

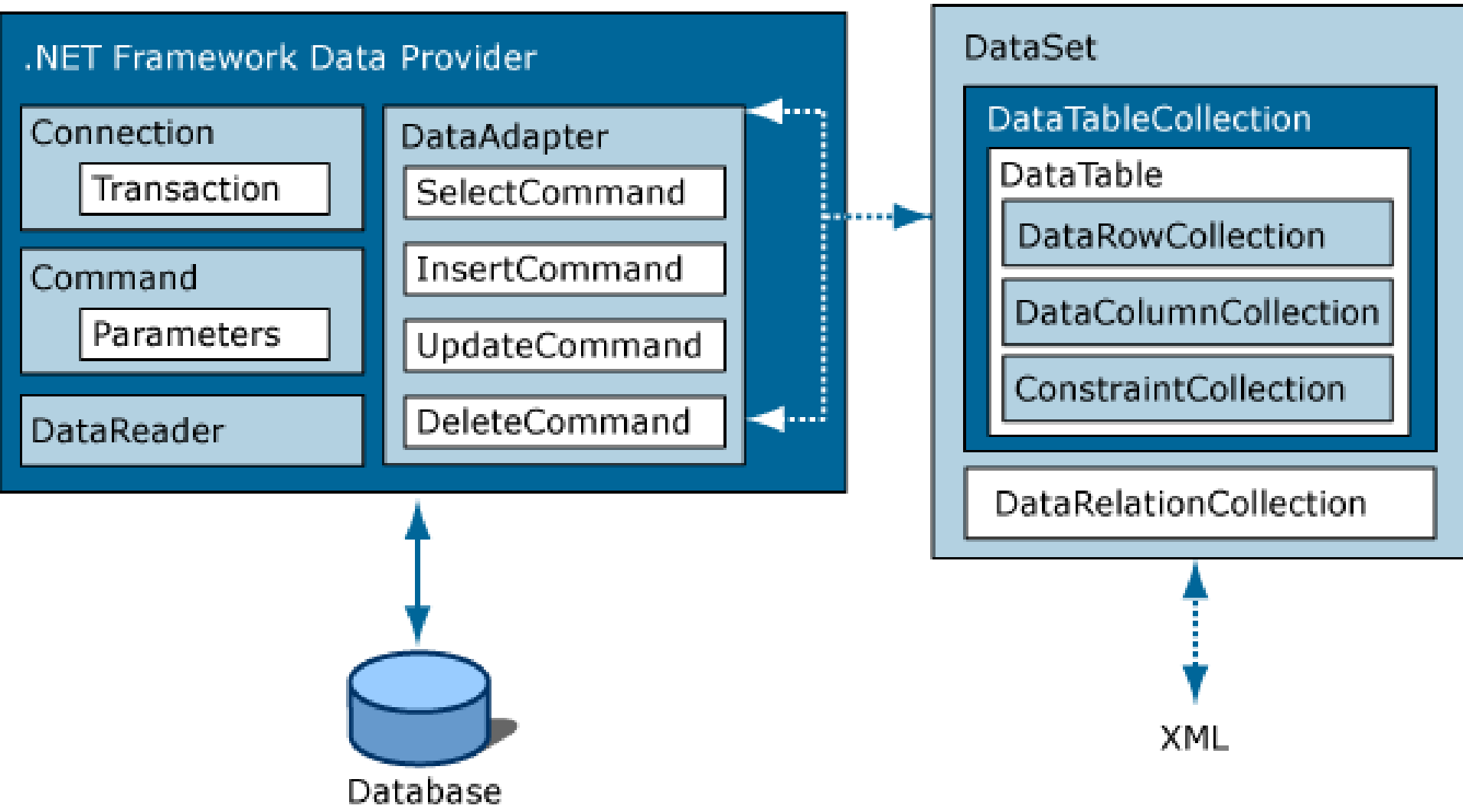
A close-up photograph of a woman with short, reddish-brown hair and bangs, smiling at the camera. She has a nose ring and is wearing a dark top. In the background, several other people are visible but out of focus.

ADO.NET

Herhaling met SQLite
DBDataAdapter
DbProviderFactories

HERHALING

ADO.NET?



Demo SQLite

Zie demo Visual Studio

Merk op: demo houdt geen rekening met presentatie, logische en data laag!

DBDATAADAPTER

Data Adapters?

DataAdapter: zorgt voor link tussen databank tabellen met tabellen in DataSet

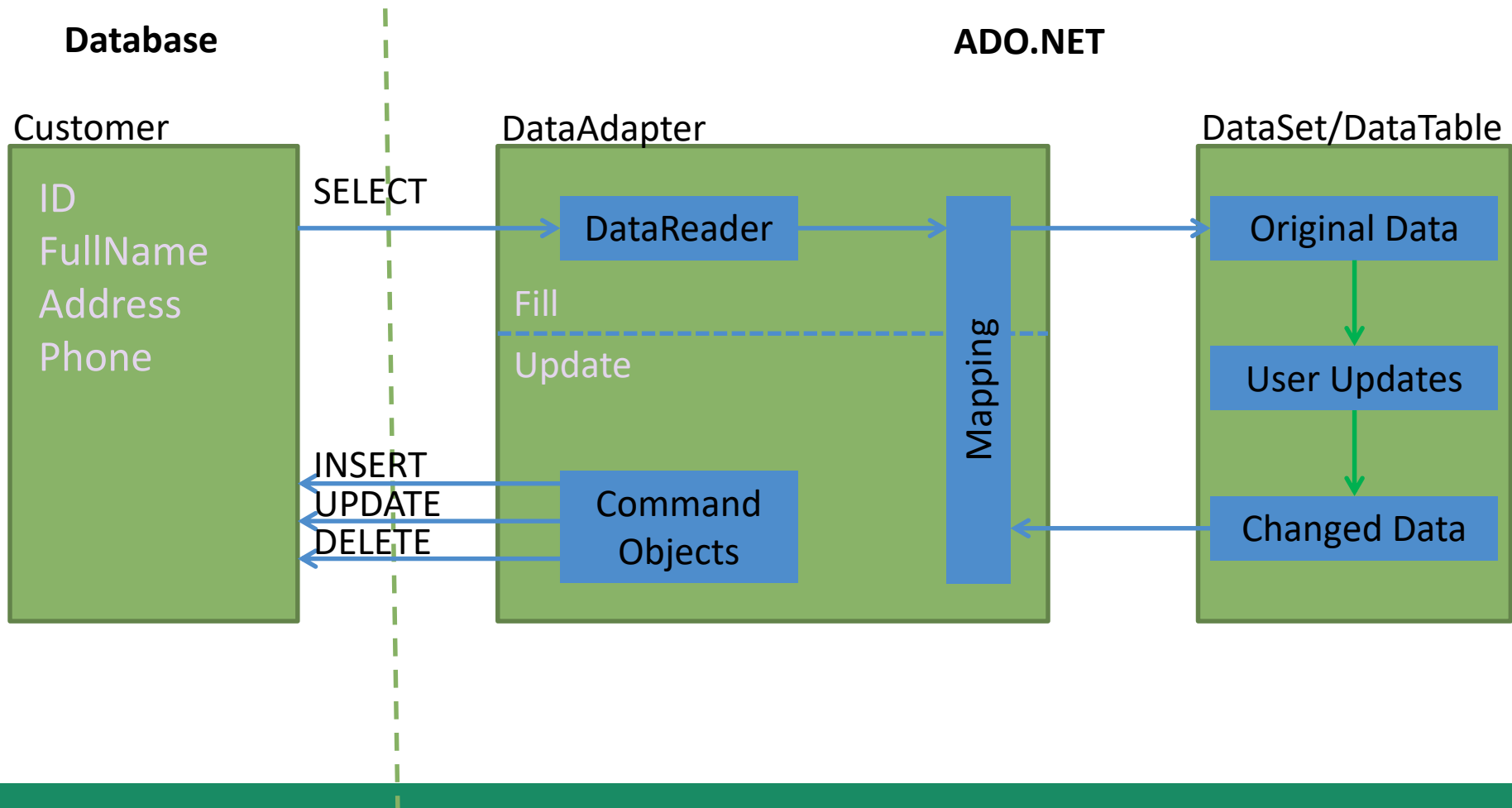
Data van databank in DataSet => Fill

- SELECT commando wordt uitgevoerd
- Data wordt in tabellen van DataSet geplaatst

In DataSet data aanpassen en nadien aanpassingen terugsturen naar databank => Update

- INSERT, UPDATE, DELETE statement worden naar databank gestuurd (dit gaat echter niet vanzelf!)

DataAdapter: grafisch voorgesteld



DataAdapter: features

Record retrieval: via select statement

Record updating:

- Ofwel zelf insert, update en delete schrijven
- Of automagisch via DBCommandBuilder, gebaseerd op select statement

Tabel- en kolomnaam mapping

- Namen van tabellen kunnen aangepast worden
- Afhankelijk van de noden van je applicatie

Data van DB => Geheugen

Data in DataTable

```
DataTable targetTable = new DataTable();  
SqlDataAdapter workAdapter = new SqlDataAdapter(  
    "SELECT * FROM Customer ORDER BY LastName",  
    connection);  
workAdapter.Fill(targetTable);
```

Data van DB => geheugen: data in DataSet

Connection hoeft niet geopend te worden

- Fill opent en sluit als operatie afgelopen is ook connectie

Data in DataSet

```
DataSet targetSet = new DataSet();  
SqlDataAdapter workAdapter = new SqlDataAdapter(  
    "SELECT * FROM Customer ORDER BY LastName",  
    connectionString);  
workAdapter.Fill(targetSet);
```

Merk op:

- De tabel gecreëerd in DataSet zal naam "Table" krijgen
- Indien je een andere naam wenst, naam van tabel opgeven

```
workAdapter.Fill(targetSet, "Customer");
```

Data aanpassen

Gebeurt in 3 stappen

- Nieuwe rij(en) creëren
- Data opslaan in rij object
- Rij object toevoegen aan tabel

Nieuwe rijen creëren

DataRow: opslaan van één rij in tabel

```
DataRow oneRow = someTable.newRow();
```

Genereerde rij bevat alle informatie over elke datakolom gedefinieerd in tabel

Rijwaarden definiëren

DataRow klasse bevat *Item* property die toegang biedt tot elke kolom op naam, of op index

```
oneRow.Item["ID"] = 123; // by column name  
oneRow.Item[0] = 123; // by column position  
DataColumn whichColumn = someTable.Columns[0];  
oneRow.Item[whichColumn] = 123; // by column instance
```

Item kan ook weggelaten worden

```
oneRow["ID"] = 123;
```

Opslaan van rijen in tabel

```
someTable.Rows.Add(oneRow);
```

Alternatief

```
// Assumes column 0 is numeric, 1 is string.  
someTable.Rows.Add(new Object[] {123, "Fred"});
```

Indien waarde niet voldoet aan opgelegde datatype van de kolom zal Add een Exception throwen

Data onderzoeken en aanpassen

Mogelijkheid om te itereren over alle rijen van de tabel

```
decimal totalTax = 0;
foreach (DataRow scanRow in someTable.Rows){
    if (!DBNull.Value.Equals(scanRow["SalesTax"])){
        totalTax += (decimal)scanRow["SalesTax"];
    }
}
```


Data uit tabel verwijderen

Voorbeeld

```
DataRow oneRow = someTable.Rows[0];  
someTable.Rows.Remove(oneRow);
```

Alternatief

```
someTable.Rows.RemoveAt(0);
```

Alternatief

```
int rowPosition = someTable.Rows.IndexOf(oneRow);
```

Om alle rijen ineens te verwijderen

```
someTable.Rows.Clear();
```

Data uit tabel verwijderen

Negatief zij-effect

- Aangezien rij helemaal verwijderd is en alle bewijs dat rij ooit bestaan heeft weg is
 - ADO.NET heeft te weinig informatie om acties uit te voeren zoals
 - Bijhouden welke records in (externe) databank moeten verwijderd worden
- Net alsof rij nooit bestaan heeft

Hiervoor bestaat gelukkig een oplossing ;-)

Veranderingen in batch uitvoeren

In plaats van onmiddellijk resultaat, in DataTable alles opsparen en dan ineens uitvoeren

- Zal gebruikt worden om veranderingen aan data door te geven aan (externe) databank
- Zo kan bijgehouden worden wat er precies gewijzigd is
- Data integriteit:
 - Lokale data en die in databank zijn gelijk

```
someTable.AcceptChanges(); // Commit all row changes
```

```
someTable.RejectChanges(); // Reject changes since last commit
```

Status van rijen

Tijdens het maken van verandering aan rijen

- ADO.NET houdt de originele en “proposed” versie bij van alle velden
- Houdt bij welke rijen nieuw of verwijderd zijn en kan eventueel naar de originele rij terugkeren indien nodig

**Wordt bijgehouden in DataRow.RowState
property**

Mogelijke waarden DataRow.RowState

Enumeratiewaarden

- *DataRowState.Detached*: rijen die nog niet toegevoegd zijn aan DataTable
- *DataRowState.Added*: status van rijen die toegevoegd zijn, maar veranderingen aan tabel zijn nog niet bevestigd
- *DataRowState.Unchanged*: rijen die in tabel bevinden, en die niet gewijzigd zijn, sinds laatste AcceptChanges
- *DataRowState.Deleted*: rijen die verwijderd zijn, maar nog niet bevestigd (nog geen AcceptChanges uitgevoerd)
- *DataRowState.Modified*: rij waaraan iets veranderd is

Telkens je rij aanpast, zal RowState aangepast worden

Indien je rij verwijdert: gebruik Delete methode van DataRow klasse

- Rij wordt nog niet verwijderd, maar status staat wel op delete

Rows.Remove en Rows.RemoveAt omzeilt dit mechanisme en informatie over toestand van rij gaat verloren

Veranderingen in batch uitvoeren:

Delete methode

Rij wordt hierdoor niet verwijderd

Rij wordt enkel gemarkeerd als verwijderd

Na het gebruik van AcceptChanges methode

- Rij zal permanent verwijderd worden

```
someTable.Rows.Remove(oneRow); // Removes row immediately  
oneRow.Delete(); // Marks row for removal during approval
```

Rijversies

Veranderingen aan data in rijen

- ADO.NET houdt kopijen bij van elke waarde die veranderd is

Rijversie info slaat altijd op een ganse rij, ook als er maar 1 kolom is veranderd

- *DataRowVersion.Original*: waarde van het veld voor het veranderd werd (waarde na het uitvoeren van meest recente AcceptChanges)
- *DataRowVersion.Proposed*: veranderd, maar nog niet bevestigd. Na uitvoeren van AcceptChanges: proposed => original
- *DataRowVersion.Current*:
 - Indien nog niet bevestigd: zelfde als proposed
 - Indien wel bevestigd: zelfde als original
- *DataRowVersion.Default*:
 - Indien rij "attached" aan DataTable: zelfde als Current
 - Indien rij "detached": zelfde als Proposed

Rijversies

Afhankelijk van huidige toestand van een rij

- Sommige rijversies zullen al dan niet bestaan

Om te bepalen of een bepaalde rijversie bestaat

- DataRow.HasVersion methode gebruiken

```
if (oneRow.HasVersion(DataRowVersion.Proposed)){  
    if (oneRow.Item["Salary", DataRowVersion.Original] !=  
        oneRow.Item["Salary", DataRowVersion.Proposed]){  
        MessageBox.Show("Proposed salary change.");  
    }  
}
```


Data van geheugen => databank

DataAdapter kan dus ook gewijzigde data in DataSet terug naar databank schrijven

In principe moet je hiervoor

- Insert, update, en delete commando's schrijven
- Bevatten een SQL statement, SqlConnection referentie, en parameters

Data van geheugen => databank: voorbeeld

```
// Build the selection query.
SqlDataAdapter unitAdapter = new SqlDataAdapter();
SqlCommand unitCommand = new SqlCommand("SELECT * FROM UnitOfMeasure", linkToDB);
unitAdapter.SelectCommand = unitCommand;

// Build the insertion query.
unitCommand = new SqlCommand(@"INSERT INTO UnitOfMeasure (ShortName, FullName)
                              VALUES (@ShortName, @FullName);
                              SET @ID = @@IDENTITY;", linkToDB);
unitCommand.Parameters.Add("@ShortName", SqlDbType.VarChar, 15, "ShortName");
unitCommand.Parameters.Add("@FullName", SqlDbType.VarChar, 50, "FullName");
SqlParameter param = unitCommand.Parameters.Add("@ID", SqlDbType.BigInt, 0, "ID");
param.Direction = ParameterDirection.Output;
unitAdapter.InsertCommand = unitCommand;
```

ID	ShortName	FullName
1	Kg	kilogram
2	oz	ounce

Data van Geheugen => DB: voorbeeld

// Build the revision query.

```
unitCommand = new SqlCommand(@"UPDATE UnitOfMeasure
                              SET ShortName = @ShortName,
                              FullName = @FullName WHERE ID = @ID", linkToDB);
unitCommand.Parameters.Add("@ShortName", SqlDbType.VarChar, 15, "ShortName");
unitCommand.Parameters.Add("@FullName", SqlDbType.VarChar, 50, "FullName");
param = unitCommand.Parameters.Add("@ID", SqlDbType.BigInt, 0, "ID");
param.SourceVersion = DataRowVersion.Original;
unitAdapter.UpdateCommand = unitCommand;
```

// Build the deletion query.

```
unitCommand = new SqlCommand("DELETE FROM UnitOfMeasure WHERE ID = @ID", linkToDB);
param = unitCommand.Parameters.Add("@ID", SqlDbType.BigInt, 0, "ID");
param.SourceVersion = DataRowVersion.Original;
unitAdapter.DeleteCommand = unitCommand;
```

Uitleg

```
unitCommand.Parameters.Add("@ShortName", SqlDbType.VarChar, 15,  
"ShortName");
```

Soort mapping tussen kolomnaam en parameter naam

```
INSERT INTO UnitOfMeasure (ShortName, FullName)  
VALUES (@ShortName, @FullName);  
SET @ID = @@IDENTITY;
```

Bepalen van de primary key bij het invoegen

2 statements in 1 batch

```
param.Direction = ParameterDirection.Output;
```

Dit geeft aan dat ondertussen de waarde van de PK ID naar de DataTable wordt gestuurd

Vervolg uitleg

Zoals gezegd: oppassen bij Delete

- info is reeds verloren (want Delete gedaan) wanneer Update (van DataSet naar DB) wordt uitgevoerd

```
param = unitCommand.Parameters.Add("@ID", SqlDbType.BigInt, 0, "ID");  
param.SourceVersion = DataRowVersion.Original;
```

Record met bewuste ID is verdwenen uit DataTable als update naar DB moet gebeuren

Daarom: 'originele' versie van de record gebruiken (voor delete in DataTable, of sinds de laatste AcceptChanges)

Update uitvoeren

Als alle insert, update, delete statements er zijn

Tijd om update uit te voeren

```
workAdapter.Update(localTable);
```

Automatisch genereren van update commando's

Indien veranderingen gebeurt aan DataSet

- Terugschrijven naar databank

DBDataAdapter doet dit niet automatisch

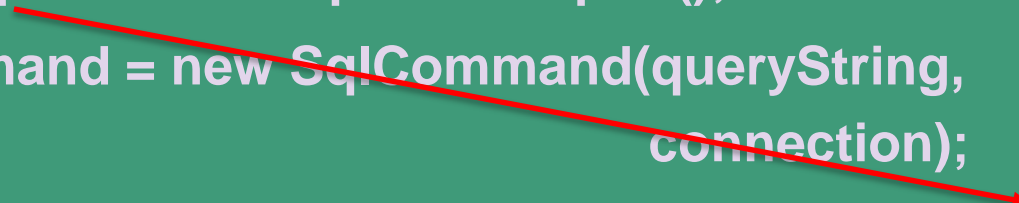
- In feite moet men zelf insert, update en delete commando's schrijven om de wijzigingen terug te schrijven naar databank

Echter:

- Via SqlCommandBuilder gaat dit automatisch
- Op voorwaarde dat er maar 1 tabel in het spel is

Voorbeeld

```
public static DataSet SelectSqlRows(string connectionString,  
                                   string queryString, string tableName) {  
    using (SqlConnection connection = new  
           SqlConnection(connectionString)){  
        SqlDataAdapter adapter = new SqlDataAdapter();  
        adapter.SelectCommand = new SqlCommand(queryString,  
                                                connection);  
        SqlCommandBuilder builder = new SqlCommandBuilder(adapter);  
        connection.Open();  
    }
```



Voorbeeld

```
DataSet dataSet = new DataSet();  
adapter.Fill(dataSet, tableName);  
//code to modify data in DataSet here  
builder.GetUpdateCommand();  
//Without the SqlCommandBuilder this line would fail  
adapter.Update(dataSet, tableName);  
return dataSet;  
}  
}
```

Beperkingen

Niet mogelijk met joins

Schema moet minstens PK of unieke waarden bevatten

Indien SelectCommand geassocieerd met DataAdapter verandert

- SqlCommandBuilder.RefreshSchema methode aanroepen om aangepaste queries te genereren

Zal enkel commando's genereren voor deze acties die nog niet gedefinieerd zijn in SqlDataAdapter.

- Bijv: stel SelectCommand en InsertCommand bestaan, dan zal enkel UpdateCommand en DeleteCommand "gemaakt" worden

Indien veldnamen spaties bevatten => zelf update statements schrijven

Tabel en kolom mapping

Soms niet mogelijk om zelfde tabel- en kolomnamen voor databank en DataTable te gebruiken

In deze situaties:

- `SqlDataAdapter.TableMappings` gebruiken
 - Mapping tussen namen in (externe) databank en DataTable

Bijv. DataSet met tabellen en corresponderende namen bestaat al, en je wilt data uit databank importeren

Mapping: voorbeeld

// Using the basic external-internal syntax is quick.

```
workAdapter.TableMappings.Add("Table", "Employee");
```

// Adding a DataTableMapping instance works also.

```
DataTableMapping nameChange = new DataTableMapping();
```

```
nameChange.SourceTable = "Table1";
```

```
nameChange.DataSetTable = "Customer";
```

```
workAdapter.TableMappings.Add(nameChange);
```

Kolommen kunnen ook gemapt worden

```
// Start with the table name.
```

```
DataTableMapping employeeMap =  
workAdapter.TableMappings.Add("Table", "Employee");
```

```
// Then add the columns
```

```
employeeMap.ColumnMappings.Add("Employee ID",  
"ID");
```

```
employeeMap.ColumnMappings.Add("Current  
Department", "DeptID");
```

SqlDataAdapter.MissingMappingAction property

Niet nodig om voor elke kolom of tabel een mapping te voorzien

SqlDataAdapter.MissingMappingAction property regelt wat er moet gebeuren in dat geval

- `MissingMappingAction.PassThrough`: gebruik de kolomnaam uit de databank (default)
- `MissingMappingAction.Ignore`: kolommen zonder mapping worden genegeerd
- `MissingMappingAction.Error`: fout wordt opgeworpen

SqlDataAdapter.MissingSchemaAction property

Eigenschap geeft aan wat er dient te gebeuren als doelDataSet of –DataTable de gemapte kolomnamen niet bevat

- **MissingSchemaAction.Add:** kolommen worden automatisch toegevoegd (default)
- **MissingSchemaAction.AddWithKey:** zelfde als hierboven + PK wordt geïmporteerd
- **MissingSchemaAction.Ignore:** ontbrekende kolommen worden genegereerd
- **MissingSchemaAction.Error:** throwt een exception

Voorbeeld

```
// Assumes a valid connection string to an Access database.
static void CreateOleDbAdapter(string connectionString) {
    OleDbDataAdapter adapter = new OleDbDataAdapter();
    adapter.SelectCommand = new OleDbCommand("SELECT * FROM
                                             Categories ORDER BY CategoryID");
    adapter.SelectCommand.Connection = new
                                     OleDbConnection(connectionString);
    adapter.MissingMappingAction = MissingMappingAction.Error;
    adapter.MissingSchemaAction = MissingSchemaAction.Error;
}
```


Bronnen

Microsoft ADO.NET 4 Step by Step

<http://msdn.microsoft.com/en-us/library/wda6c36e.aspx>

DBPROVIDERFACTORIES

DbProviderFactory

Aanwezig in .NET sinds 2.0

Basisklassen zijn abstract => kunnen niet direct geïntantieerd worden

- DbConnection
- DbCommand
- DbDataAdapter

Worden gedeeld door de .NET data providers zoals System.Data.SqlClient, System.Data.OleDb, System.Data.SQLite, ...

Laat ontwikkelaar toe om generieke data access code te schrijven die niet afhankelijk is van een specifieke dataprovider!

Factory patroon (pattern)

Object dat volgens factory patroon gecreëerd is, wordt gebruikt om andere objecten te maken

Voorbeeld + DEMO

Connection klasse

```
public abstract class Connection {  
    public Connection() { }  
  
    public abstract String Description() ;  
}
```

SqlServerConnection klasse

```
public class SqlServerConnection : Connection {  
    public SqlServerConnection() {}  
    public override String Description() {  
        return "SQL Server";  
    }  
}
```

OracleConnection klasse

```
public class OracleConnection : Connection {  
  
    public OracleConnection() { }  
  
    public override string Description(){  
        return "Oracle";  
    }  
}
```

MySQLConnection klasse

```
public class MySQLConnection : Connection {  
  
    public MySQLConnection() { }  
  
    public override String Description() {  
        return "MySQL";  
    }  
}
```


FirstFactory klasse

```
public class FirstFactory {  
    protected string type;  
    public FirstFactory(string t) {  
        type = t;  
    }  
    public Connection CreateConnection() {  
        if (type == "Oracle") {  
            return new OracleConnection();  
        }  
        else if (type == "SQL Server") {  
            return new SqlConnection();  
        }  
        else {  
            return new MySqlConnection();  
        }  
    }  
}
```

Main klasse

```
public class Program {  
    static void Main(string[] args) {  
        FirstFactory factory = new FirstFactory("Oracle");  
        Connection conn = factory.CreateConnection();  
        Console.WriteLine("You are connected with " +  
            conn.Description());  
    }  
}
```

DbProviderFactories klasse

Je kan mbv DbProviderFactories klasse opvragen welke data providers er op je PC aanwezig zijn

Zal daarvoor de *machine.config* file raadplegen op je PC (op mijn machine te vinden onder C:\Windows\Microsoft.NET\Framework\v4.0.30319\Config)

Opvragen van beschikbare dataproviders

```
static DataTable GetProviderFactoryClasses() {  
    // Retrieve the installed providers and factories.  
    DataTable table = DbProviderFactories.GetFactoryClasses();  
  
    // Display each row and column value.  
    foreach (DataRow row in table.Rows) {  
        foreach (DataColumn column in table.Columns) {  
            MessageBox.Show(row[column]+"");  
        }  
    }  
    return table;  
}
```

Connection string opvragen adhv provider naam

Je kan specifieke informatie over connection string en provider opslaan in app.config bestand (in je project)

Voorbeeld van zo'n app.config bestand:

```
<configuration>
  <connectionStrings>
    <clear/>
    <add name="NorthwindSQL" providerName=
      "System.Data.SqlClient" connectionString= "Data Source =
      sqlExpress; Initial Catalog=Northwind; Integrated
      Security=SSPI" />
  </connectionStrings>
</configuration>
```

Code om connection string op te halen

Om een provider factory aan te maken, nood aan

- Connection string
- Provider name

Aanwezig in app.config bestand

```
// Retrieve a connection string by specifying the providerName.  
// Assumes one connection string per provider in the config file.  
static string GetConnectionStringByProvider(string providerName) {  
    // Return null on failure.  
    string returnValue = null;  
    // Get the collection of connection strings.  
    ConnectionStringSettingsCollection settings =  
        ConfigurationManager.ConnectionStrings;  
  
    // Walk through the collection and return the first  
    // connection string matching the providerName.  
    if (settings != null) {  
        foreach (ConnectionStringSettings cs in settings) {  
            if (cs.ProviderName == providerName)  
                returnValue = cs.ConnectionString;  
            break;  
        }  
    }  
    return returnValue;  
}
```

DbProviderFactory & DbConnection

```
// Given a provider name and connection string,  
// create the DbProviderFactory and DbConnection.  
// Returns a DbConnection on success; null on failure.  
static DbConnection CreateDbConnection(string providerName, string  
                                       connectionString) {  
  
    // Assume failure.  
    DbConnection connection = null;  
    // Create the DbProviderFactory and DbConnection.  
    if (connectionString != null) {  
        try {  
            DbProviderFactory factory =  
                DbProviderFactories.GetFactory(providerName);  
            connection = factory.CreateConnection();  
            connection.ConnectionString = connectionString;  
        }  
    }  
}
```


DbProviderFactory & DbConnection

```
        catch (Exception ex) {  
            // Set the connection to null if it was created.  
            if (connection != null) {  
                connection = null;  
            }  
            Console.WriteLine(ex.Message);  
        }  
    }  
    // Return the connection.  
    return connection;  
}
```

Data ophalen mbv DbConnection, DbCommand

```
// Takes a DbConnection and creates a DbCommand to retrieve data
// from the Categories table by executing a DbDataReader.
static void DbCommandSelect(DbConnection connection) {
    string queryString = "SELECT CategoryID, CategoryName FROM Categories";
    // Check for valid DbConnection.
    if (connection != null) {
        using (connection) {
            try {
                // Create the command.
                DbCommand command = connection.CreateCommand();
                command.CommandText = queryString;
                command.CommandType = CommandType.Text;
                // Open the connection.
                connection.Open();
                // Retrieve the data.
                DbDataReader reader = command.ExecuteReader();
            }
        }
    }
}
```

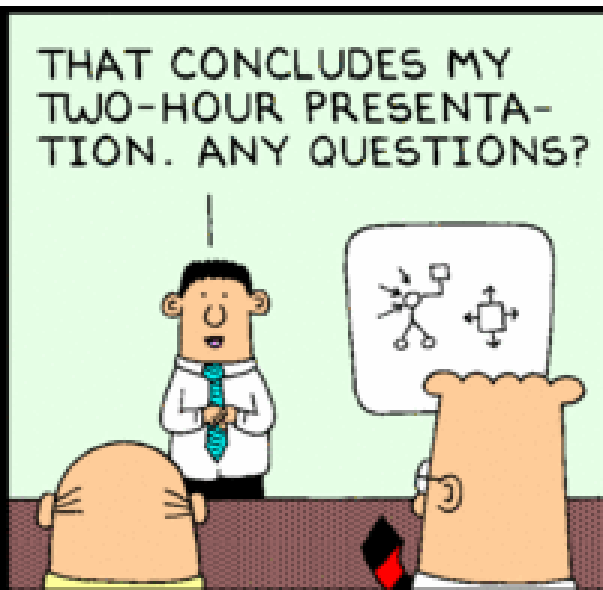
Data ophalen mbv DbConnection, DbCommand

```
        while (reader.Read()) {  
            Console.WriteLine("{0}. {1}", reader[0], reader[1]);  
        }  
    }  
    catch (Exception ex) {  
        Console.WriteLine("Exception.Message: {0}", ex.Message);  
    }  
}  
}  
else {  
    Console.WriteLine("Failed: DbConnection is null.");  
}  
}
```

DEMO

In Visual Studio

Dilbert



www.dilbert.com scottadams@aol.com



8/4/03 © 2003 United Feature Syndicate, Inc.

