Alex Thissen
is senior docent software development bij Twice IT Training, http://www.twice.nl, email: athissen@twice.nl

Advanced ADO.NET

Updaten van disconnected data in een DataSet

Veel programmeurs hebben al kennis gemaakt met ADO.NET en zijn verrast door het feit dat er alleen nog gewerkt kan worden met disconnected data als de DataSet gebruikt wordt. Door een DataSet te vullen met data met behulp van DataAdapter-objecten is het relatief eenvoudig om de DataSet in een user interface te laten bewerken via databound grids en controls in WinForms of WebForms. De meeste problemen ontstaan juist op het moment van updaten van de data naar de databases. Tijdens het werken met disconnected data kunnen meerdere wijzigingen op de data gedaan worden, die vervolgens batchgewijs uitgevoerd moeten worden op de database. Het kan daarbij voorkomen dat andere gebruikers ook dezelfde data manipuleren, zodat er concurrency-problemen ontstaan. Reden genoeg om eens te kijken hoe de gewijzigde data in een DataSet weer gepersisteerd kan worden in de oorspronkelijke databronnen en hoe conflicten in dat proces opgelost kunnen worden.

De DataSet is niet zomaar een verzameling van DataTables en DataRows. Met name de DataRow heeft allerlei karakteristieken die het werken met disconnected data aanzienlijk vereenvoudigen. DataRows hebben een status- en versionerings- mechanisme om de diverse operaties die op een DataRow worden uitgevoerd te kunnen volgen.

Data row state

Een DataRow heeft een RowState property, die de status van een rij aangeeft, oftewel wat er met de rij gebeurd is. Er is een vijftal verschillende toestanden, die overeenkomen met de waarden van de DataRowState-enumeratie: Unchanged, Modified, Added, Deleted en Detached.

De meeste van deze toestanden spreken voor zich. De Detached-toestand geeft aan dat een rij aan geen enkele DataTable is verbonden. Dit is het geval na het maken van een nieuwe rij met DataTable.NewRow of na het verwijderen van een rij met behulp van de DataRow.Remove methode. U zult zelden direct met Detached DataRows werken.

Data row versions

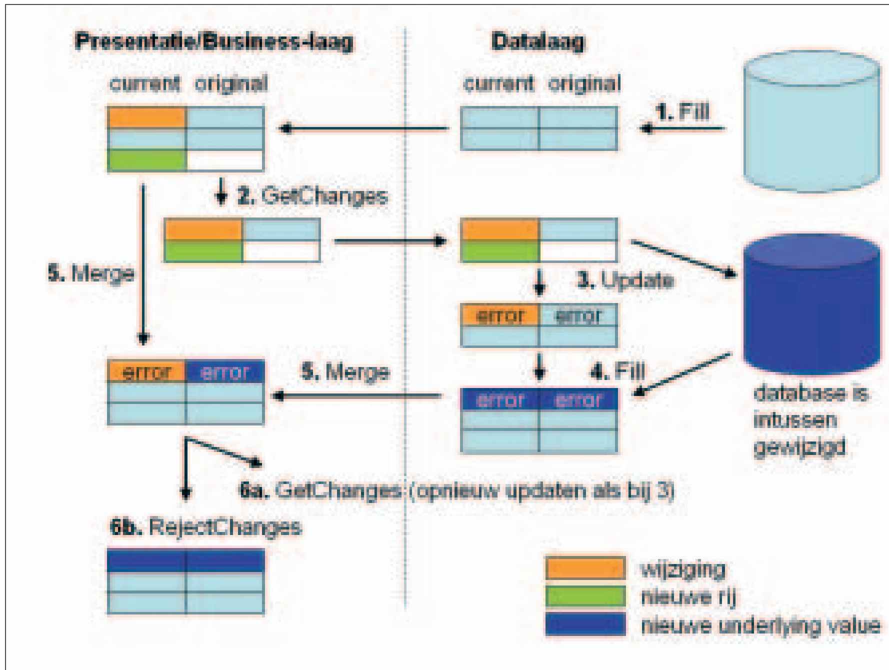
Eén bepaalde DataRow kan uit drie verschillende versies bestaan: current, original en proposed. Ieder van deze versies bevat waarden voor alle kolommen in de DataRow, die verschillend kunnen zijn. Normaliter zullen alleen de Current- en Original-versies beschikbaar zijn. De Proposed versie is alleen voorhanden voor DataRows in edit mode of voor Detached DataRows. De indexer property van een DataRow biedt de mogelijkheid om de kolomwaarde van een bepaalde RowVersion uit te lezen. De syntax is te zien in afbeelding 1.
Er bestaat ook nog een DataRowVersion.Default die automatisch de meest logische of enige juiste rijversie van een DataRow selecteert (zoals bijvoorbeeld de Original versie voor een Deleted rij) en niet de default waarde van een kolom, hetgeen in de ADO.NET documentatie en menigeen boek regelmatig wordt gesuggereerd. Deze enumeratiewaarde is zinvol als de RowState niet à priori bekend is, zoals bij een generiek stuk code. Met behulp van de methode HasVersion kan gecontroleerd worden of een DataRow überhaupt een bepaalde versie bevat, alvorens deze te evalueren.

Persisteren van data

Met de versies en de status van een DataRow bevat een DataSet voldoende informatie om de gemaakte mutaties te laten persisteren in de beoogde databronnen. Het zou echter onverstandig zijn om de complete, deels bewerkte DataSet naar de datalaag terug te sturen. Immers, er kan wor-

```
// Originele waarde ophalen
string name = row["CompanyName", DataRowVersion.Original];
// Waarde tijdens een edit operatie
string CustomerID =  =  = row["CustomerID", DataRowVersion.Proposed];
```

Afbeelding 1. Gebruik van de indexer property



Afbeelding 2. Levenscyclus van disconnected data

```
// Isoleer wijzigingen in nieuwe dataset
DataSet changes = ds.GetChanges();
if (changes == null)
  return;

// Voer update uit naar database
OleDbCommandBuilder builder = new OleDbCommandBuilder(da);
da.ContinueUpdateOnError = true;
dt = changes.Tables["Customers"];
da.Update(dt);
da.Fill(changes, "Customers");
```

Afbeelding 3. Selecteren van wijzigingen en uitvoeren van update

den volstaan met de veranderingen. De methode DataSet.GetChanges maakt een kopie van de DataSet, met daarin alleen die DataRows die een RowState hebben ongelijk aan Unchanged of Detached. Deze meestal kleinere DataSet kan worden teruggestuurd naar de datalaag. Een bijkomend voordeel is dat de oorspronkelijke DataSet onaangetast blijft tijdens de update-actie, waarvan later dankbaar gebruik gemaakt kan worden. Zie ook afbeelding 2. Omdat de GetChanges-methode potentieel een null-resultaat geeft, is het verstandig te toetsen met de HasChanges-methode of er daadwerkelijk wijzigingen zijn. Afbeelding 3 toont een voorbeeld van het gebruik van de GetChanges methode.

Updaten met DataAdapters

In veel gevallen, zoals bij één enkele DataTable en zonder relaties tussen verschillende tabellen, is het voldoende om de Update-methode van de DbDataAdapter (1) te gebruiken om de wijzigingen naar een database weg te schrijven (afbeelding 2, stap 3). Merk op dat de naam van de methode Update enigszins verwarrend is, omdat het hier niet alleen updates, maar ook deletes en inserts betreft{LET OP, hier voetnoot 1: Ofschoon hier DbDataAdapter wordt genoemd, zult u een van DbDataAdapter afgeleide class gebruiken: OleDbDataAdapter of SqlDataAdapter. Hetzelfde geldt voor de Command-objecten: OleDbCommand, SqlCommand, OdbcCommand en Oracle-Command.}. De DbDataAdapter is in essentie een verzameling van vier Command-objecten: een SelectCommand, Insert-, Update en DeleteCommand. Ieder van deze Command-objecten zal voor een bepaalde DataRow de overeenkomstige operatie uitvoeren op de database tijdens de Update-methode. Daarbij wordt voor een DataRow het juiste Command geselecteerd door de RowState te evalueren.

Wanneer de Update-methode gebruikt wordt om de wijzigingen in een DataTable op te slaan, moeten ook alle vier Command-objecten beschikbaar zijn. Deze Command-objecten kunt u zelf instantiëren, configureren en vervolgens toevoegen aan de DbDataAdapter. De Sql- of OleDbCommandBuilder kan deze objecten ook voor u genereren. Deze klasse heeft een constructor die een DbDataAdapter als argument accepteert. Voorwaarde is wel dat de DbDataAdapter al een SelectCommand heeft en dat het SELECT statement minstens een primary key of unique kolom selecteert.

De door de CommandBuilder gegenereerde Update- en DeleteCommand bevatten SQL statements die gebruik maken van optimistic locking. Beiden zullen controleren of de original waarden van de te verwerken DataRow nog overeenkomen met de waarden die op dat moment in de database staan (de underlying values). Indien dat het geval is, wordt de wijziging uitgevoerd, de current value van de DataRow naar de original value gekopieerd en de RowState op Unchanged gezet.

Echter, de verwerking van de wijzigingen wordt gestaakt door het gooien van een DbConcurrencyException indien in één van de kolommen original en underlying waarden niet overeenkomen. Dit gedrag wordt veroorzaakt door de default waarde false van de ContinueUpdateOnError property van de DbDataAdapter. Door deze waarde voor het aanroepen van Update op true te zetten, wordt na een eventuele concurrency error verder gegaan met het updaten van de overige rijen in de DataSet. Eventuele errors tijdens het updaten worden in de RowError property van de betreffende DataRow gekopieerd. Dit betekent dat aan het einde van de Update-operatie de DataSet, die oorspronkelijk alleen wijzigingen

bevatte, nu ten dele over unchanged DataRows beschikt voor alle geslaagde updates. De DataRows waarvoor de updates niet lukten hebben hun RowState (Deleted, Added of Modified) behouden en hebben nog steeds de original en current values van voor de Update-operatie.

Concurrency errors oplossen

Het is nu zaak om de DataRows met concurrency errors te gaan verwerken. Immers, de gebruiker heeft wijzigingen gemaakt, die niet doorgevoerd zijn, omdat iemand intussen in hetzelfde databaserecord ook wijzigingen heeft uitgevoerd. Afhankelijk van de programmering zullen de gebruiker of business rules beslissen hoe dit conflict opgelost moet worden. Om een goede keuze te kunnen maken moet er beschikking zijn over minstens twee sets van waarden: de waarden die op dat moment in de database staan en de wijzigingen van de gebruiker.

De huidige waarden van de database kunnen worden verkregen door nogmaals de Fill-methode van de DbDataAdapter te gebruiken (afbeelding 2, stap 4). Daarmee wordt de wijzigingen-DataSet opnieuw gevuld met alle huidige rijen die in stap 1 ook werden geselecteerd, inclusief nieuwe rijen die intussen zijn toegevoegd. Normaal gesproken worden de geselecteerde records allemaal toegevoegd aan de DataTable. Indien deze een primary key heeft, wordt getracht om overeenkomende keys te vinden en worden bestaande DataRows in de DataSet overschreven. In afbeelding 3 ziet u een codevoorbeeld van deze fase in het updateproces. De Fill-methode zal standaard de original en current values gelijkstellen aan die in de database en de RowState op Unchanged zetten. De RowError properties blijven ongewijzigd bestaan. Met andere woorden, door de DataSet met wijzigingen opnieuw te vullen, zoals in eerste instantie de originele DataSet werd gevuld, bevat deze nu de underlying values oftewel de huidige waarden in de database, met errors voor al die

```
// Voeg originele dataset samen met ververste wijzigingen
ds.Merge(changes, true);
DataTable mergedCustomers = ds.Tables["Customers"];

// Controleer op errors
if (mergedCustomers.HasErrors)
{
  foreach (DataRow row in mergedCustomers.GetErrors())
  {
    Console.WriteLine("Concurrency error bij ID: {0}\n" +
      "Gebruiker: {1}\nDatabase: {2}", row["CustomerID"],
      row["CompanyName", DataRowVersion.Current],
      row["CompanyName", DataRowVersion.Original]);
  }
}
```

Afbeelding 4. Uitvoeren van updates en controle op concurrency errors

DataRows die niet succesvol opgeslagen konden worden.

We beschikken nu over een tweetal DataSets: één met de originele en current values, de andere (ontstaan vanuit de aanroep van GetChanges) met de underlying values en errors. Desgewenst kunnen de twee DataSets samengevoegd worden met de Merge methode (afbeelding 2, stap 5). De DataSet waarop Merge wordt aangeroepen wordt veelal de target DataSet genoemd. Het eerste argument van de methode is de source DataSet, die zo genoemd wordt omdat deze niet verandert. Het tweede argument van de Merge methode, de boolean preserveChanges, geeft aan of de wijzigingen in de target DataSet behouden worden. Als deze waarde true is zullen alleen de original values vanuit de source DataSet overgenomen worden, terwijl dat bij false zowel de current als original waarden zijn. In ons geval is het verstandig om deze waarde op true te zetten, zodat de uiteindelijke DataSet de combinatie van current (door de gebruiker bewerkte waarden) en underlying values (nieuwe database waarden) als original values bevat in die DataRows waarvoor de updates niet lukten. Afbeelding 4 laat het gebruik van de Merge methode zien.

Vervolgens kan met behulp van de property HasErrors van de DataSet, DataTable of DataRow bepaald worden of het betreffende object errors bevat. De rijen die een error bevatten zijn eenvoudig te selecteren met de methode GetErrors van de DataTable. Zie afbeelding 4. Voor ieder van deze rijen kan besloten worden welke acties ondernomen moeten worden. Voor de hand liggende acties zijn het verwerpen van de wijzigingen van de gebruiker of het opnieuw uitvoeren van een update.

Herkansing

Als besloten wordt om de updates van de gebruiker alsnog weg te schrijven, zal er een nieuwe poging ondernomen worden om de updates weg te schrijven naar de database (afbeelding 2, stap 6a). Omdat nu de original values overeenkomen met de underlying values zouden er geen concurrency errors meer moeten optreden, tenzij er intussen weer updates door anderen zijn uitgevoerd. Voor DataRows waarvan de wijzigingen van de gebruiker weggegooid kunnen worden (afbeelding 2, stap 6b) kan de methode RejectChanges worden gebruikt. Deze methode kopieert de original values naar de current values en zet de RowState weer op Unchanged, zodat effectief de nieuwe waarden in de database geaccepteerd worden. De tegenhanger van RejectChanges heet AcceptChanges methode en kopieert de current-waarden naar de original-waarden. Hiermee kunnen de wijzigingen in een DataSet geaccepteerd worden, bijvoorbeeld als er geen errors waren bij het updaten.

	DataRowState				
DataViewRowState	Added	Unchanged	Modified	Deleted	Detached
Added	Current				
CurrentRows	Current	Current	Current		
Deleted				Original	
ModifiedCurrent			Current		
ModifiedOriginal			Original		
OriginalRows		Current	Original	Original	
Unchanged		Current			

Current en Original zijn waarden uit DataRowVersion enumeratie

Tabel 1. Overzicht DataViewRowState in combinatie met RowState en RowVersion

Daarmee kan het opnieuw vullen van de wijzigingen DataSet komen te vervallen, als er geen dringende behoefte is om de originele DataSet te verversen. Tenslotte zult u nog de ClearErrors methode van de DataRow moeten aanroepen om alle errors in de rij te verwijderen. Op dit punt heeft u een DataSet met daarin de gemaakte wijzigingen en opgeloste conflicten. Door het gebruik van de Fill methode in stap 4 zijn bovendien ook alle updates, deletes en inserts van concurrent users voorhanden.

Volgorde van updaten

In complexere update-situaties dan single-table updates kunnen er problemen ontstaan met de volgorde waarin updates door DbDataAdapters worden uitgevoerd. Met name wanneer een 1-op-n relatie bestaat tussen twee tabellen, zoals bijvoorbeeld Orders en OrderDetails. In een relationele database is deze relatie met behulp van een foreign-key constraint geïmplementeerd. De DataSet, die overeenkomstige DataTables heeft met eventueel daartussen een DataRelation, bevat nieuwe, bewerkte en verwijderde orders met bijbehorende details. Een tweetal DbDataAdapters daOrders en daOrderDetails zullen de updates naar de database uitvoeren. U roept bijvoorbeeld eerst daOrders.Update(ds) aan. De nieuwe Order records kunnen worden toegevoegd, maar het verwijderen van orders lukt niet vanwege het bestaan van gerelateerde OrderDetails records. Wanneer de eerste methode-aanroep naar daDetails.Update zou worden gemaakt, verloopt het verwijderen van records correct, maar ontstaan er referentiële integriteitsproblemen bij het toevoegen van details voor nog niet bestaande orders.

In deze of soortgelijke situaties moet meer sturing gegeven worden aan de DbDataAdapters. De inserts en deletes moeten afzonderlijk uitgevoerd worden. Eén van de mogelijkheden is het selecteren van alleen toegevoegde dan wel verwijderde rijen in een nieuwe DataSet. Hiervoor wordt aan de DataSet.GetChanges-methode de gewenste DataRowState meegegeven.

Een andere mogelijkheid om rijen te selecteren wordt geboden door de methode DataTable.Select. Deze methode accepteert naast een filter- en sorteerstring een DataViewRowState-enumeratiewaarde. Als u niet wilt filteren of sorteren, dan geeft u null op voor de filter- en sorteerargumenten in de Select-methode. Met een DataViewRowState-waarde (of OR-combinaties daarvan) kunnen rijen met een specifieke RowState en RowVersion geselecteerd worden. In tabel 1 kunt u zien hoe een waarde van DataViewRowState in combinatie met de DataRowState van de rij bepaalt of een rij geselecteerd wordt en welke versie van de rij dit dan is. Hiermee kunt u zelf sturing aanbrengen in de volgorde van updates. De Select methode verdient in het geval van parent-child updates de voorkeur boven de GetChanges methode van de DataSet. De Select methode retourneert eenvoudigweg een array van DataRows uit één tabel, terwijl de GetChanges methode een complete DataSet oplevert met eventueel ook andere tabellen dan de parent en child tabel.

Vele paden met disconnected data

Lang niet alle aspecten van het werken met disconnected data zijn aan de orde gekomen, zoals het updaten van IDENTITY kolommen. Er zijn ook vele paden te bewandelen in de wereld van ADO.NET en disconnected data. Zeker zult u door het lezen van dit artikel de eerste stappen durven zetten bij het opslaan van de data.

Nuttige internetadressen

- http://msdn.microsoft.com/library/en-us/cpguide/html/cpconaccessingdatawithadonet.asp
- http://msdn.microsoft.com/library/en-us/cpguide/html/cpconoptimisticconcurrency.asp
- http://www.wrox.com/books/186100527x.htm
- http://www.microsoft.com/mspress/books/4825.asp