**How to Save Object Graph in Master-Detail Relationship with One Stored Procedure**

## Introduction

Ho un client Web che invia dati alla mia applicazione ASP.NET in formato JSON. I dati sono un oggetto grafico in relazione master-detail o anche una raccolta di oggetti grafici. I dati vengono deserializzati in oggetti grafici C# con il controller dell'API Web. Voglio salvare quel grafico oggetto C # con ADO.NET e una stored procedure. E voglio farlo senza GUID, XML ed EF!

## Background

Let's suppose that a web client sends the object graphs consisting of three objects:

* GrandRecord
* Record
* ChildRecord

Let it be a collection of GrandRecords where:

* each GrandRecord has a collection of Records
* and each Record has a collection of ChildRecords
* Id value is integer and autogenerated by database
* and while an object is not saved in database, Id has value = 0

Here is the sample of object graph collection (or object graphs):

Id, Name

GrandRecord 1, (A)

Record |-- 2, (A)A

ChildRecord |-- 3, (A)Aa

ChildRecord |-- 0, (A)Ab

Record |-- 0, (A)B

ChildRecord |-- 0, (A)Ba

ChildRecord |-- 0, (A)Bb

GrandRecord 0, (B)

Record |-- 0, (B)A

Or the same in JSON format:

Shrink ▲

grandRecords: [

{

id: 1,

name: "(A)",

records: [

{

id: 2,

name: "(A)A",

childRecords: [

{

id: 3,

name: "(A)Aa",

},

{

id: 0,

name: "(A)b",

},

]

},

{

id: 0,

name: "(A)B",

childRecords: [

{

id: 0,

name: "(A)Ba",

},

{

id: 0,

name: "(A)Bb",

},

]

}

]

},

{

id: 0,

name: "(B)",

records: [

{

id: 0,

name: "(B)A",

childRecords: []

}

]

}

]

On the web server in ASP.NET controller, the above JSON string is deserialized into object graphs of three classes:

C#

public class GrandRecord

{

public Int32 Id { get; set; }

public String Name { get; set; }

public IList<Record> Records { get; set; }

}

public class Record

{

public Int32 Id { get; set; }

public Int32 GrandRecordId { get; set; }

public String Name { get; set; }

public IList<ChildRecord> ChildRecords { get; set; }

}

public class ChildRecord

{

public Int32 Id { get; set; }

public Int32 RecordId { get; set; }

public String Name { get; set; }

}

Now the object graphs has to be saved with one stored procedure into three database tables:

SQL

create table dbo.GrandRecords

(

Id int not null identity primary key clustered,

Name varchar(30) not null

);

create table dbo.Records

(

Id int not null identity primary key clustered,

GrandRecordId int not null ,

Name varchar(30) not null ,

foreign key (GrandRecordId) references dbo.GrandRecords (Id) on delete cascade

);

create table dbo.ChildRecords

(

Id int not null identity primary key clustered,

RecordId int not null ,

Name varchar(30) not null ,

foreign key (RecordId) references dbo.Records (Id) on delete cascade

);

The question is **how**?

## Solution

Naturalmente, i parametri con valori di tabella per la procedura memorizzata fanno parte della risposta! Avere questi tipi di tabella definiti dall'utente:

create type dbo.GrandRecordTableType as table

(

Id int not null primary key clustered,

Name varchar(30) not null

);

create type dbo.RecordTableType as table

(

Id int not null primary key clustered,

GrandRecordId int not null ,

Name varchar(30) not null

);

create type dbo.ChildRecordTableType as table

(

Id int not null primary key clustered,

RecordId int not null ,

Name varchar(30) not null

);

the stored procedure for saving the above object graphs starts with:

SQL

create procedure dbo.SaveGrandRecords

@GrandRecords dbo.GrandRecordTableType readonly,

@Records dbo.RecordTableType readonly,

@ChildRecords dbo.ChildRecordTableType readonly

as

Quindi dobbiamo raccogliere tutti i dati per tipo (GrandRecord, Record e ChildRecord), creare DataTable ADO.NET e passarli alla stored procedure. Ma! Poiché le nostre tabelle nel database sono collegate da chiavi esterne GrandRecordId e RecordId, dobbiamo in qualche modo mantenere tale collegamento durante la conversione dell'oggetto grafico nelle DataTable separate. Inoltre, le identità dei nuovi oggetti devono essere uniche! Altrimenti, non possiamo distinguere i Registri di GrandRecord (A) dai Registri di GrandRecord (B). Tuttavia, come ricordiamo, i nuovi oggetti hanno Id = 0!

Per risolvere questo problema, assegniamo identità negative sempre crescenti agli ID oggetto, se sono uguali a 0:

var id = int.MinValue;

foreach (var grandRecord in grandRecords)

{

if (grandRecord.Id == 0)

grandRecord.Id = id++;

foreach (var record in grandRecord.Records)

{

if (record.Id == 0)

record.Id = id++;

record.GrandRecordId = grandRecord.Id;

foreach (var childRecord in record.ChildRecords)

{

if (childRecord.Id == 0)

childRecord.Id = id++;

childRecord.RecordId = record.Id;

}

}

}

Now it's time to populate the Data tables.

For example, here is how to prepare a DataTable with Records data:

C#

var recordTable = new DataTable("RecordTableType");

recordTable.Columns.Add( "Id" , typeof( Int32 ));

recordTable.Columns.Add( "GrandRecordId" , typeof( Int32 ));

recordTable.Columns.Add( "Name" , typeof( String ));

var records = grandRecords.SelectMany(gr => gr.Records);

foreach(var record in records)

{

table.Rows.Add(new object[] {record.Id, record.GrandRecordId, record.Name});

}

So after the DataTables are prepared, the stored procedure will receive the following data in table valued parameters:

#### @GrandRecords

| Id | Name |
| --- | --- |
| 1 | (A) |
| -2147483648 | (B) |

#### @Records

| Id | GrandRecordId | Name |
| --- | --- | --- |
| 2 | 1 | (A)A |
| -2147483647 | 1 | (A)B |
| -2147483646 | -2147483648 | (B)A |

#### @ChildRecords

| Id | RecordId | Name |
| --- | --- | --- |
| 3 | 2 | (A)Aa |
| -2147483645 | 2 | (A)Ab |
| -2147483644 | -2147483647 | (A)Ba |
| -2147483643 | -2147483647 | (A)Bb |

### Object Graph Saving Technique

Per aggiornare i dati esistenti, inserire nuovi ed eliminare i vecchi dati, SQL Server utilizza l'istruzione MERGE. L'istruzione MERGE ha la clausola OUTPUT. L'istruzione OUTPUT in MERGE può raccogliere gli ID appena inseriti e gli ID dalla tabella di origine (parametro). Quindi la tecnica "per salvare tutte e tre le tabelle con chiavi esterne corrette" consiste nel raccogliere le coppie InsertedId - ParamId dalla prima tabella e tradurre quei valori per la seconda. Quindi fai lo stesso per la seconda e la terza tavola.

* Se esiste un record in una tabella, MERGE esegue l'AGGIORNAMENTO, insert.Id e source.Id sono uguali all'ID esistente.
* Se un record non esiste in una tabella il MERGE fa INSERT, insert.Id è uguale al nuovo Id e source.Id è uguale all'identità negativa.
* Se un record non esiste nella tabella di origine (parametro), MERGE esegue DELETE, insert.Id e source.Id sono uguali a NULL, ma deleted.Id ha l'ID del record eliminato. Ecco la stored procedure che salva i nostri oggetti grafici:
* create procedure dbo.SaveGrandRecords
* @GrandRecords dbo.GrandRecordTableType readonly,
* @Records dbo.RecordTableType readonly,
* @ChildRecords dbo.ChildRecordTableType readonly
* as
* begin
* set nocount on;
* declare @GrandRecordIds table ( -- translation table
* InsertedId int primary key,
* ParamId int unique
* );
* declare @RecordIds table ( -- translation table
* InsertedId int primary key,
* ParamId int unique,
* [Action] nvarchar(10)
* );
* -- save GrandRecords
* merge into dbo.GrandRecords as target
* using
* (
* select Id, Name from @GrandRecords
* )
* as source on source.Id = target.Id
* when matched then
* update set
* Name = source.Name
* when not matched by target then
* insert ( Name )
* values ( source.Name )
* output -- collecting translation Ids
* inserted.Id,
* source.Id
* into @GrandRecordIds (
* InsertedId ,
* ParamId );
* -- save Records
* merge into dbo.Records as target
* using
* (
* select
* Id ,
* GrandRecordId = ids.InsertedId, -- Id translation target
* Name
* from
* @Records r
* inner join @GrandRecordIds ids
* on ids.ParamId = r.GrandRecordId -- Id translation source
* )
* as source on source.Id = target.Id
* when matched then
* update set
* GrandRecordId = source.GrandRecordId,
* Name = source.Name
* when not matched by target then
* insert (
* GrandRecordId ,
* Name )
* values (
* source.GrandRecordId ,
* source.Name )
* when not matched by source
* and target.GrandRecordId in (select InsertedId from @GrandRecordIds) then
* delete
* output -- collecting translation Ids
* isnull(inserted.Id, deleted.Id),
* isnull(source.Id, deleted.Id),
* $action
* into @RecordIds (
* InsertedId ,
* ParamId ,
* [Action] );
* delete from @RecordIds where [Action] = 'DELETE';
* -- save ChildRecords
* merge into dbo.ChildRecords as target
* using
* (
* select
* Id ,
* RecordId = ids.InsertedId, -- Id translation target
* Name
* from
* @ChildRecords cr
* inner join @RecordIds ids
* on ids.ParamId = cr.RecordId -- Id translation source
* )
* as source on source.Id = target.Id
* when matched then
* update set
* RecordId = source.RecordId ,
* Name = source.Name
* when not matched by target then
* insert (
* RecordId ,
* Name )
* values (
* source.RecordId ,
* source.Name )
* when not matched by source
* and target.RecordId in (select InsertedId from @RecordIds) then
* delete;
* end;

#### Important Notice

Nell'istruzione MERGE, le tabelle di origine e di destinazione DEVONO avere indici clustered nelle rispettive colonne di join! Questo previene i deadlock e garantisce l'ordine di inserimento. Le colonne di join sono in as source sulla riga source.Id = target.Id dell'istruzione MERGE. Ecco perché i tipi di tabella definiti dall'utente sopra hanno una chiave primaria clustered nelle loro definizioni. Ecco perché le identità negative sono in continuo aumento e iniziano con MinValue. Si noti anche la definizione di chiavi esterne nelle tabelle permanenti. Contengono la clausola on delete cascade, che aiuta a eliminare i record figlio durante l'eliminazione dei record padre nell'istruzione MERGE.