	NULL	NULL	NULL
2	1		
2	1		
2	1		
2	1		
2	1		
2	1		
2	1		

Com ele temos como organizar



1	NULL	21655300	NULL	NULL	r_siro@di
2	1	21655301			r_siro@di
1	NULL	21655400	NULL	NULL	adriano.g
2	1	21655401	Adriano	Alves	adriano.g
1	NULL	21655500	NULL	NULL	wcrudeli@
2	1	21655501			wcrudeli@
1	NULL	21655600	NULL	NULL	nathaly@
2	1	21655601			nathaly@
1	NULL	21655700	NULL	NULL	r_siro@di
2	1	21655701			r_siro@di
1	NULL	21655800	NULL	NULL	franice@

Por isso fazemos a utilização da coluna chave e nela fazemos uma determinada conta para os números próximos não se repetirem.

O outro SELECT está para trazer os resultados da TAG 2 e PARENT 1, porque queremos que o Nome seja um nó filho dentro do nó principal. Se precisássemos de outro nó, novamente teríamos que usar o SELECT.

Não termino o FOR XML EXPLICIT dizendo que é assim que funciona pois tem N maneiras e N atribuições que podemos fornecer ao mesmo. Mas garanto uma boa leitura no site da MSDN.

5 - OUTROS ARGUMENTOS DO FOR XML (AUTO, RAW e PATH apenas)

Podemos aplicar a diretiva ELEMENTS após o FOR XML, assim o XML será todo ajustado a base de elementos (os valores não serão mais atributos):

```
       SELECT UsuariosGerais.[UsuNome] AS Nome
              ,UsuariosGeraisIntro.[UsuSobrenome] AS Sobrenome
              ,UsuariosGerais.[UsuEmail] AS Email
         FROM [tbUsuarios] AS UsuariosGerais INNER JOIN
                     [tbUsuarios] AS UsuariosGeraisIntro
                             ON UsuariosGerais.UsuCodigo =
UsuariosGeraisIntro.UsuCodigo
       FOR XML AUTO, ELEMENTS
```

Resultado:

```
<UsuariosGerais>
  <Nome>Adriano</Nome>
  <Email>adriano.galesso@teste.com.br</Email>
  <UsuariosGeraisIntro>
    <Sobrenome>Teste</Sobrenome>
  </UsuariosGeraisIntro>
</UsuariosGerais>
<UsuariosGerais>
  <Nome>renata</Nome>
  <Email>renata@teste.com.br</Email>
  <UsuariosGeraisIntro>
    <Sobrenome>barbosa</Sobrenome>
  </UsuariosGeraisIntro>
</UsuariosGerais>
<UsuariosGerais>
  <Nome>Christiano</Nome>
  <Email>web@teste.com.br</Email>
  <UsuariosGeraisIntro>
    <Sobrenome>Teste</Sobrenome>
```

```
  </UsuariosGeraisIntro>
</UsuariosGerais>
<UsuariosGerais>
  <Nome>suporte</Nome>
  <Email>suporte@teste.com.br</Email>
  <UsuariosGeraisIntro>
    <Sobrenome>Teste</Sobrenome>
  </UsuariosGeraisIntro>
</UsuariosGerais>
```

Nota-se que está tudo como Tag e seus valores não estão mais como atributos, também nota-se o novo nó que se criou com a tabela “diferente”.

Outra diretiva que podemos aplicar é o ROOT. Com ela atribuímos um elemento pai ao elemento geral:

```
      SELECT [UsuCodigo] AS '@ID'
                ,[UsuNome] AS 'Nome/Nome'
                ,[UsuSobrenome] AS 'Nome/Sobrenome'
                ,[UsuEmail] AS 'Email'
        FROM [tbUsuarios] AS Usuarios
      FOR XML PATH('Usuarios'), ROOT('Teste')
```

Resultado:

```
 <Usuarios ID="216562">
  <Nome>
    <Nome>renata</Nome>
    <Sobrenome>Teste</Sobrenome>
  </Nome>
  <Email>renata@teste.com.br</Email>
</Usuarios>
<Usuarios ID="216562">
  <Nome>
    <Nome>renata</Nome>
    <Sobrenome>Teste</Sobrenome>
  </Nome>
  <Email>renata@teste.com.br</Email>
</Usuarios>
<Usuarios ID="216563">
  <Nome>
    <Nome>Christiano</Nome>
    <Sobrenome>Teste</Sobrenome>
  </Nome>
  <Email>web@teste.com.br</Email>
</Usuarios>
```

```
  <Usuarios ID="216566">
    <Nome>
      <Nome>suporte</Nome>
      <Sobrenome>Teste</Sobrenome>
    </Nome>
    <Email>suporte@teste.com.br</Email>
  </Usuarios>
```

Temos outras diretivas como o TYPE, ELEMENTXSINIL, BINARY BASE64, XMLSCHEMA... Mas essas são mais avançadas, fogem do nosso artigo principal.

Bom, agora sabemos como gerar XMLs pelo SQL e formata-los da maneira que precisamos.

Mas para onde vai esse resultado?

Você pode gravar em uma coluna de uma tabela qualquer do tipo XML no SQL, ou pode mandar pro C# fazer alguma coisa. E é isso que vamos fazer, já que é este o cenário.

Vamos imaginar o projeto criado, e lá na nossa classe de dados temos os NAMESPACES:

```
using System.Xml;
using System.Xml.XPath;
using System.Data.SqlClient;
```

Vamos ter os atributos:

```
#region "  Atributos  "

private SqlConnection oConn;
private SqlCommand oCmd;
private XmlReader xmlReader;
private XPathDocument xpathDoc;

#endregion
```

Um construtor:

```
#region "  Construtor  "

public MinhaClasse()
{
      oConn = new
SqlConnection(ConfigurationManager.AppSettings["MinhaConn"]);
oConn.Open();
       oCmd = oConn.CreateCommand();
}

#endregion
```

E por fim, o meu método que retorna um XPathDocument:

```
#region "  Métodos  "


protected internal XPathDocument SalvarXML()

{

      oCmd.CommandType = CommandType.StoredProcedure;

      oCmd.CommandText = "GerarXMLUsuarios”;


      try

      {

         xmlReader = oCmd.ExecuteXmlReader();

               xpathDoc = new XPathDocument(xmlReader,
XmlSpace.Preserve);

       }

       catch (Exception ex)

       {

               throw new Exception(ex.ToString());

       }

       finally

       {

               oConn.Close();

       }


       return xpathDoc;

}


#endregion
```

Nada de novo até esse método. Chamamos a procedure e damos um ExecuteXMLReader.

O XmlReader fornece uma acesso somente leitura e somente para frente de um fluxo de dados Xml

O XPathDocument fornece representação em memória de um documento, também somente leitura e somente para frente de um Xml.

Sabendo isso, temos um acesso ao XML gerado pela base, e uma representação em memória deste XML, preservando espaços.

Agora na nossa classe de negócios:

Temos os NAMESPACES:

```
using System.Xml.XPath;

using System.Xml;
```

Temos atributos:

```
private string diretorio =
ConfigurationManager.AppSettings["Diretorio"];
private string URL = ConfigurationManager.AppSettings["URL"];
```

E temos um método que gera e salva um XML:

```
public string GerarXML()

{

        string arquivo = "teste.xml";

        string dirArq = diretorio + arquivo;


        TesteDAO oTesteDao = new TesteDAO ();

        XPathDocument oXPdoc = oTesteDao.ListarXML(tipoXML);

        XPathNavigator oXPNav = oXPdoc.CreateNavigator();


        XmlTextWriter Xwriter = new XmlTextWriter(dirArq,
System.Text.Encoding.UTF8);

        Xwriter.Formatting = Formatting.Indented;

        Xwriter.Indentation = 3;

        Xwriter.WriteStartDocument();

        Xwriter.WriteNode(oXPNav, true);

        Xwriter.WriteEndDocument();

        Xwriter.Close();


        return URL + arquivo;

}
```

Nada de novo até esse método. Criamos um XPathDocument, que recebe o retorno do meu método na classe de acesso a dados vista acima.

Criamos um XPathNavigator que é uma classe abstrata, onde temos um modelo de cursor para navegar (como o próprio nome diz) em nosso XML.

Como não temos nada para editar no XML então vamos salvar o mesmo com o XmlTextWriter, onde passamos o caminho que queremos salvar e a codificação (no construtor). Depois podemos fornecer o tipo da formatação, indentação, começo e documento XML, nós (que nesse caso é o nosso XML inteiro), o fim do documento XML e fechamos o arquivo. Vamos ser bonzinhos e vamos retornar o caminho do XML em formato Web.

Pronto!

Fiz este artigo baseado em um problema real, o problema era bem parecido com o colocado aqui, mas não era 1 XML e sim 4, claro, não tem tantos usuários quanto o que eu falei, mas é um calculo baseado em possibilidades.

Esses 4 XMLs, atualmente com 25 MB somados, são gerados diariamente.

A 1º tentativa foi usando um framework e escrevendo os XML pelo C#, o servidor travava graças aos inúmeros acessos ao banco que o componente (framework) fazia.

A 2º tentativa foi criar procedures especificas que já traziam o resultado perfeito e no C# jogadas em objetos específicos também, os objetos eram listados e assim os XMLs eram escritos, não tinha estress, mas ainda demorava cerca de 30 minutos para gerar por completo.

A 3º tentativa esta da forma que eu falei neste artigo, uso o FOR XML PATH, trago pro C# e escrevo o arquivo.

Por isso este artigo está aqui. Gero 25 MB de arquivos XML em 28 SEGUNDOS... Um lixo mesmo! hahahahahha

Até a próxima.

Artigo publicado no Linha de Código

Jogado no lixo por Adriano Galesso Alves

Marcadores: ASP.Net, C#, Lixeira, SQL

Reações: Legal (1) Fiquei com dúvidas (0) Então tá bão (0) Mais... (0)

1 comentários:

Anônimo disse...

Olá, gostaria de saber como fazer isso usando EF, tenho uma procedure que gera esse xml no sql server, só que eu queria pegar esse xml em C#. como fazer isso?

26 de novembro de 2010 07:48

Postar um comentário

Jogue sua opinião na lixeira!

Comentar como: Selecionar perfil...

Postar comentário Visualizar

Links para esta postagem

Criar um link

Postagem mais recente Início Postagem mais antiga

Assinar: Postar comentários (Atom)