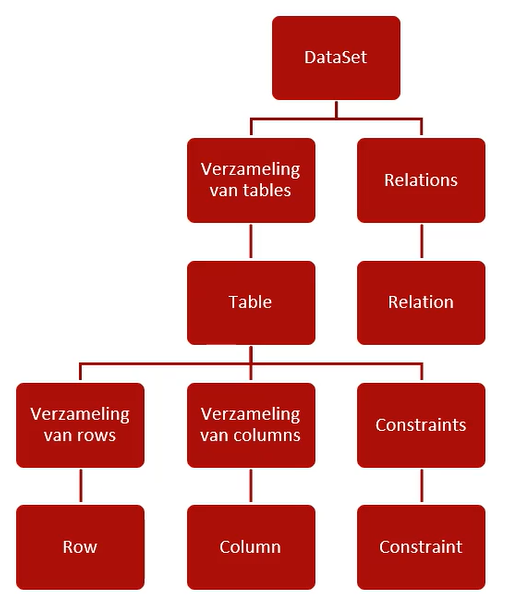
**Deel 10 - ADO.NET Vervolg**

**Relationele in-memory datastore**

= Interne representatie (binnen de app waarin je aan het programmeren bent) hoe de data structuur er uit zou zien in de effectieve database

**DataSet object**

* **Data provider onafhankelijk object**
* Terug te vinden onder de namespace **System.Data**
* Voorziet in een relationele view van gegevens analoog aan een database
* Volledig onafhankelijk van gegevensbron
  + Oracle database, SQL Server database, XML bestand, …
* Wordt bewaard in een in-memory cache
  + **DataReader** = rechtstreekse koppeling met bron
  + **DataSet** = geen rechtstreekse interactie met datasource
* Object model van een DataSet

**DataTable object**

* Terug te vinden onder de namespace **System.Data**
* Conceptueel analoog aan een tabel uit een relationele database
* Standalone of binnen een DataSet
* Collection van DataTables in DataSet via **Tables** property

=> Individuele DataTable aanspreekbaar via index of naam

**DataColumn object**

* Terug te vinden onder de namespace **System.Data**
* Properties instelbaar (bv. DataType, DefaultValue, Name, Unique)
* Toevoegen aan een DataTable
* Collection van DataColumns in DataTable via **Columns** property

=> individuele DataColumn aanspreekbaar via index of naam

**DataRow**

* Terug te vinden onder de namespace **System.Data**
* Conceptueel analoog aan een record (rij) uit een tabel
* Collection van Rows in DataTable via **Rows** property

=> individuele DataRow aanspreekbaar via index

* Kolom van een rij aanspreekbaar via
  + Indexer
  + ColumnName
  + DataColumn
* Nieuwe rij **toevoegen** aan een DataTable

1. Nieuwe rij aanmaken vie NewRow() method van DataTable instance
2. Rij toevoegen aan DataRowCollection van DataTable instance via Add() method

* Bestaande rij **wijzigen** in een DataTable

1. Rij zoeken
2. Waarde(n) aanpassen van kolom(men)

* Bestaande rij **verwijderen** kan op meerdere manieren
  + Delete() method aanroepen van een DataRow instance
    - RowState => Deleted
  + Remove() method aanroepen van de DataTable.Rows property
    - Verandert de RowState niet maar verwijdert de rij uit de DataTable
    - DataRow instance meegeven als parameter
  + RemoveAt() method aanroepen van de DataTable.Rows property
    - Verandert de RowState niet maar verwijdert de rij uit de DataTable
    - Index van de te verwijderen rij meegeven als parameter

**RowState**

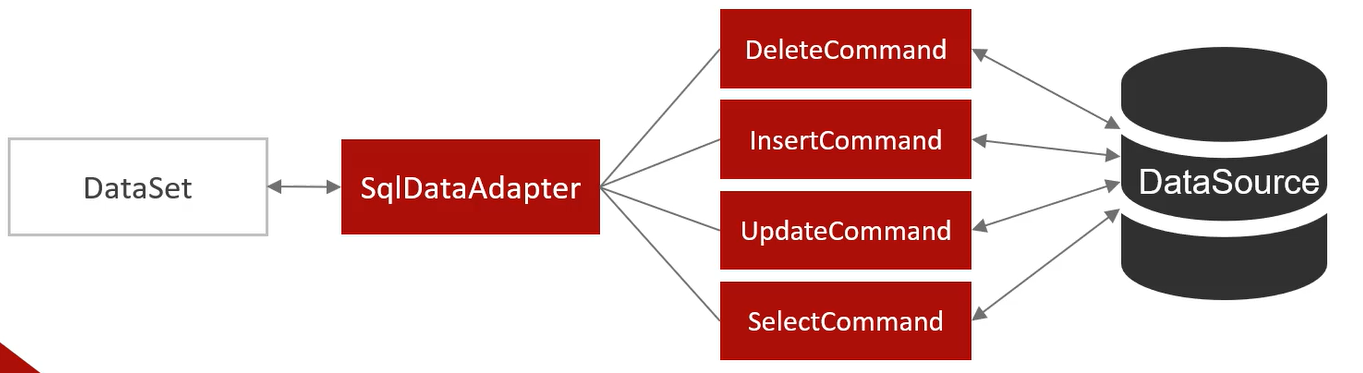
* Wijzigingen gebeuren in eerste instantie enkel binnen de DataSet
* Op een bepaald moment moeten wijzigingen gepushed worden naar de data source
  + Rij verwijderd in dataset? => Delete in data source
  + Rij aangepast in dataset? Update in data source
  + Rij toegevoegd in dataset? Insert in data source
* Elke DataRow heeft een **RowState** die aangeeft wat er met een rij is gebeurd  
  **(Added - Deleted - Modified - Unchanged - Detached)**
* Op basis van deze RowState weten we welke actie uit te voeren in de data source
* Rowstate wijzigen
  + Methode **AcceptChanges()** uitvoeren zet RowState op Unchanged
    - Voor een enkele DataRow indien uitgevoerd op een DataRow
    - Voor alle DataRows in een DataTable indien uitgevoerd op een DataTable
    - Voor alle DataRows in alle DataTables indien uitgevoerd op een DataSet
  + Methode **SetAdded()** zet RowState op Added
  + Methode **SetModified()** zet RowState op Modified

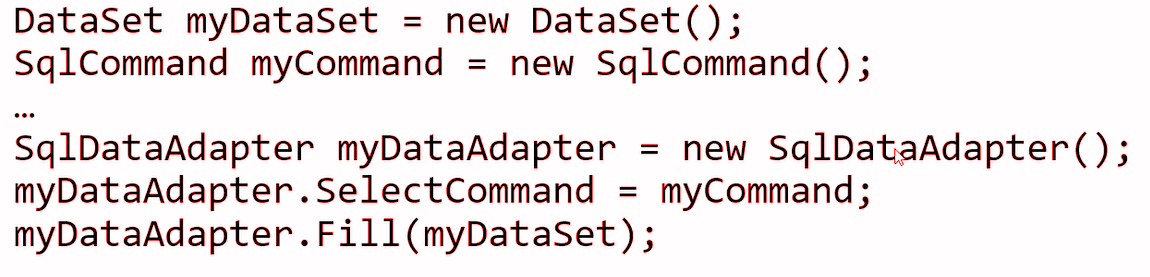
**Primary keys en relaties**

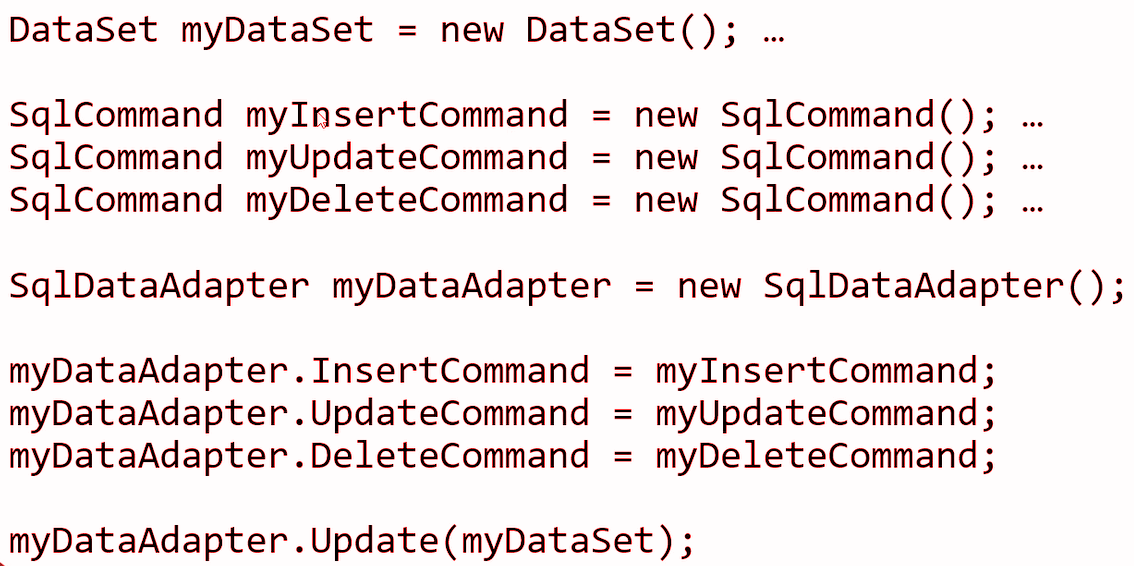
* **Primary key**
  + Analoog aan concept in concrete relationele database
  + In te stellen via primaryKey property van de DataTable
  + Geïmplementeerd als een array van DataColumn objecten
* **Relatie**
  + Analoog aan concept in concrete relationele database
  + Toe te voegen aan de **Relations** property van de DataSet
  + Geïmplementeerd als een collection
  + Toevoegen via de **Add()** method van de Relations property
    - Parameter 1: naam van de relatie
    - Parameter 2: Primary key kolom(men) van de Parent tabel
    - Parameter 3: Foreign key kolom(men) van de Child tabel

**DataAdapter**

* Gegevens moeten vanuit de data source in de DataSet geraken en omgekeerd
* DataAdapter object faciliteert deze processen via 4 commands
* Elke dataprovider heeft zijn implementatie (SqlDataAdapter voor SQL Server)



* Ophalen gegevens mbv (Sql)Command object
* SqlCommand object configureren
* Toewijzen aan **SelectCommand** property van DataAdapter
* **Fill()** method aanroepen 
  + Opent/sluit impliciet connectie, behalve wanneer connectie expliciet werd geopend
  + Gebruikt achterliggend DataReader om DataSet te vullen

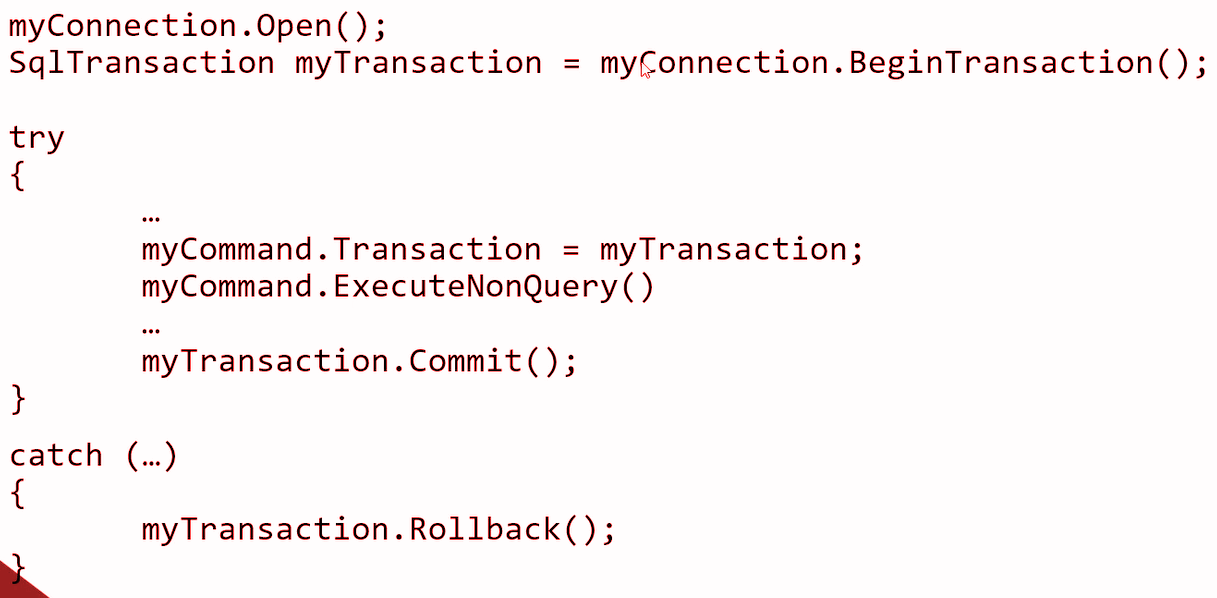


* Effectief doorvoeren van mutaties naar data source
* Instellen van InsertCommand, UpdateCommand en DeleteCommand
* DataAdapter werkt met de Changes van een DataSet via de RowState
* Changes doorvoeren via Update() method van DataAdapter
  + Voor DataRows
  + Voor ganse DataSet
  + Voor specifieke DataTable
* Voorzie SqlParameter objecten voor SqlCommand objecten
* Geef hierbij steeds de “Source” column mee

=> Nodig om matching te maken welke kolom als bronwaarde zal gebruikt worden voor de SqlParameter objecten

=> Gebruik kolomnaam zoals gebruikt in de DataSet

**Transacties**

* Transactie opstarten vanuit code
* SqlTransaction object gebruiken
* Instantiëren via **BeginTransaction()** method van SqlConnection object
* Object dat je terugkrijgt toewijzen aan SqlCommand object
* Bij succesvol uitvoeren van alle instructies transactie doorvoeren via **Commit()** method
  + Instructies inwerken in **try{}** block
* Bij optreden van fout => transactie ongedaan maken via **Rollback()** method
  + Uitwerken in **catch{}** block
* Te gebruiken bij lokale transactie

**Concurrency, locking en isolation levels**

**Concurrency**

* Transactie moet voldoen aan 4 karakteristieken voor database consistentie
* Men refereert hiernaar met de term **ACID**
  + **Atomic:**
    - Alles of niets (commit of rollback)
  + **Consistency:**
    - Data die bewaard wordt mag niet “zondigen” tegen integriteitsregels
  + **Isolation:**
    - Transacties moeten los van elkaar uitgevoerd kunnen worden
    - => meerdere transacties concurrent (gelijktijdig) uitgevoerd worden
  + **Durability:**
    - Data die werd gecommit mag niet verloren gaan bij bv. stroomuitval (hiervoor dient o.a. de transaction log in SQL Server)
* Verwerken van concurrent transacties kan voor problemen zorgen (concurrency)
* Meerdere clients die gelijktijdig zelfde record(s) aanspreken
* Welke concurrency problemen kunnen optreden:
  + Dirty reads
  + Unrepeatable read
  + Phantom reads (phantom rows)
  + Lost updates

**Dirty reads**

* Transactie B leest een record die wijzigt in de nog niet gecommitte transactie A

**Unrepeatable read**

* Bij het meermaals lezen van eenzelfde rij krijg je telkens andere waarden terug  
  => 2 personen bestellen zelfde product in een webshop waarbij voorraad wordt opgehaald

**Phantom reads (rows)**

* Bij het meermaals uitvoeren van dezelfde query in eenzelfde transactie A krijg je een verschillend aantal rijen terug
* Veroorzaakt door het toevoegen/aanpassen/verwijderen van rijen door een andere transactie B tijdens het uitvoeren van transactie A

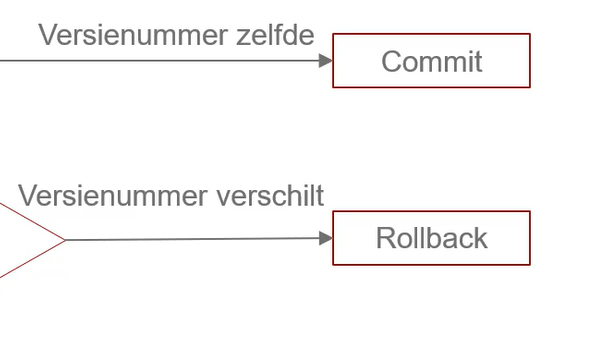
**Lost updates**

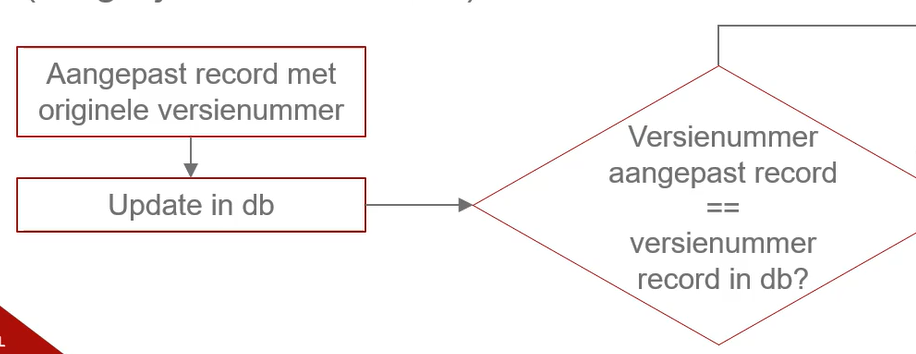
* Aanpassing van transactie A wordt overschreven door aanpassing van transactie B

**Oplossing => Locking**

* 2 manieren
  + Optimistic locking
  + Pessimistic locking

**Optimistic locking**

* Je gaat er vanuit dat meerdere transacties door elkaar kunnen lopen zonder elkaar te beïnvloeden
* Geen effectieve locking nodig
* Wel mechanisme dat valideert of een andere transactie jouw data heeft gewijzigd
  + Versienummer validatie (met een nummer of timestamp)
  + Vergelijken van waarden



**Pessimistic locking**

* Je gaat er vanuit dat meerdere transacties door elkaar zullen lopen die elkaar zullen beïnvloeden
* Meerdere soorten locks die toegepast kunnen worden:
  + Shared lock
  + Exclusive lock
  + Update lock
  + Schema locks
  + Bulk update locks



**Isolation levels**

* Pessimistic locking toepassen door isolation level in te stellen voor een transactie
* Isolation level bepaalt hoe “volatiele” data van je transactie wordt geïsoleerd van andere data
* Kan ingesteld worden in: Stored procedures, ADO.NET