2022/2023(2)
IF184504 Web Programming

Lecture #12

ADO.NET

Misbakhul Munir IRFAN SUBAKTI
司馬伊凡
Мисбакхул Мунир Ирфан Субакти

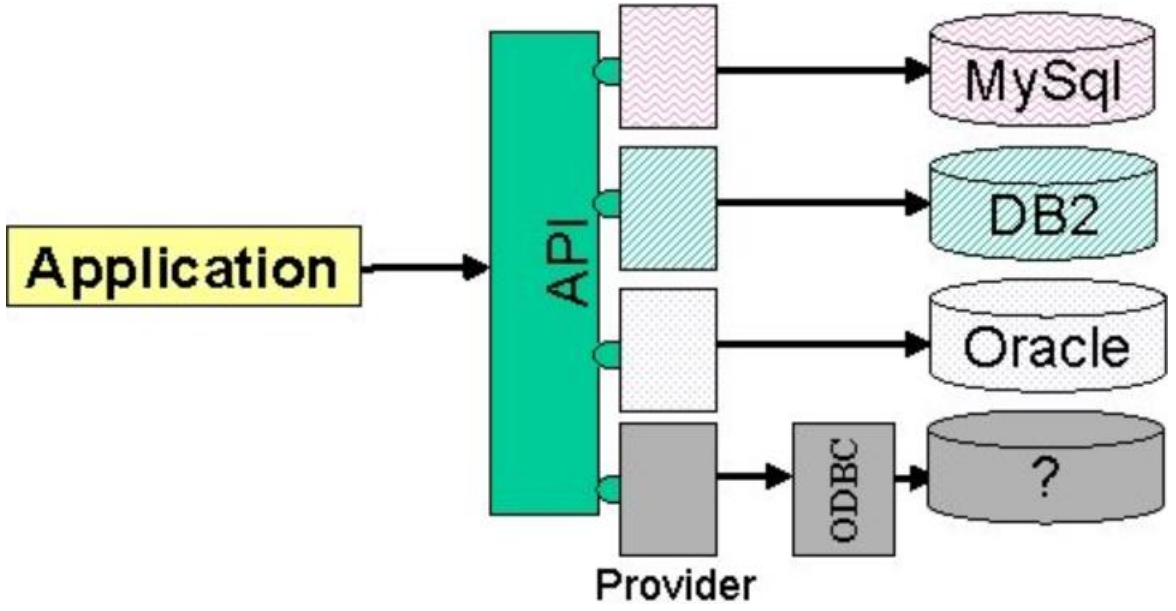
ADO.NET: What is that?

- Technology used .NET for accessing data structured, where this technology handling different data source into a single interface
- Uniform object oriented interface for different data sources
  - Relational database
  - XML data
  - Any other data sources: text, CSV, etc.
- ADO.NET technology designed for web-based distributed applications
- Two kind of accessing data models
  - Connection-oriented
  - Connectionless

Universal data access: Data uniformity

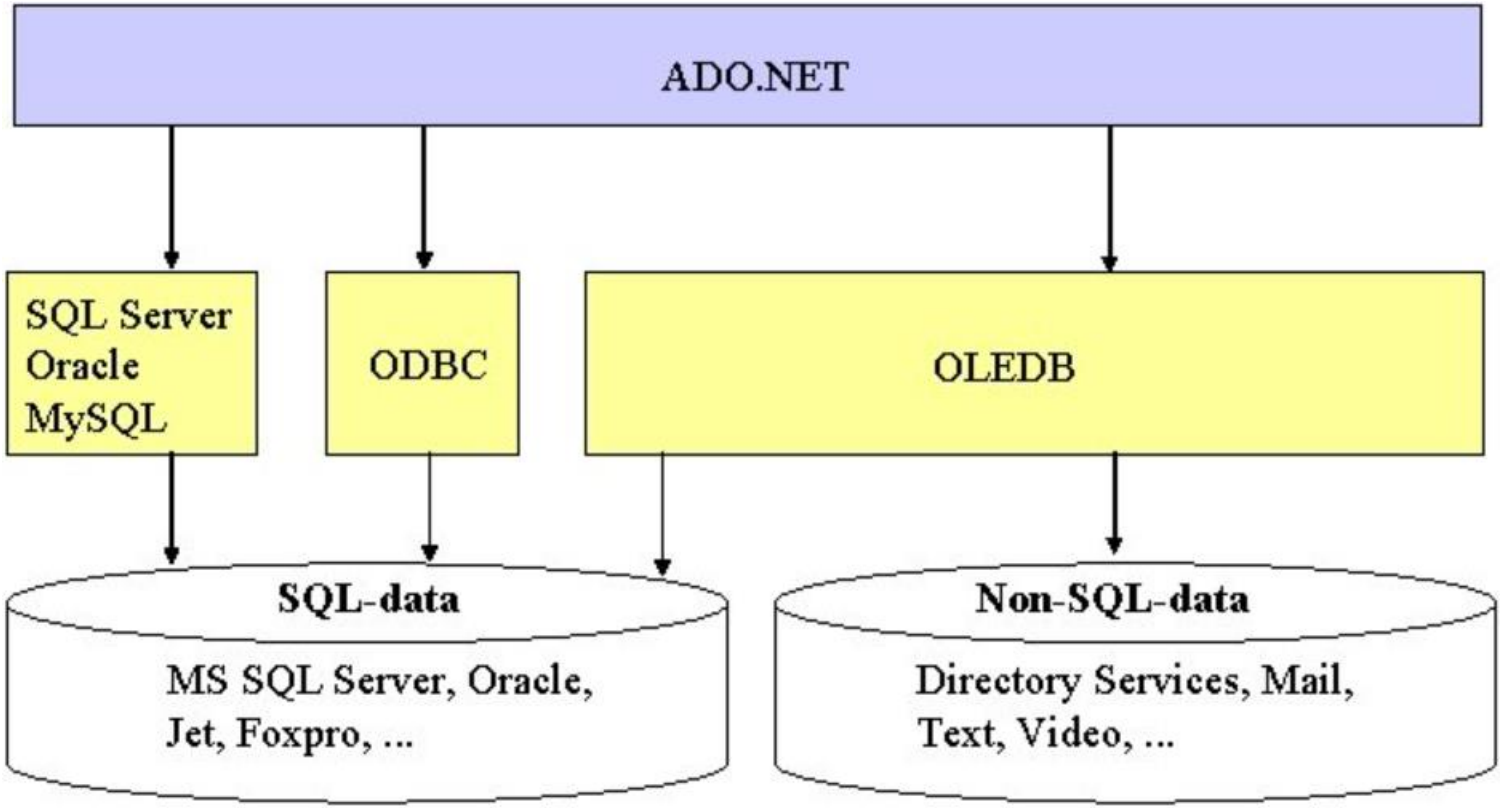
- Object-Oriented Programming concept extended to data
  - Data can be seen as an object → its source doesn't matter anymore
- Need a bridge (like an *Application Programming Interface – API,* in programming) to access data source via a provider

Application & data source: Relationship



Data providers: Enabling data access for app

- Microsoft's layered architecture for data access



Universal data access (Microsoft): History of

- Prior to ADO.NET, there was ADO (ActiveX Data Objects) technology
- It's Microsoft's technology for accessing data source

  - ODBC
    - OLE DB
      - ADO (ActiveX Data Objects)
        - ADO.NET

ADO vs ADO.NET: Comparison

ADO

- Connection-oriented
- Sequential access
- Only one table supported
- COM-marshalling

ADO.NET

- Connection-oriented + connectionless
- Main-memory representation with direct access
- More than one table supported
- XML-marshalling

ADO.NET: Architecture

- Data processing has traditionally relied primarily on a connection-based, two-tier model.
- As data processing increasingly uses multi-tier architectures, programmers are switching to a disconnected approach to provide better scalability for their applications.

ADO.NET: Components

- The two main components of ADO.NET for accessing and manipulating data are
  - .NET Framework data providers, and
  - DataSet.

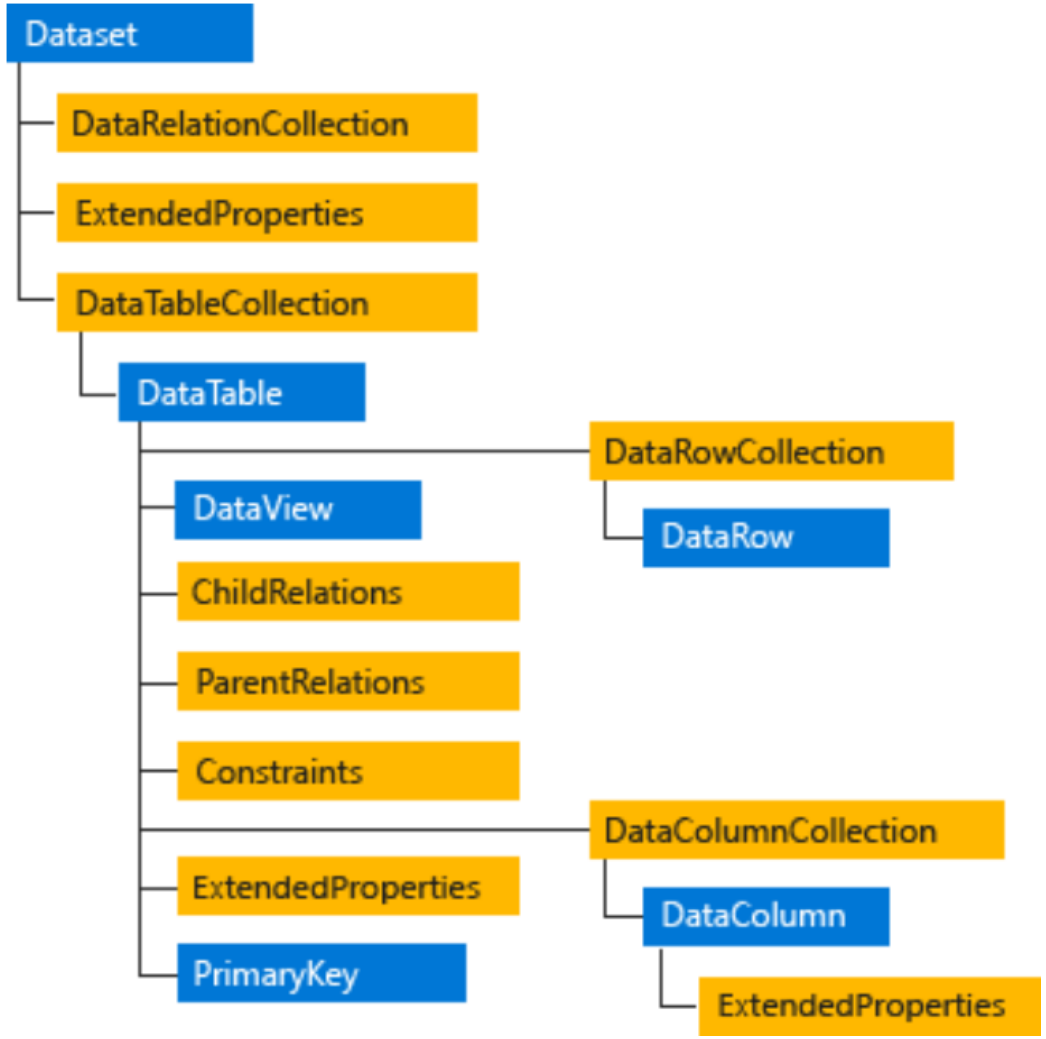
.NET Framework data providers

- The .NET Framework Data Providers are components that have been explicitly designed for data manipulation and fast, forward-only, read-only access to data.
- The Connection object provides connectivity to a data source.
- The Command object enables access to database commands to return data, modify data, run stored procedures, and send or retrieve parameter information.
- The DataReader provides a high-performance stream of data from the data source.
- Finally, the DataAdapter provides the bridge between the DataSet object and the *data source*.
  - The *DataAdapter* uses *Command* objects to execute SQL commands at the data source to both load the *DataSet* with data and reconcile changes that were made to the data in the *DataSet* back to the data source.

DataSet

- The ADO.NET DataSet is explicitly designed for data access independent of any data source.
- As a result, it can be used with multiple and differing data sources, used with XML data, or used to manage data local to the application.
- The DataSet contains a collection of one or more *DataTable* objects consisting of rows and columns of data, and also primary key, foreign key, constraint, and relation information about the data in the *DataTable* objects.

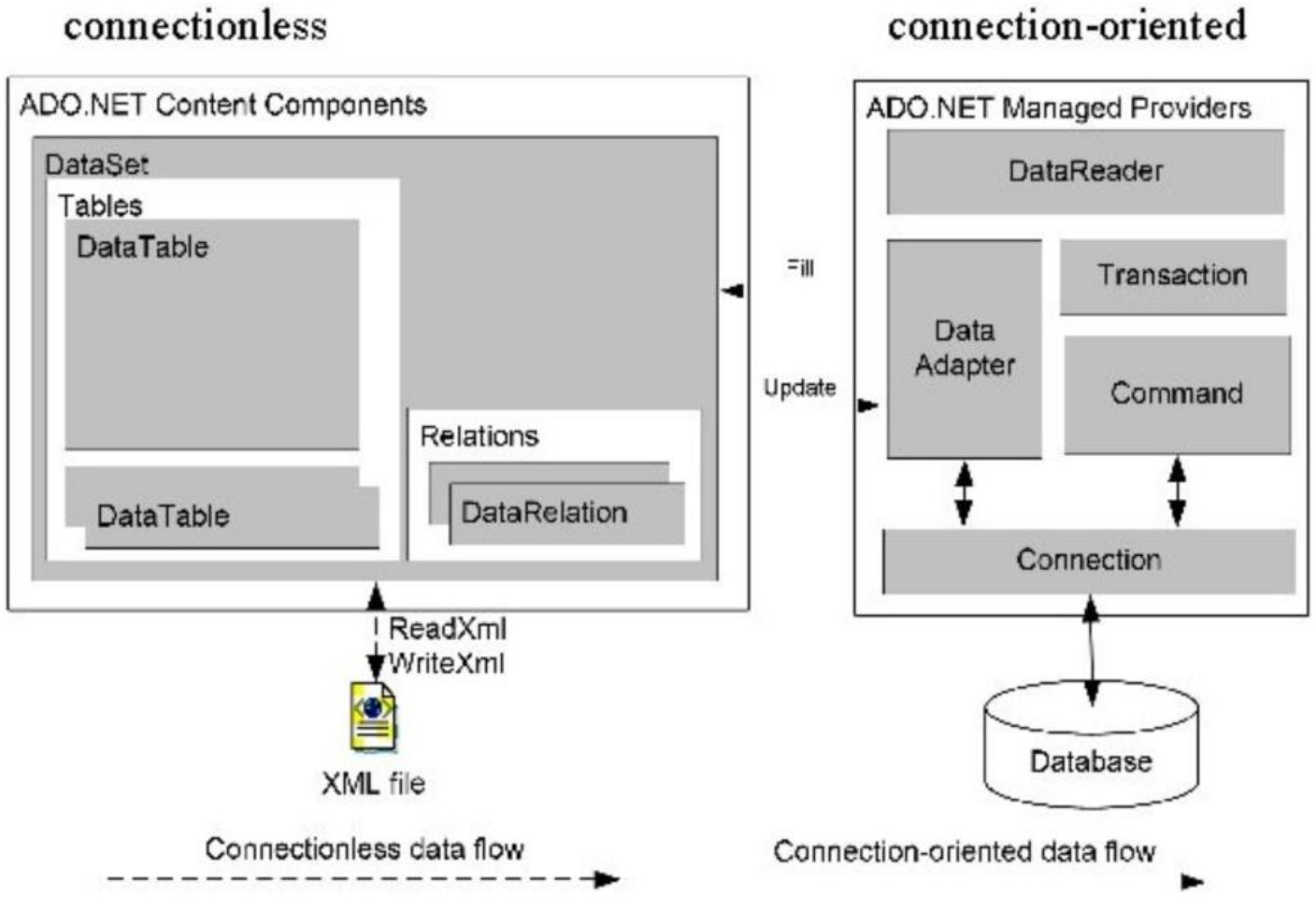
.NET Framework data providers & DataSet



DataReader or DataSet?

- When you decide whether your application should use a DataReader or a DataSet, consider the type of functionality that your application requires. Use a DataSet to do the following:
  - Cache data locally in your application so that you can manipulate it. If you only need to read the results of a query, the DataReader is the better choice.
  - Remote data between tiers or from an XML Web service.
  - Interact with data dynamically such as binding to a Windows Forms control or combining and relating data from multiple sources.
  - Perform extensive processing on data without requiring an open connection to the data source, which frees the connection to be used by other clients.
- If you do not require the functionality provided by the DataSet, you can improve the performance of your application by using the DataReader to return your data in a forward-only, read-only manner.
- Although the DataAdapter uses the DataReader to fill the contents of a DataSet, by using the DataReader, you can boost performance because you will save memory that would be consumed by the DataSet, and avoid the processing that is required to create and fill the contents of the DataSet.

ADO.NET: Connection architecture



Connection-oriented vs Connectionless

- Connection-oriented
  - Keeps the connection to the data base alive
  - Intended for application with
    - Short running transactions
    - Only a few parallel accesses
    - Up-to-date data
- Connectionless
  - No permanent connection to the data source
  - Data cached in main memory
  - Changes in main memory ≠ changes in data source
  - Intended for application with
    - Many parallel and long lasting accesses, e.g., web applications

ADO.NET: Assembly & Namespaces

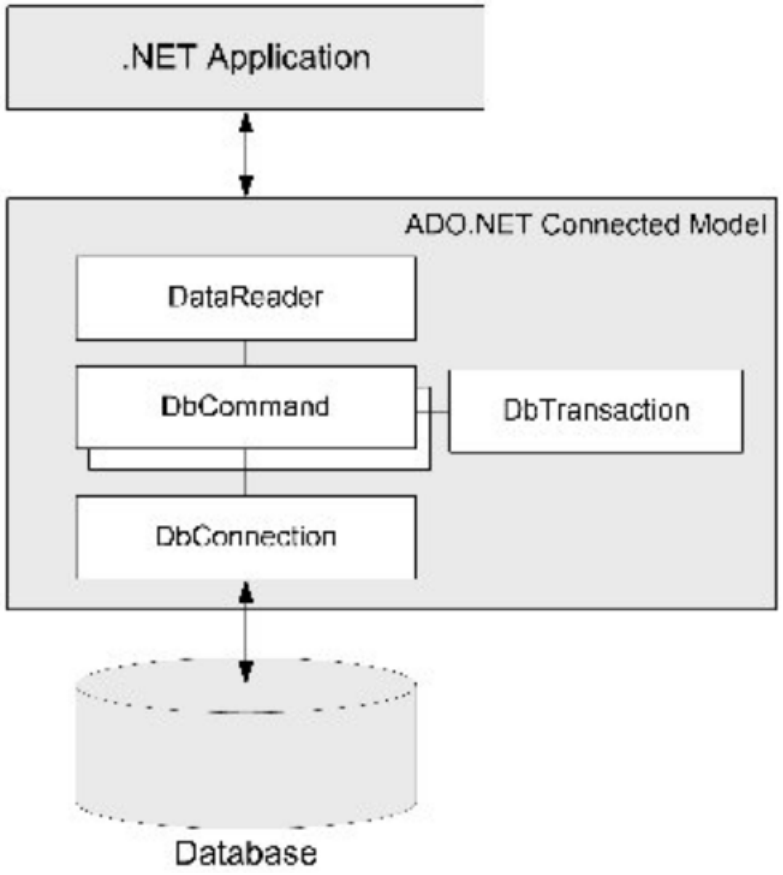
- ADO.NET access involved two kind of libraries
  - Assembly
    - System.Data.dll
  - Namespaces
    - System.Data — General data types
    - System.Data.Common — Classes for implementing providers
    - System.Data.OleDb — OLE DB provider
    - System.Data.SqlClient — Microsoft SQL Server provider
    - System.Data.SqlTypes — Data types for SQL Server
    - System.Data.Odbc — ODBC provider (since .NET 1.1)
    - System.Data.OracleClient — Oracle provider (since .NET 1.1)
    - System.Data.SqlServerCe — Compact Framework

Connection-oriented

ADO.NET

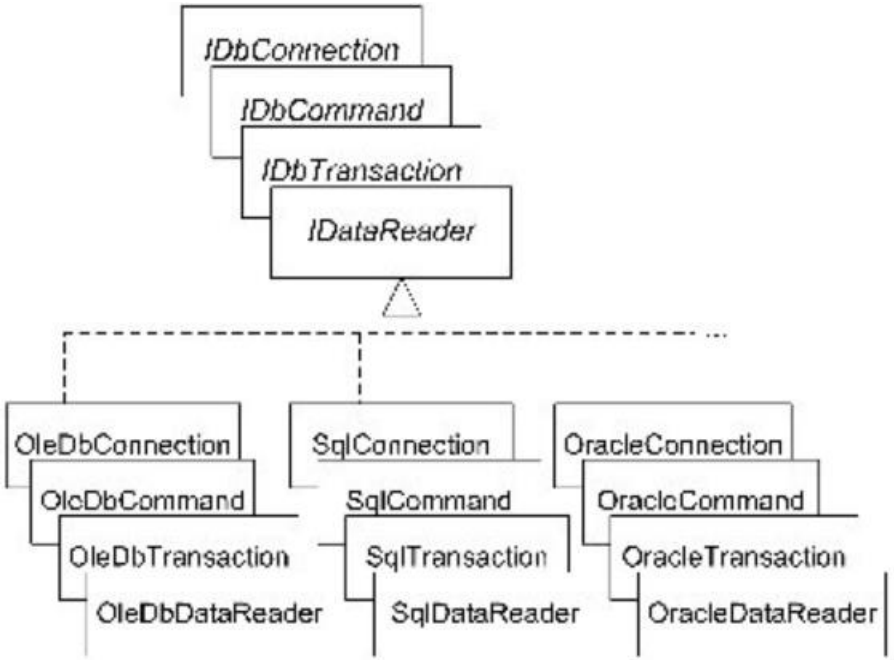
Connection-oriented: Architecture

- DbConnection
  - Represents connection to the data source
- DbCommand
  - Represents a SQL command
- DbTransaction
  - Represents a transaction
  - Commands can be executed within a transaction
- DataReader
  - Results of a database query
  - Allows sequential reading of rows

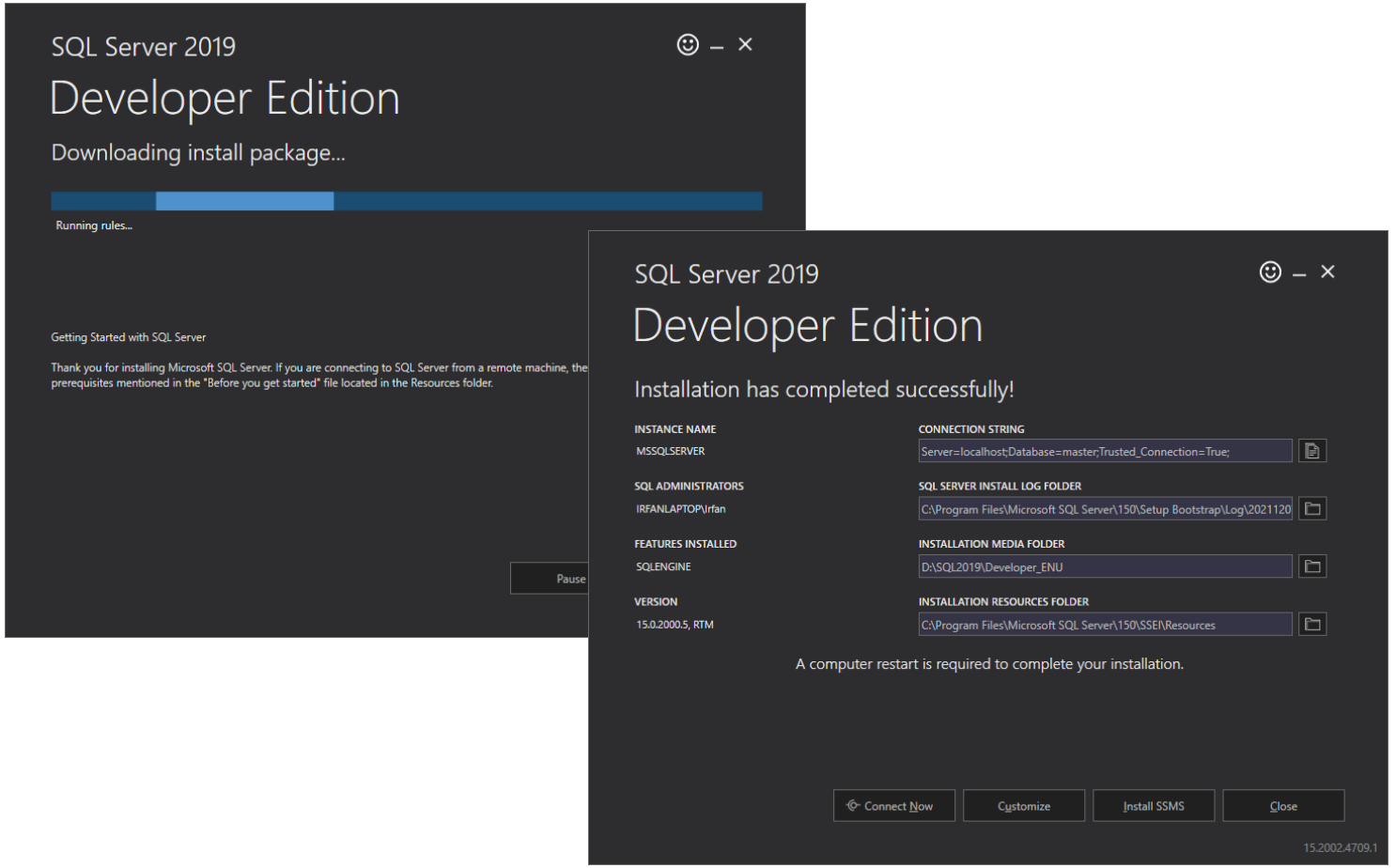


Class hierarchy

- General interface definitions
  - IDbConnection
  - IDbCommand
  - IDbTransaction
  - IDataReader
- Special implementations
  - OleDb: implementation for OLEDB
  - Sql: implementation for SQL Server
  - Oracle: implementation for Oracle
  - Odbc: implementation for ODBC
  - SqlCe: implementation for SQL Server CE database



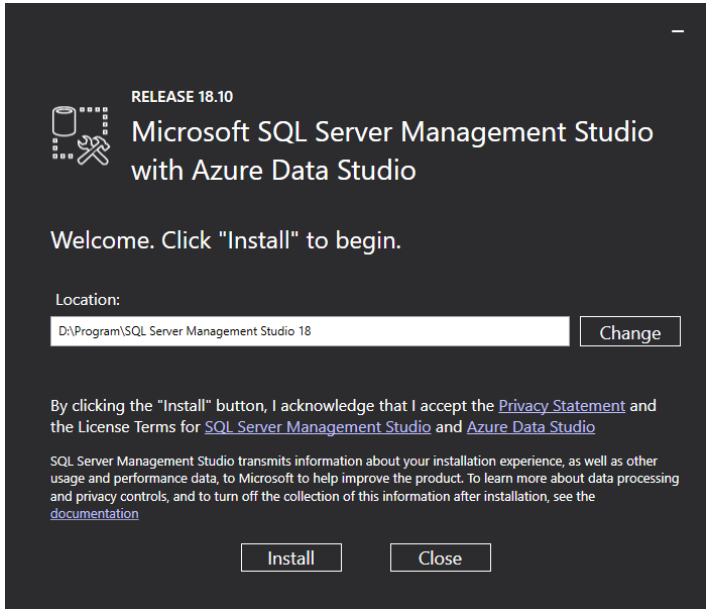
Microsoft SQL Server: Download & install

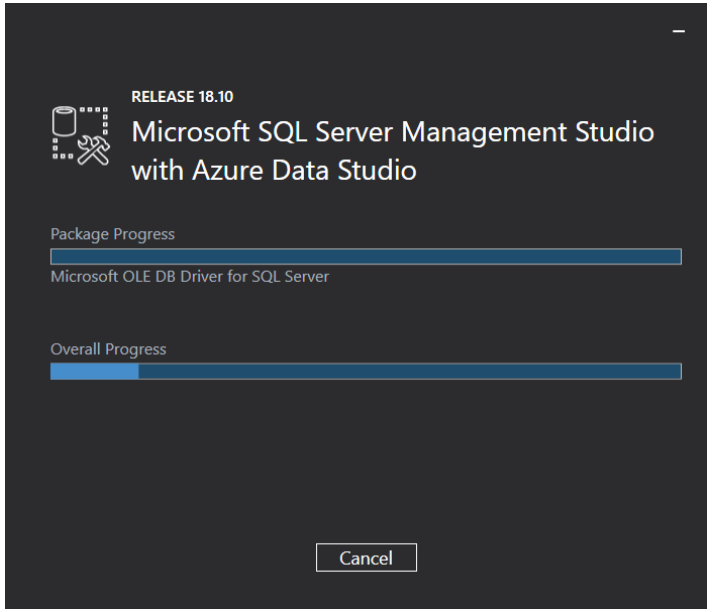


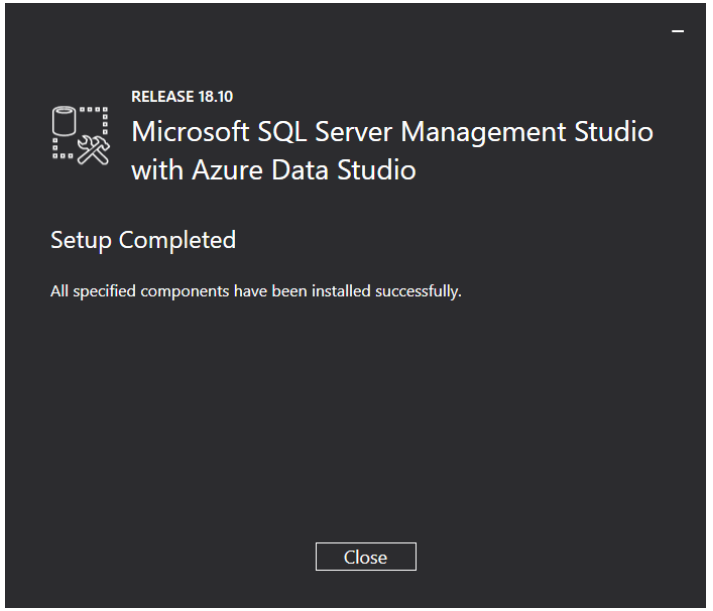
SQL Server Management Studio (SSMS)

- SQL Server Management Studio (SSMS) is an integrated environment for managing any SQL infrastructure, from SQL Server to Azure SQL Database. SSMS provides tools to configure, monitor, and administer instances of SQL Server and databases. Use SSMS to deploy, monitor, and upgrade the data-tier components used by your applications, and build queries and scripts.
- Use SSMS to query, design, and manage your databases and data warehouses, wherever they are - on your local computer, or in the cloud.

SSMS: Download & install

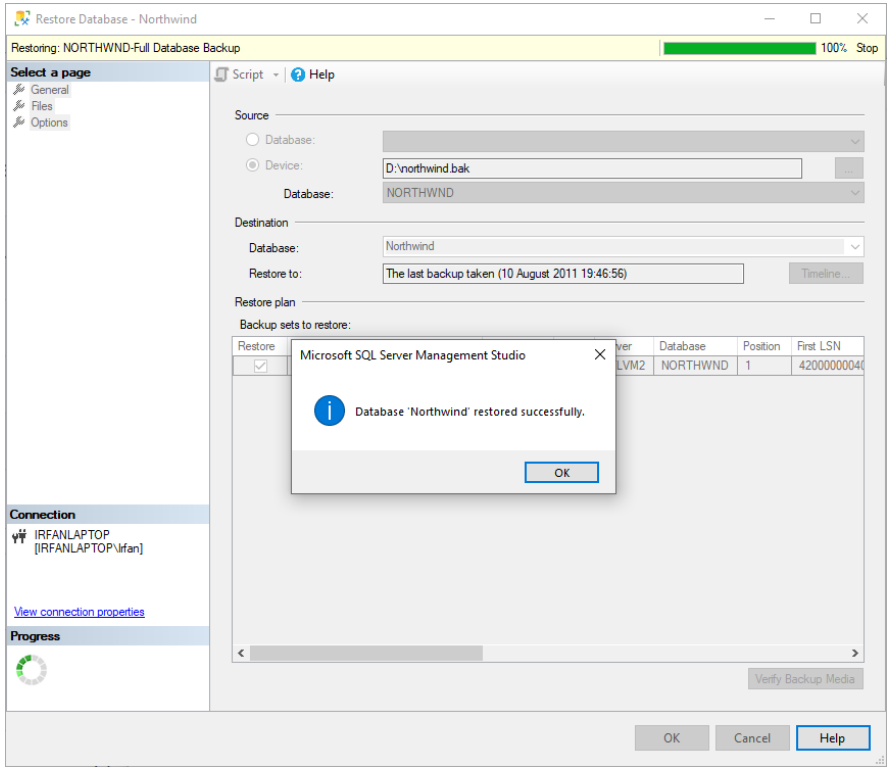






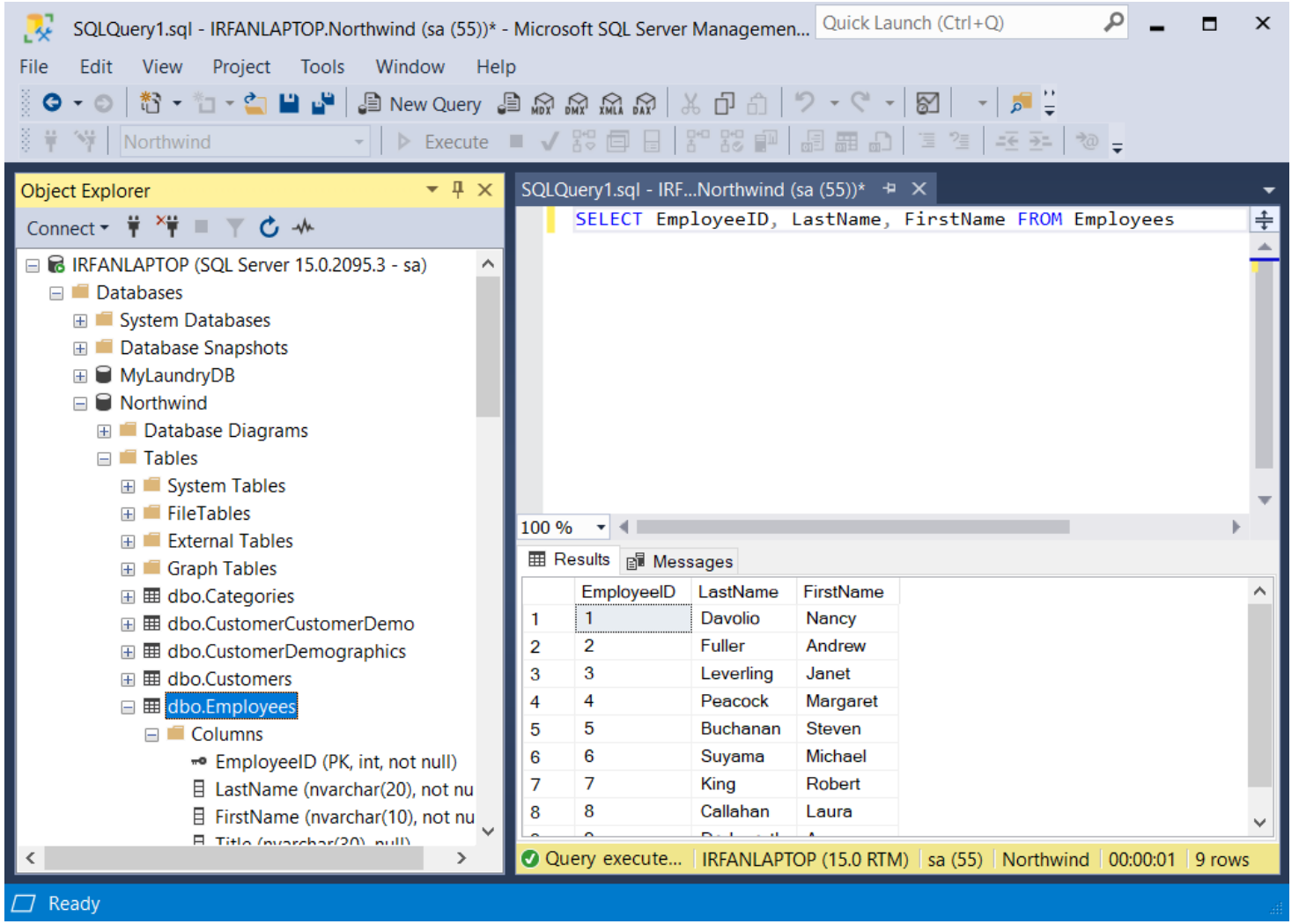
Example: Northwind database

- Microsoft SQL Server example: Northwind DB
- Download & restore Northwind DB, e.g., via SSMS



Northwind database

- Reading the table **Employees**, then print out all the rows based on `EmployeeID`, `LastName` and `FirstName`



Connection-oriented data access: Pattern

1. Declare the connection

```
try {
        1. Request connection to database
        2. Execute SQL statements
        3. Process result
        4. Release Resources
} catch (Exception) {
    Handle exception
} finally {
    try {
        5. Close connection
    }   catch (Exception) {
        { Handle exception }
    }
}
```

Example: EmployeesReader

```
C:\WINDOWS\system32\cmd.exe
| 1 | Davolio | Nancy
| 2 | Fuller | Andrew
| 3 | Leverling | Janet
| 4 | Peacock | Margaret
| 5 | Buchanan | Steven
| 6 | Suyama | Michael
| 7 | King | Robert
| 8 | Callahan | Laura
| 9 | Dodsworth | Anne
Press any key to continue . . .
```

Program.cs — EmployeesReader — EmployeesReader.Program — Main(

```
1  using System;
2  using System.Data;
3  using System.Data.OleDb;
4
5  namespace EmployeesReader {
       0 references
6      class Program {
7          // Establish connection
           0 references
8          static void Main(string[] args) {
9              string connStr = "provider=SQLOLEDB;data source=IrfanLaptop;"
10                 + "initial catalog=Northwind;user id=sa;password=123;"
11                 + "Trusted_Connection=true;";
12             IDbConnection con = null; // Declare connection object
13             try {
14                 // 1. Create connection object
15                 con = new OleDbConnection(connStr);
16                 con.Open(); // Open connection
17                 // 2. Execute command
18                 // --- Create SQL command
19                 IDbCommand cmd = con.CreateCommand();
20                 cmd.CommandText = "SELECT EmployeeID, LastName, " +
21                     "FirstName FROM Employees";
22                 // --- Execute SQL Command; result is an OleDbDataReader
23                 IDataReader reader = cmd.ExecuteReader();
24                 object[] dataRow = new object[reader.FieldCount];
25                 // 3. Process result
26                 while (reader.Read()) {
27                     int cols = reader.GetValues(dataRow);
28                     for (int i = 0; i < cols; i++) {
29                         Console.Write("| {0} ", dataRow[i]);
30                     }
31                     Console.WriteLine();
32                 }
33                 // 4. Release resources
34                 // --- Close reader
35                 reader.Close();
36             } catch (Exception e) {
37                 Console.WriteLine(e.Message);
38             } finally {
39                 try {
40                     if (con != null) {
41                         // 5. Close connection
42                         // --- Close con: the connection object
43                         con.Close();
44                     }
45                 } catch (Exception ex) {
46                     Console.WriteLine(ex.Message);
47                 }
48             }
49         }
50     }
51 }
```

Interface IDbConnection

- **ConnectionString** defines data base connection

```
string ConnectionString {get; set;}
```

- Open and close connection

```
void Close();
void Open();
```

- Properties of connection object

```
string  Database {get;}
int ConnectionTimeout {get;}
ConnectionState State {get;}
```

- Creates Command-Object

```
IDbCommand CreateCommand();
```

- Creates Transaction-Object

```
IDbTransaction BeginTransaction();
IDbTransaction BeginTransaction(IsolationLevel lvl);
```

<<interface>>
IDbConnection

```
//----- Properties
string ConnectionString
         {get; set;}
string  Database {get;}
int ConnectionTimeout {get;}
ConnectionState State {get;}

//----- Methods
IDbTransaction
BeginTransaction();
IDbTransaction BeginTransaction
         (IsolationLevel lvl);

IDbCommand CreateCommand();

void Close();
void Open();

...
```

```
public enum ConnectionState {
  Broken, Closed,
  Connecting, Executing,
  Fetching, Open
}
```

IDbConnection: Property ConnectionString

- Key-value-pairs separated by semicolon (;)
- Configuration of the connection
  - name of the provider
  - identification of data source
  - authentication of user
  - other database-specific settings

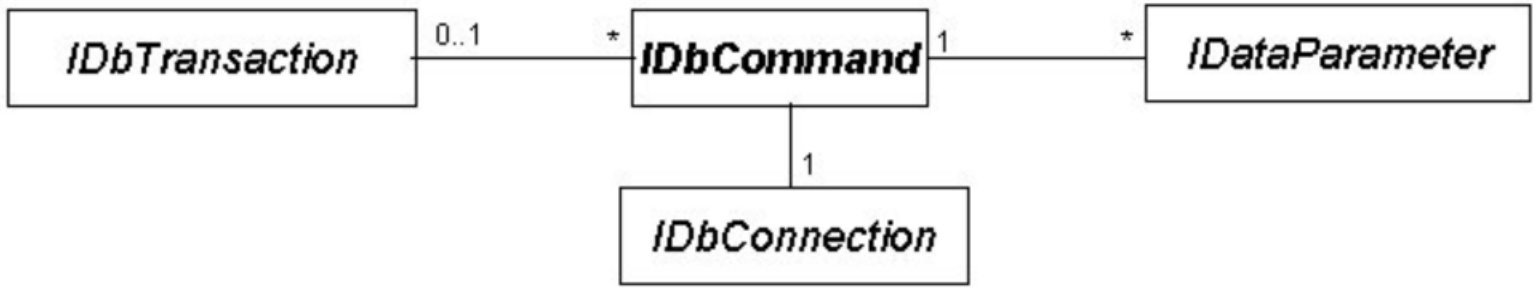
- e.g.: OLEDB:

```
"provider=SQLOLEDB; data source=127.0.0.1\NetSDK;
      initial catalog=Northwind; user id=sa; password=; "
"provider=Microsoft.Jet.OLEDB.4.0;data source=c:\bin\LocalAccess40.mdb;"
"provider=MSDAORA; data source=ORACLE8i7; user id=OLEDB; password=OLEDB;"
```

- e.g.: MS-SQL-Server:

```
"data source=(local)\\NetSDK; initial catalog=Northwind; user id=sa;
      pooling=false; Integrated Security=SSPI; connection timout=20;"
```

Command objects



- Command objects define SQL statements or stored procedures
- Executed for a connection
- May have parameters
- May belong to a transaction

Interface IDbCommand

- **CommandText** defines SQL statement or stored procedure

```
string CommandText {get; set;}
```

- **Connection** object

```
IDbConnection Connection {get; set;}
```

- **Type and timeout properties**

```
CommandType CommandType {get; set;}
int CommandTimeout {get; set;}
```

- **Creating and accessing parameters**

```
IDbDataParameter CreateParameter();
IDataParameterCollection Parameters {get;}
```

- **Execution of command**

```
IDataReader ExecuteReader();
IDataReader ExecuteReader(CommandBehavior b);
object ExecuteScalar();
int ExecuteNonQuery();
```

<<interface>>
IDbCommand

```
//----- Properties
string CommandText {get; set;}
CommandType CommandType
        {get; set;}
int CommandTimeout
        {get; set;}
IDbConnection Connection
        {get; set;}
IDataParameterCollection
        Parameters {get;}
IDbTransaction Transaction
        {get; set};
...

//----- Methods
IDbDataParameter
         CreateParameter();
IDataReader ExecuteReader();
IDataReader ExecuteReader
        (CommandBehavior b);
object ExecuteScalar();
int ExecuteNonQuery();
...
```

```
public enum CommandType {
    Text,
    StoredProcedure,
    TableDirect   }
```

ExecuteReader method

```
IDataReader ExecuteReader()
IDataReader ExecuteReader( CommandBehavior behavior );
```

```
public enum CommandBehavior {
    CloseConnection, Default, KeyInfo, SchemaOnly,
    SequentialAccess, SingleResult, SingleRow
}
```

- Executes the data base query specified in CommandText
- Result is an IDataReader object

Example:

```
cmd.CommandText =
            "SELECT EmployeeID, LastName, FirstName FROM Employees ";
IDataReader reader = cmd.ExecuteReader();
```

ExecuteNonQuery method

```
int ExecuteNonQuery();
```

- Executes the non-query operation specified in CommandText
  - UPDATE
  - INSERT
  - DELETE
  - CREATE TABLE
  - …
- Result is number of affected rows

Example:

```
cmd.CommandText = "UPDATE Empls SET City = 'Seattle' WHERE ID=8";
int affectedRows = cmd.ExecuteNonQuery();
```

ExecuteScalar method

```
object ExecuteScalar();
```

- Returns the value of the 1st column of the 1st row delivered by the database query
- CommandText typically is an aggregate function

Example:

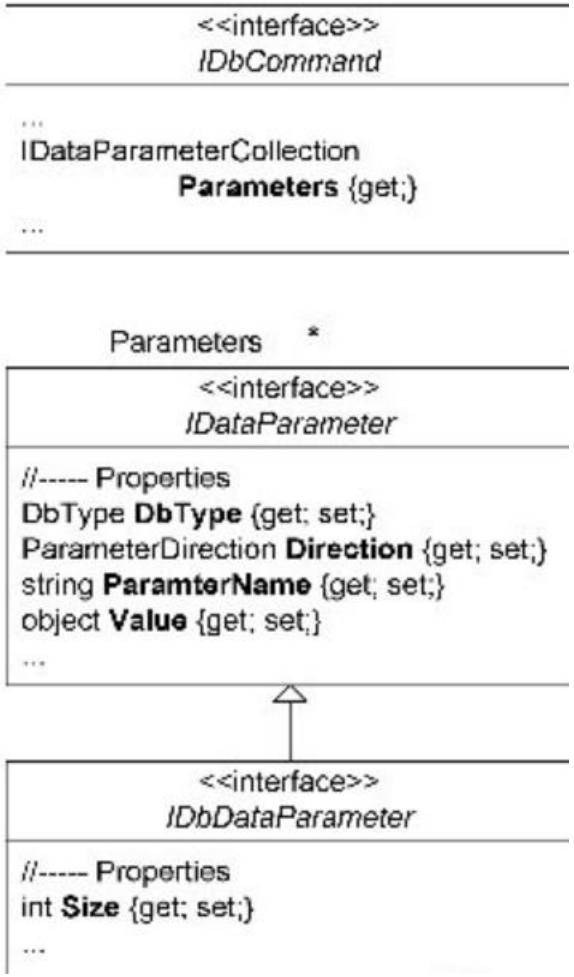
```
cmd.CommandText = " SELECT count(*) FROM Employees ";
int count = (int) cmd.ExecuteScalar();
```

Parameter

- **Command** objects allow for input and output parameters

```
IDataParameterCollection Parameters {get;}
```

- Parameter objects specify
  - **Name**: name of the parameter
  - **Value**: value of the parameter
  - **DbDataType**: data type of the parameter
  - **Direction**: direction of the parameter
    - Input
    - Output
    - InputOutput
    - ReturnValue



Working with parameters

1. Define SQL command with place holders

OLEDB: Identification of parameters by position (notation: "?")

```
OleDbCommand cmd = new OleDbCommand();
cmd.CommandText = "DELETE FROM Empls WHERE EmployeeID = ?";
```

SQL Server: Identification of parameters by name (notation: "@*name*")

```
SqlCommand cmd = new SqlCommand();
cmd.CommandText = "DELETE FROM Empls WHERE EmployeeID = @ID";
```

2. Create and add parameter

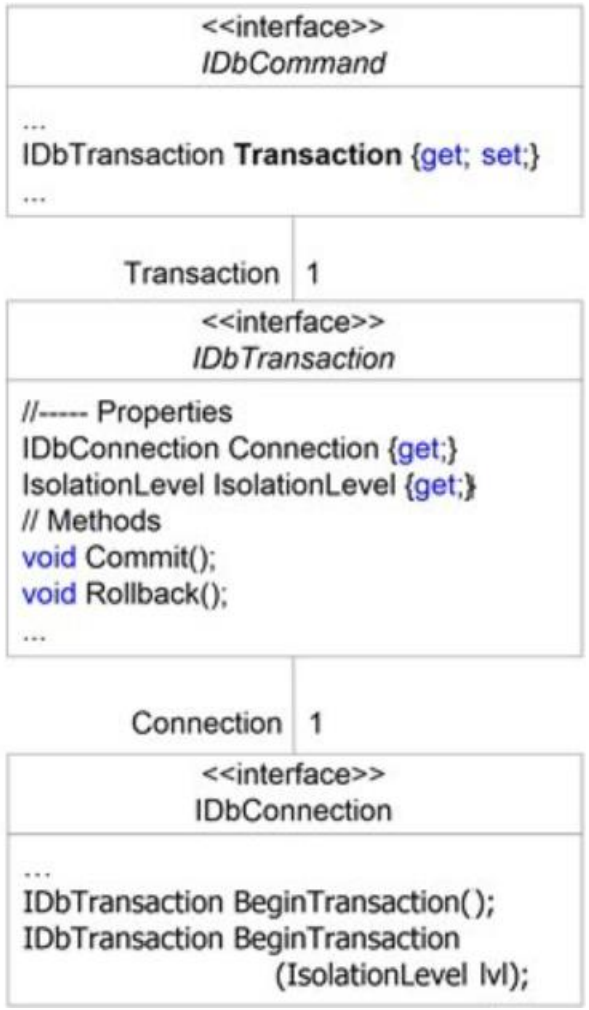
```
cmd.Parameters.Add( new OleDbParameter("@ID", OleDbType.BigInt));
```

3. Assign values and execute command

```
cmd.Parameters["@ID"].Value = 1234;
cmd.ExecuteNonQuery();
```

Transactions

- ADO.NET supports transactions
- Commands are assigned to transactions
- Execution of commands are
  - committed with **Commit**
  - aborted with **Rollback**



Working with transactions

1. Define connection and create Transaction object

```
SqlConnection con = new SqlConnection(connStr);
IDbTranaction trans = null;
try {
    con.Open();
    trans = con.BeginTransaction(IsolationLevel.ReadCommitted);
```

2. Create Command object, assign it to Transaction object, and execute it

```
    IDbCommand cmd1 = con.CreateCommand();
    cmd1.CommandText = "DELETE [OrderDetails] WHERE OrderId = 10258";
    cmd1.Transaction = trans;
    cmd1.ExecuteNonQuery();

    IDbCommand cmd2 = con.CreateCommand();
    cmd2.CommandText = "DELETE Orders WHERE OrderId = 10258";
    cmd2.Transaction = trans;
    cmd2.ExecuteNonQuery();
```

Working with transactions (continued)

3. Commit or abort transaction

```
        trans.Commit();
catch (Exception e) {
        if (trans != null)
                trans.Rollback();
} finally {
        try {
                con.Close();
        }
}
```

Isolation levels for transactions

- Define usage of read and write locks in transaction
- ADO.NET transactions allow different isolation levels

```
public enum IsolationLevel {
    ReadUncommitted, ReadCommitted, RepeatableRead, Serializable, ...
}
```

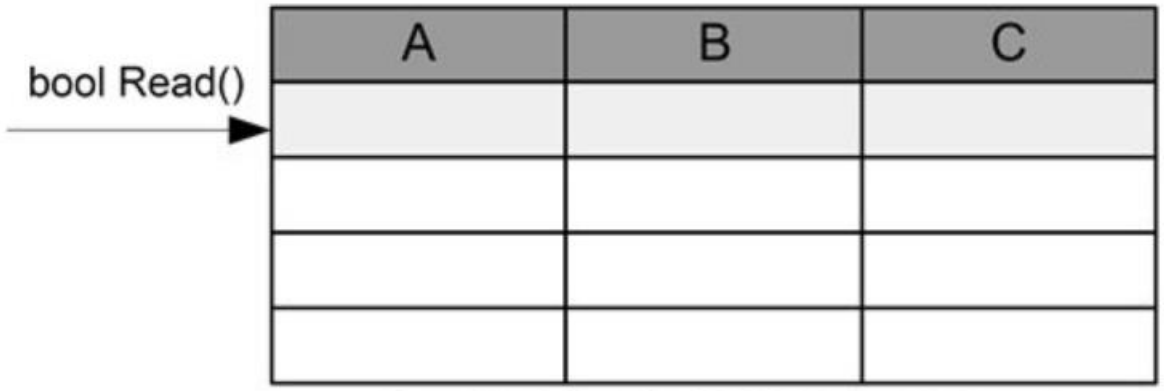
ReadUncommitted	• Allows reading of locked data • *Dirty reads* possible
ReadCommitted (Standard)	• Reading of locked data prohibited • No *dirty reads* but *phantom rows* can occur • *Non-repeatable reads*
RepeatableRead	• Same as *ReadCommitted* but *repeatable reads*
Seriazable	• Serialized access • *Phantom rows* cannot occur

DataReader

- ExecuteReader() returns DataReader object

```
IDataReader ExecuteReader()
IDataReader ExecuteReader( CommandBehavior behavior );
```

- DataReader allows sequential reading of result (row by row)



Result table of a SELECT statement

Interface IDataReader

- **Read** reads next row

```
bool Read();
```

- Access to column values using indexers

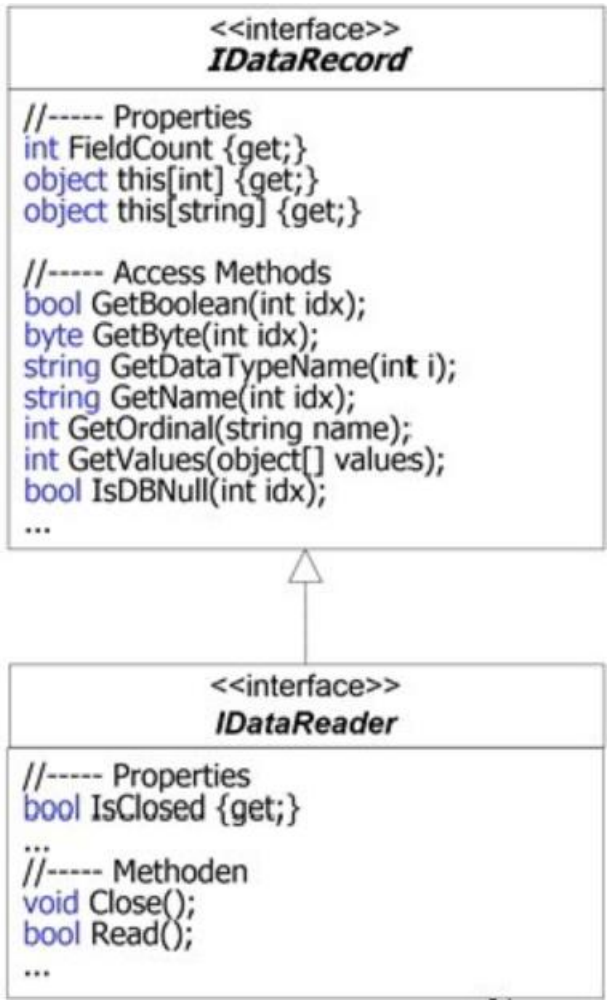
```
object this[int] {get;}
object this[string] {get;}
```

- Typed access to column values using access methods

```
bool GetBoolean(int idx);
byte GetByte(int idx);
...
```

- Getting meta information

```
string GetDataTypeName(int i);
string GetName(int idx);
int GetOrdinal(string name);
...
```



Working with IDataReader

- Create DataReader object and read rows

```
IDataReader reader = cmd.ExecuteReader();
while (reader.Read()) {
```

- Read column values into an array

```
    object[ ] dataRow = new object[reader.FieldCount];
    int cols = reader.GetValues(dataRow);
```

- Read column values using indexers

```
    object val0 = reader[0];
    object nameVal = reader["LastName"];
```

- Read column value using typed access method getString

```
    string firstName = reader.getString(2);
```

- Close DataReader

```
}
reader.Close();
```

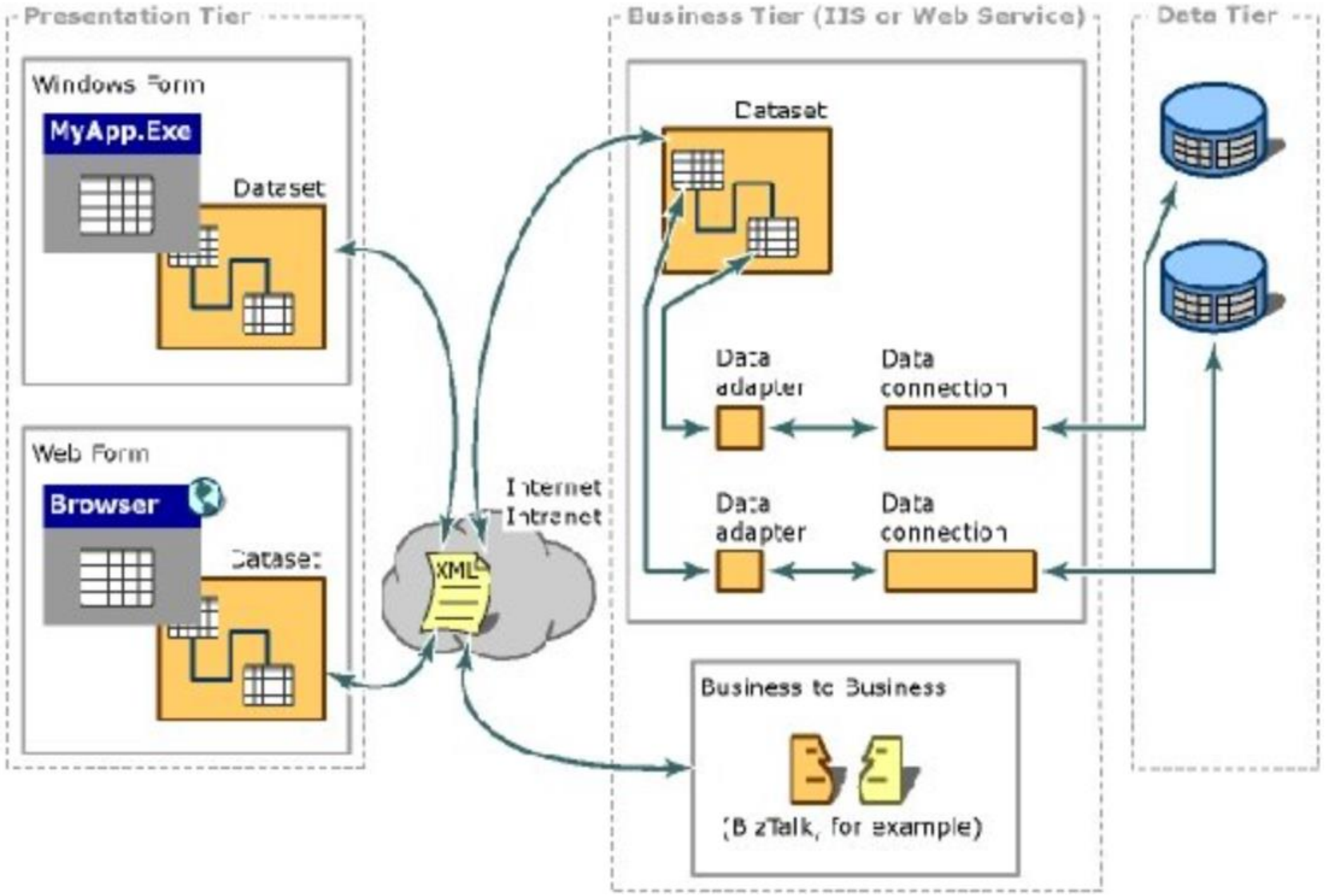
Connectionless-oriented

ADO.NET

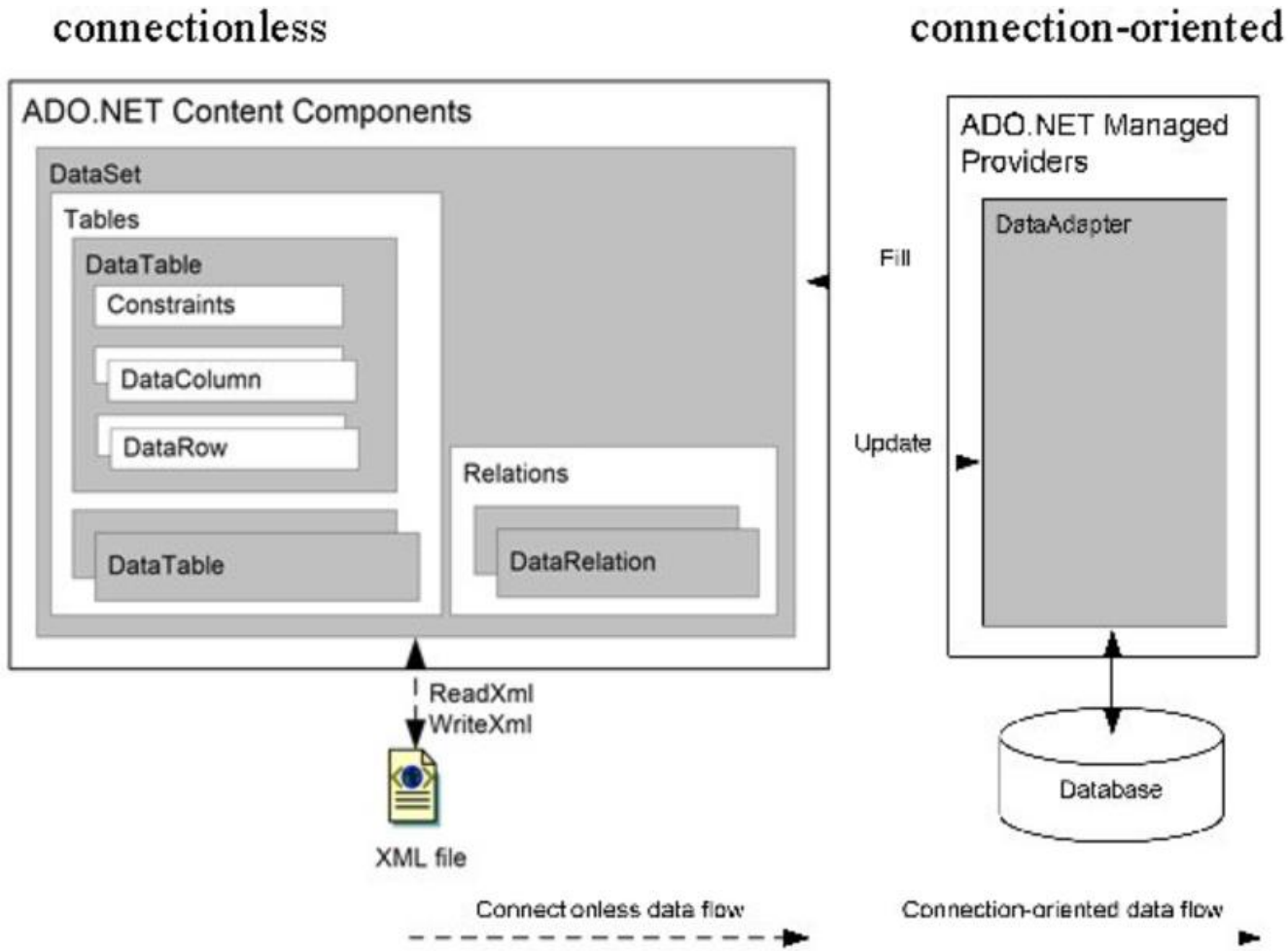
Motivation & idea

- Motivation
  - Many parallel, long lasting access operations
  - Connection-oriented data access too costly
- Idea
  - Caching data in main memory
    - “main memory database”
  - Only short connections for reading & updates
    - DataAdapter
  - Main memory database independent from data source
    - Conflicting changes are possible

Microsoft 3-tier: Architecture



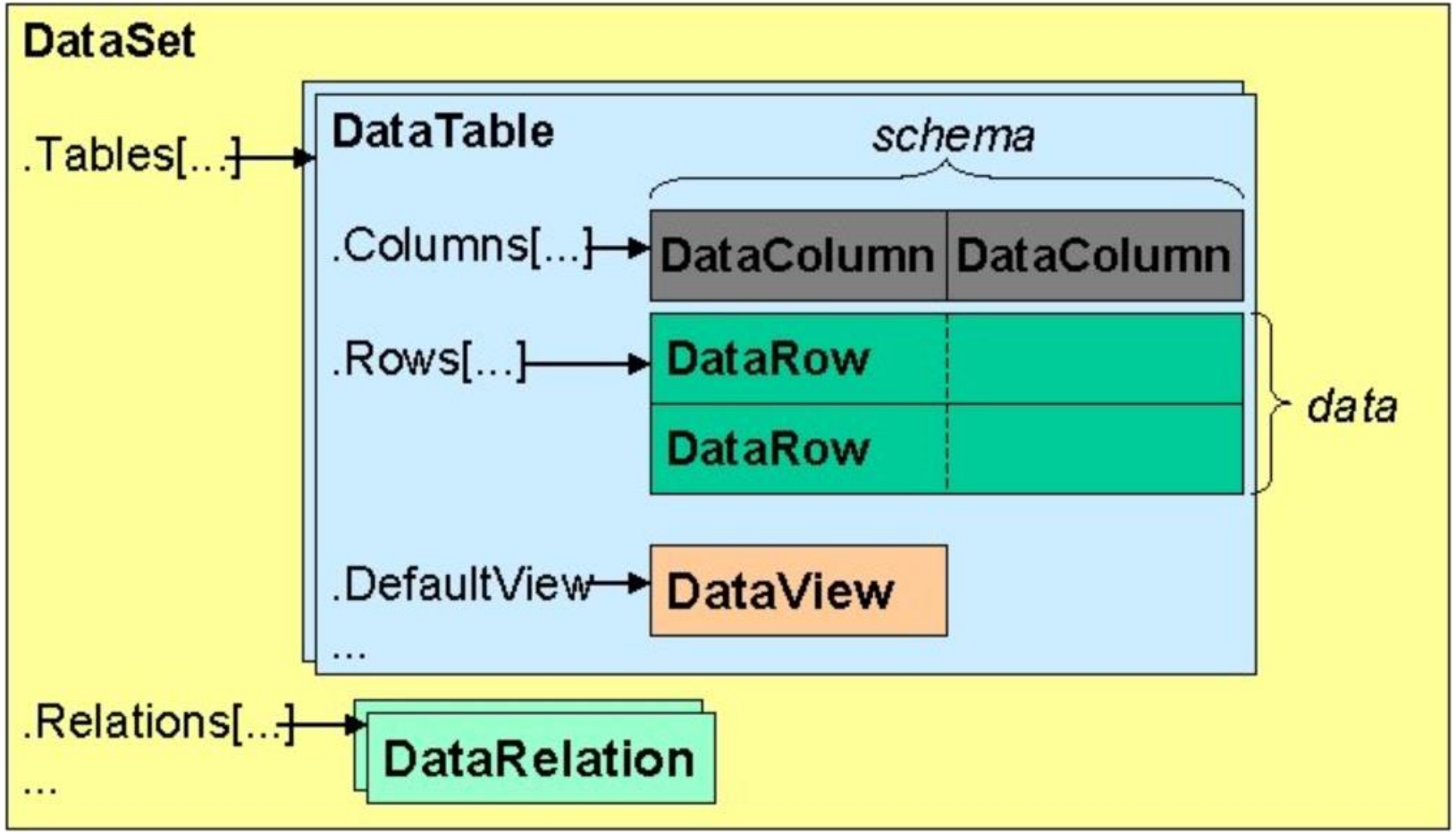
Connectionless data access: Architecture



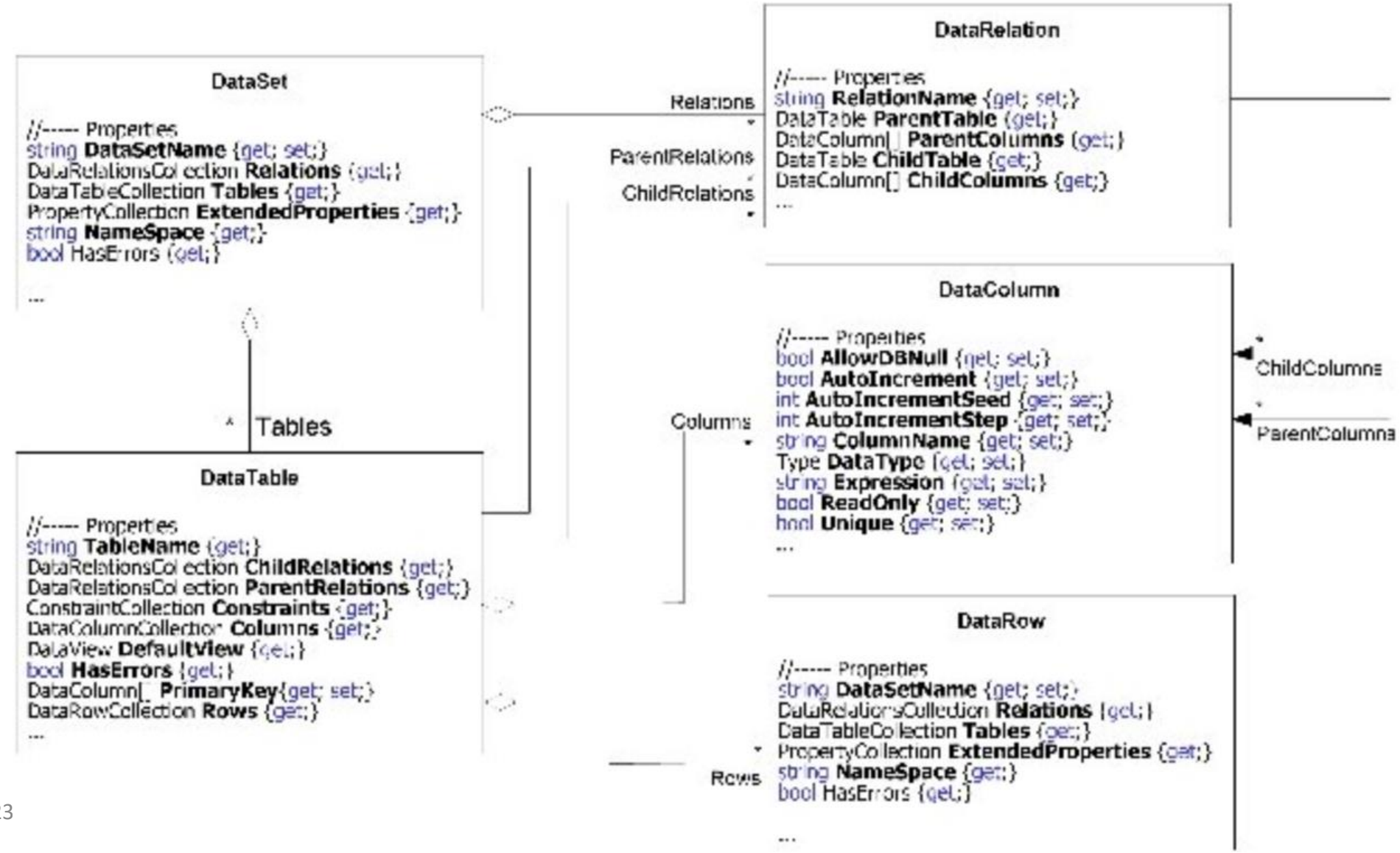
DataSet

- Main memory data base
  - relational structure
  - object oriented interface
- DataSet consists of
  - collection of DataTables
  - collection of DataRelations
- DataTables consists of
  - collection of DataTableColumns (= schema definition)
  - collection of DataTableRows (= data)
  - DefaultView (DataTableView, see later)
- DataRelations
  - associate two DataTable objects
  - define ParentTable and ParentColumns and ChildTable and ChildColumns

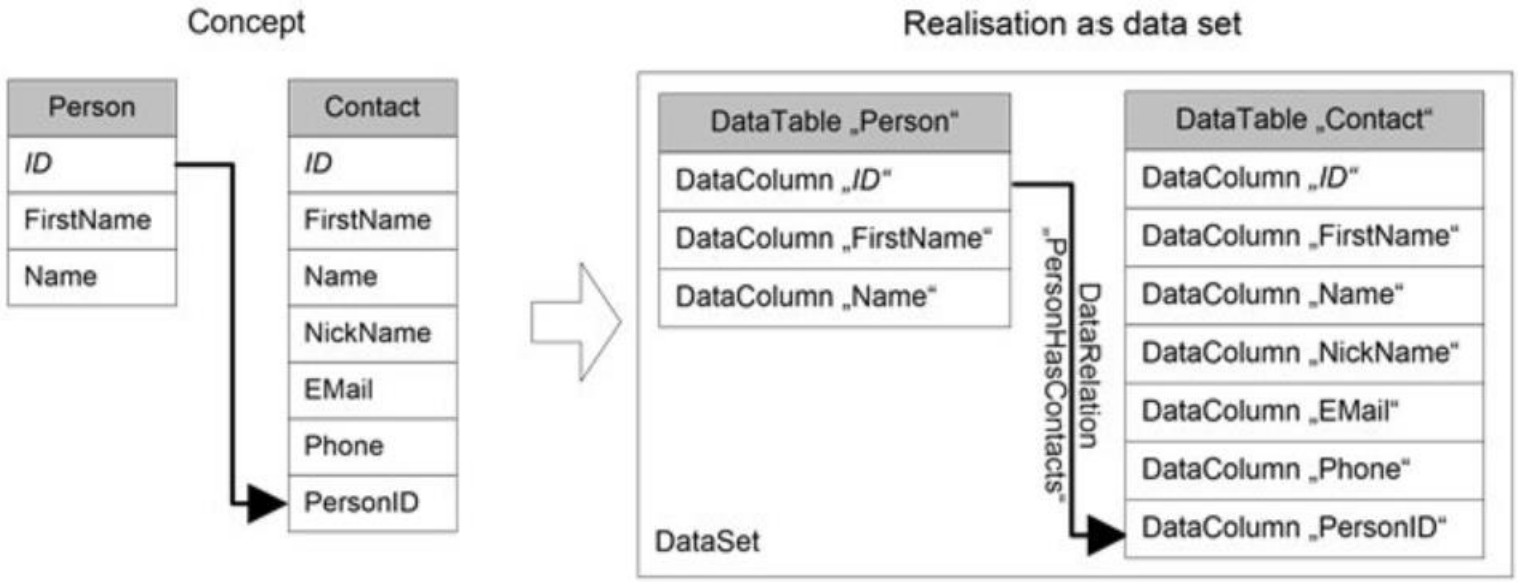
DataSet: Structure



DataSet: Class diagram



Example: Person contacts



Implementation steps:

- Define schema
- Define data
- Access data

Person Contacts: Define schema

- **Create DataSet and DataTable "Person"**

```
DataSet ds = new DataSet("PersonContacts");
DataTable personTable = new DataTable("Person");
```

- **Define column "ID" and set properties**

```
DataColumn col = new DataColumn();
col.DataType = typeof(System.Int64);
col.ColumnName = "ID";
col.ReadOnly = true;
col.Unique = true;                  // values must be unique
col.AutoIncrement = true;           // keys are assigned automatically
col.AutoIncrementSeed = -1;         // first key starts with -1
col.AutoIncrementStep = -1;         // next key = prev. key - 1
```

- **Add column to table and set as primary key**

```
personTable.Columns.Add(col);
personTable.PrimaryKey = new DataColumn[ ] { col };
```

Person Contacts: Define schema (continued)

- Define and add column "FirstName"

```
col = new DataColumn();
col.DataType = typeof(string);
col.ColumnName = "FirstName";
personTable.Columns.Add(col);
```

- Define and add column "Name"

```
col = new DataColumn();
col.DataType = typeof(string);
col.ColumnName = "Name";
personTable.Columns.Add(col);
```

- Add table to DataSet

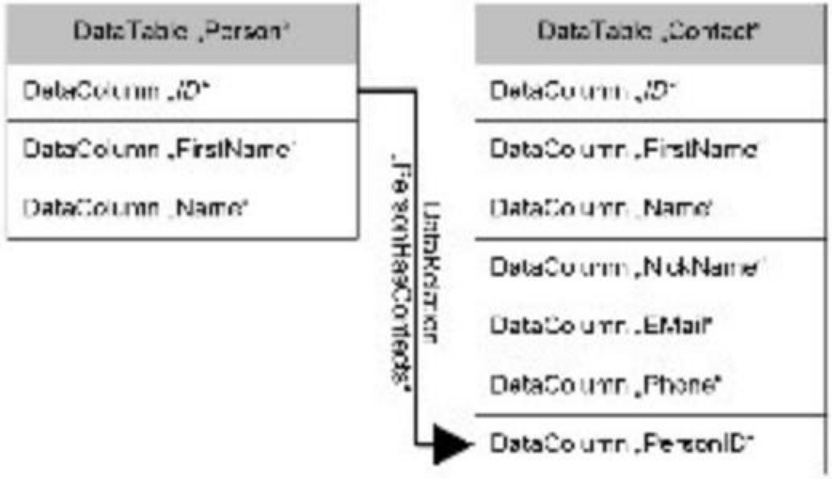
```
ds.Tables.Add(personTable);
```

- Create table "Contact" in similar way

```
DataTable contactTable = new DataTable("Contact");
...
ds.Tables.Add(contactTable);
```

Person Contacts: Define relation

- Create relation PersonHasContacts
- and add it to the DataSet




```
DataColumn parentCol = ds.Tables["Person"].Columns["ID"];
DataColumn childCol = ds.Tables["Contact"].Columns["PersonID"];

DataRelation rel = new DataRelation("PersonHasContacts", parentCol, childCol);
ds.Relations.Add(rel);
```

Person Contacts: Define DataRow

- Create new row and assign column values

```
DataRow personRow = personTable.NewRow();
personRow[1] = "Wolfgang";
personRow["Name"] = "Beer";
```

- Add row to table "Person"

```
personTable.Rows.Add(row);
```

- Create and add row to table "Contact"

```
DataRow contactRow = contactTable.NewRow ();
contactRow[0] = "Wolfgang";
...
contactRow["PersonID"] = (long)personRow["ID"]; // defines relation
contactTable.Rows.Add (row);
```

- Commit changes

```
ds.AcceptChanges();
```

Person Contacts: Access data

- Iterate over all persons of personTable and put out the names

```
foreach (DataRow person in personTable.Rows) {
    Console.WriteLine("Contacts of {0}:", person["Name"]);
```

- Access contacts through relation "PersonHasContacts" and print out contacts

```
foreach (DataRow contact in person.GetChildRows("PersonHasContacts")) {
    Console.WriteLine("{0}, {1}: {2}", contact[0], contact["Name"], contact["Phone"] );
}
```

DataSet: Change management

- DataSets maintain all changes
- Changes are accepted with `acceptChanges`
- Or discarded with `rejectChanges`

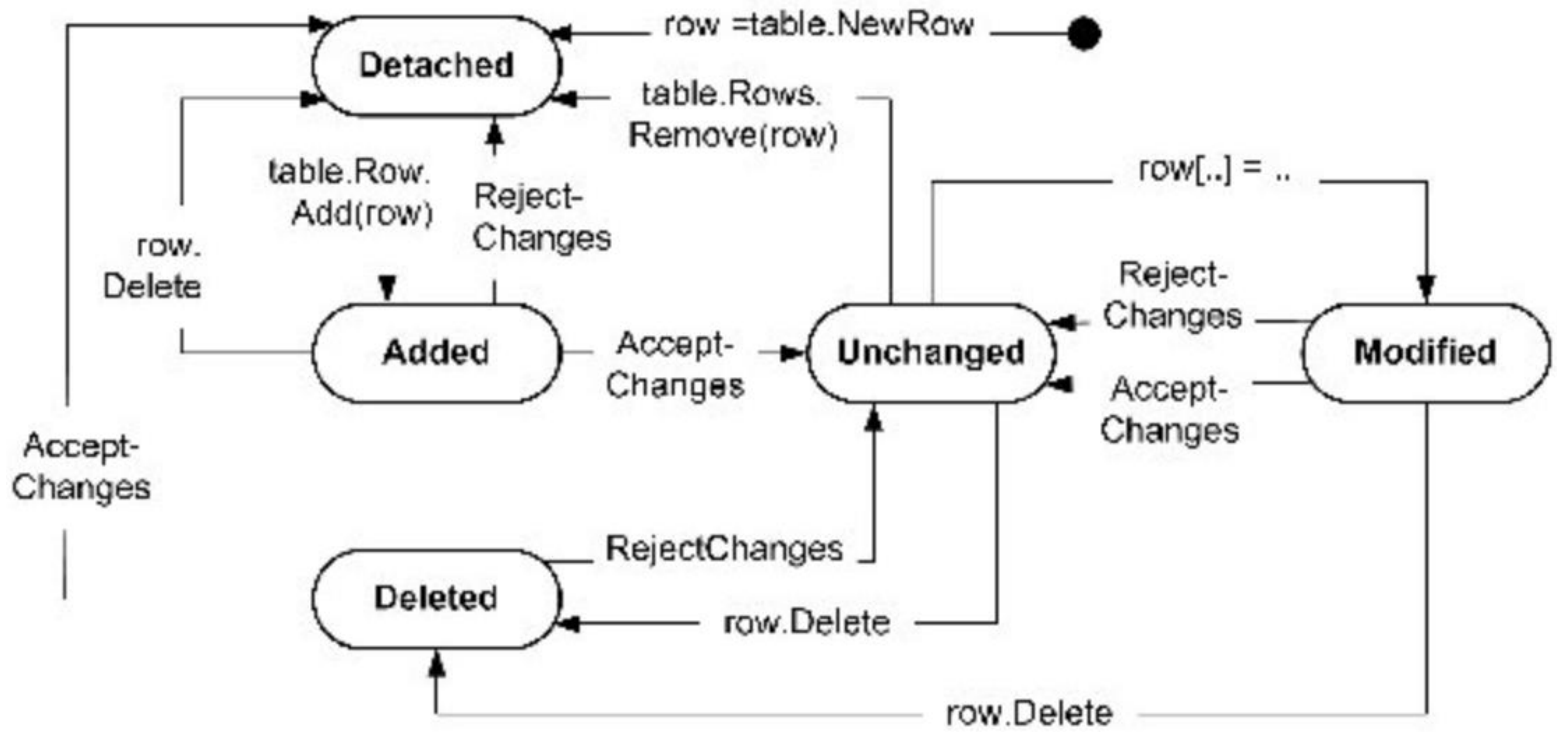
```
    ...
    if (ds.HasErrors) {
        ds.RejectChanges();
    } else {
        ds.AcceptChanges();
    }
}
```

State diagram of a DataRow object

- DataRow objects have different states

```
public DataRowState RowState {get;}
```

```
public enum DataRowState {
    Added, Deleted, Detached, Modified, Unchanged
}
```



DataRowVersion

DataSets store different versions of data row values:

```
public enum DataRowVersion {
    Current, Original, Proposed, Default
}
```

Current: current values

Original: original values

Proposed: proposed values (values which are currently processed)

Default: standard, based on DataRowState

DataRowState	Default
Added, Modified, Unchanged	Current
Deleted	Original
Detached	Proposed

Example:

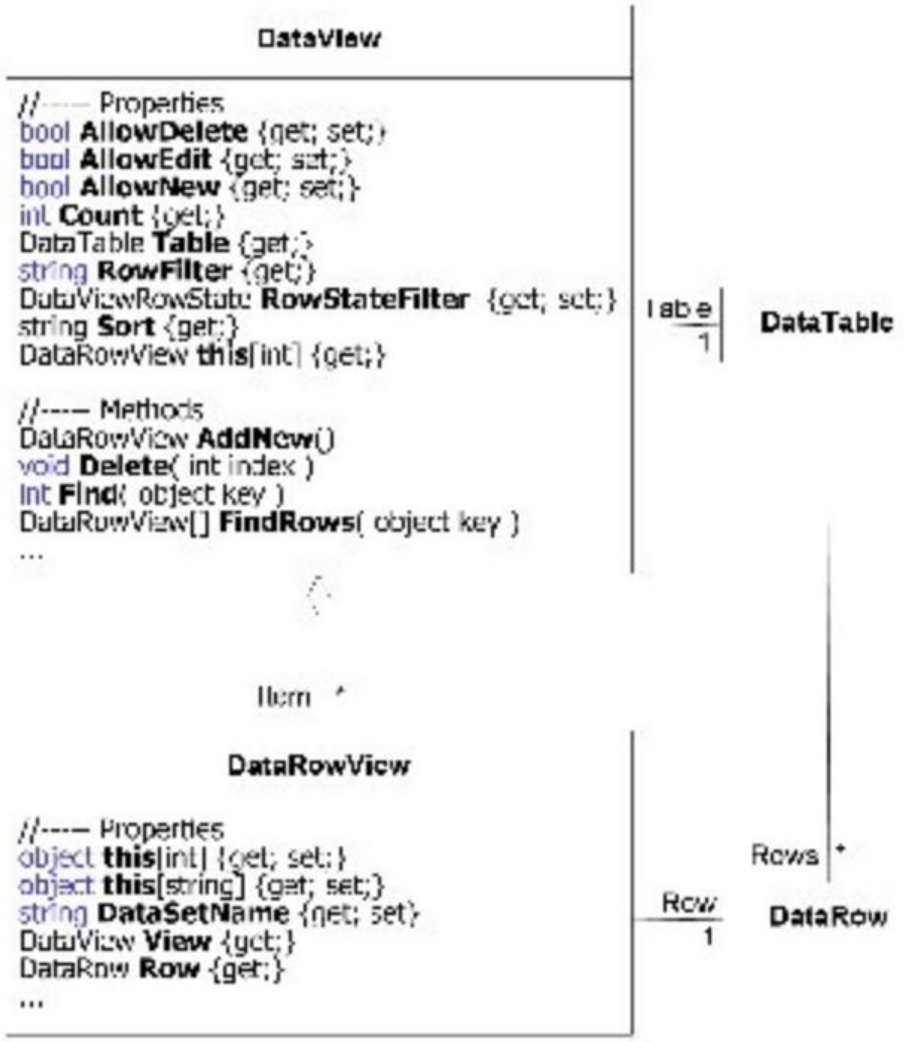
```
bool hasOriginal = personRow.HasVersion(DataRowVersion.Original);
if (hasOriginal) {
    string originalName = personRow["Name", DataRowVersion.Original];
}
```

Exception handling

- ADO.NET checks validity of operations on `DataSets`
- And throws `DataException`s

DataView

- DataViews support views of tables
  - RowFilter: Filtering based on filter expression
  - RowStateFilter: Filtering based on row states
  - Sort: Sorting based on columns
- DataView supports
  - changing data rows
  - fast search (based on sorted columns)
- DataView objects can be displayed by GUI elememts
  - e.g. DataGrid



DataView: Working with

- Create DataView object and set filter and sorting criteria

```
DataView a_kView = new DataView(personTable);
dataView.RowFilter = "FirstName <= 'K'";
dataView.RowStateFilter =
        DataViewRowState.Added | DataViewRowState.ModifiedCurrent;
dataView.Sort = "Name ASC";              // sort by Name in ascending order
```

- Display data in DataGrid

```
DataGrid grid = new DataGrid();
...
grid.DataSource = dataView;
```

- Fast search for row based on "Name" column

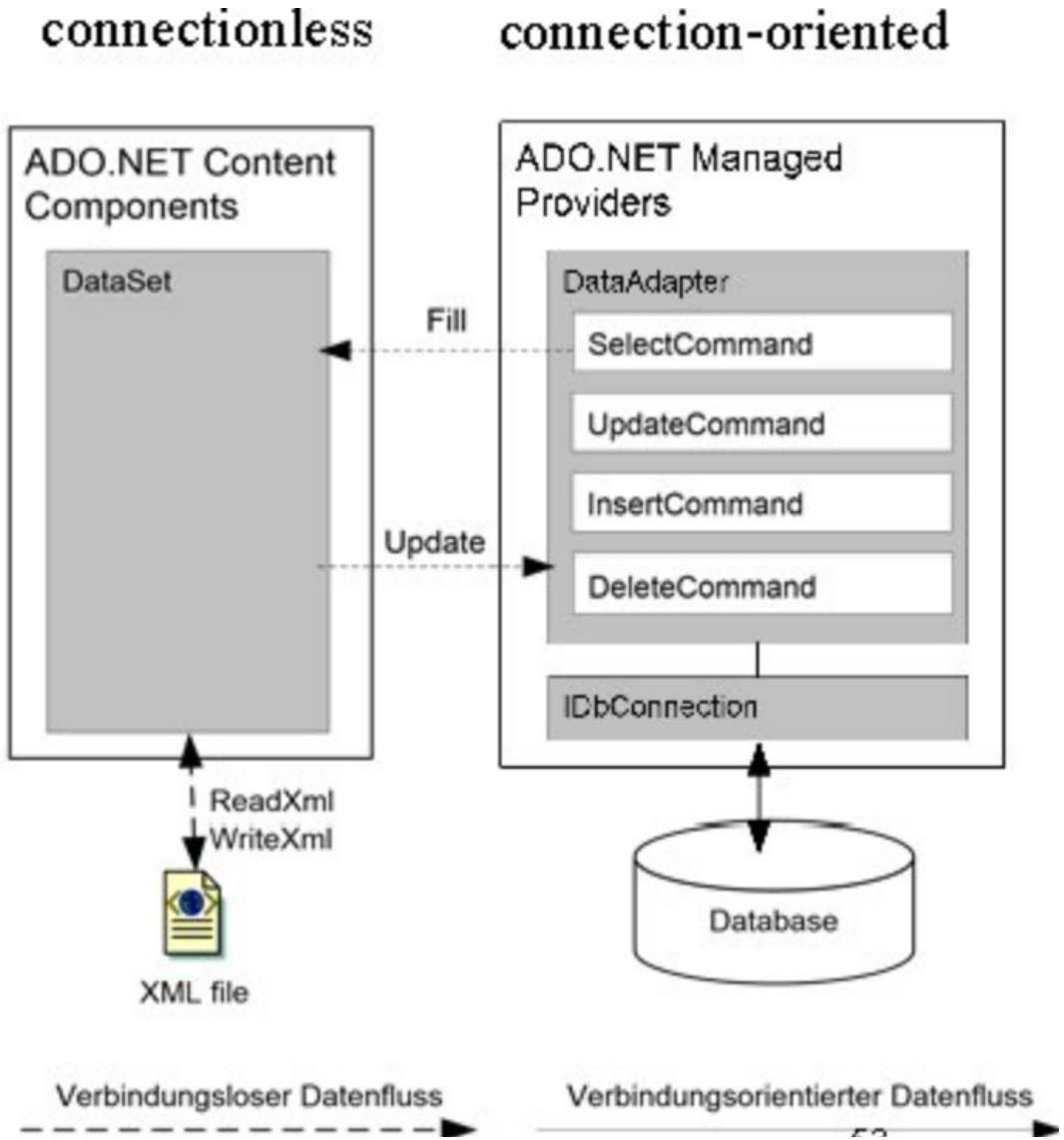
```
int i = a_kView.Find("Beer");
grid.Select(i);
```

Database access with DataAdapter

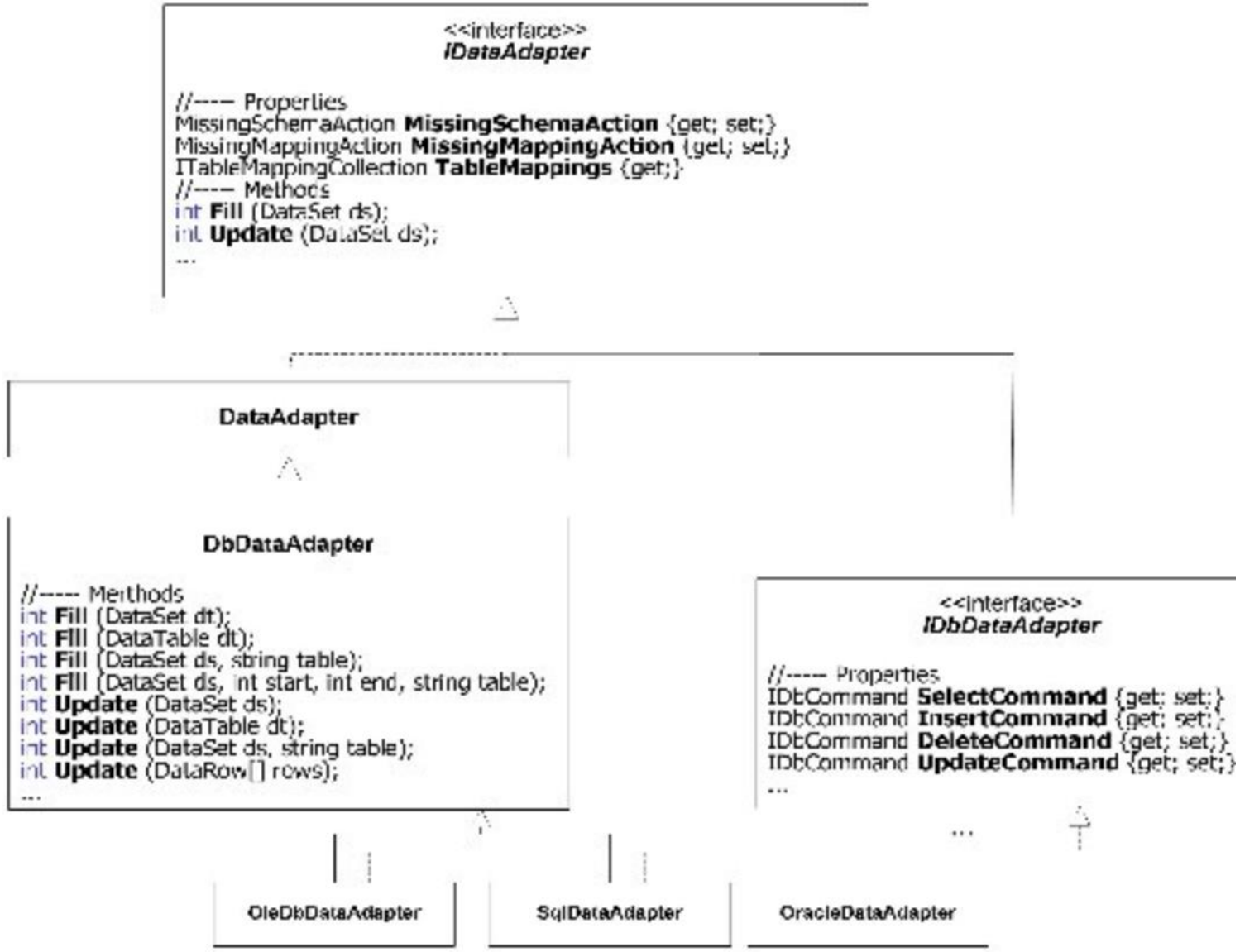
ADO.NET

Architecture

- DataAdapter for connection to date source
  - Fill: Filling the DataSet
  - Update: Writing back changes
- DataAdapters use Command objects
  - SelectCommand
  - InsertCommand
  - DeleteCommand
  - UpdateCommand



DataAdapter: Class diagram



DataAdapter: Loading data

- Create DataAdapter object and set SelectCommand

```
IDbDataAdapter adapter = new OleDbDataAdapter();
OleDbCommand cmd = new OleDbCommand();
cmd.Connection = new OleDbConnection ("provider=SQLOLEDB; ..." );
cmd.CommandText = "SELECT * FROM Person";
adapter.SelectCommand = cmd;
```

- Read data from data source and fill DataTable "Person"

```
adapter.Fill(ds, "Person");
```

- Accept or discard changes
- Delete DataAdapter object

```
if (ds.HasErrors) ds.RejectChanges();
else ds.AcceptChanges();
if (adapter is IDisposable) ((IDisposable)adapter).Dispose();
```

DataAdapter: Loading schema & data

- **Create DataAdapter object and set SelectCommand**

```
IDbDataAdapter adapter = new OleDbDataAdapter();
OleDbCommand cmd = new OleDbCommand();
cmd.Connection = new OleDbConnection ("provider=SQLOLEDB; ..." );
cmd.CommandText = "SELECT * FROM Person; SELECT * FROM Contact";
adapter.SelectCommand = cmd;
```

- **Define action for missing schema and mapping to tables**

```
adapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;
adapter.TableMappings.Add("Table", "Person");
adapter.TableMappings.Add("Table1", "Contact");
```

- **Read data from data source and fill DataTable "Person"**

```
adapter.Fill(ds);
```

- **Accept or discard changes; delete DataAdapter object**

```
if (ds.HasErrors) ds.RejectChanges();
else ds.AcceptChanges();
if (adapter is IDisposable) ((IDisposable)adapter).Dispose();
```

DataAdapter: Writing back changes

- Changes are written back with Update method
- Update-, Insert- and DeleteCommand define how changes are written
- CommandBuilder can create Update-, Insert- und DeleteCommand from SelectCommand automatically (in simple cases)
- Conflict management for updates:
  - comparison of data in DataTable and data source
  - in case of conflict DBConcurrencyException is thrown

DataAdapter: Writing back changes (cont'd)

- Create DataAdapter with SELECT expression

```
OleDbConnection con = new OleDbConnection ("provider=SQLOLEDB; ...");
adapter = new OleDbDataAdapter("SELECT * FROM Person", con);
```

- Create update commands using CommandBuilder

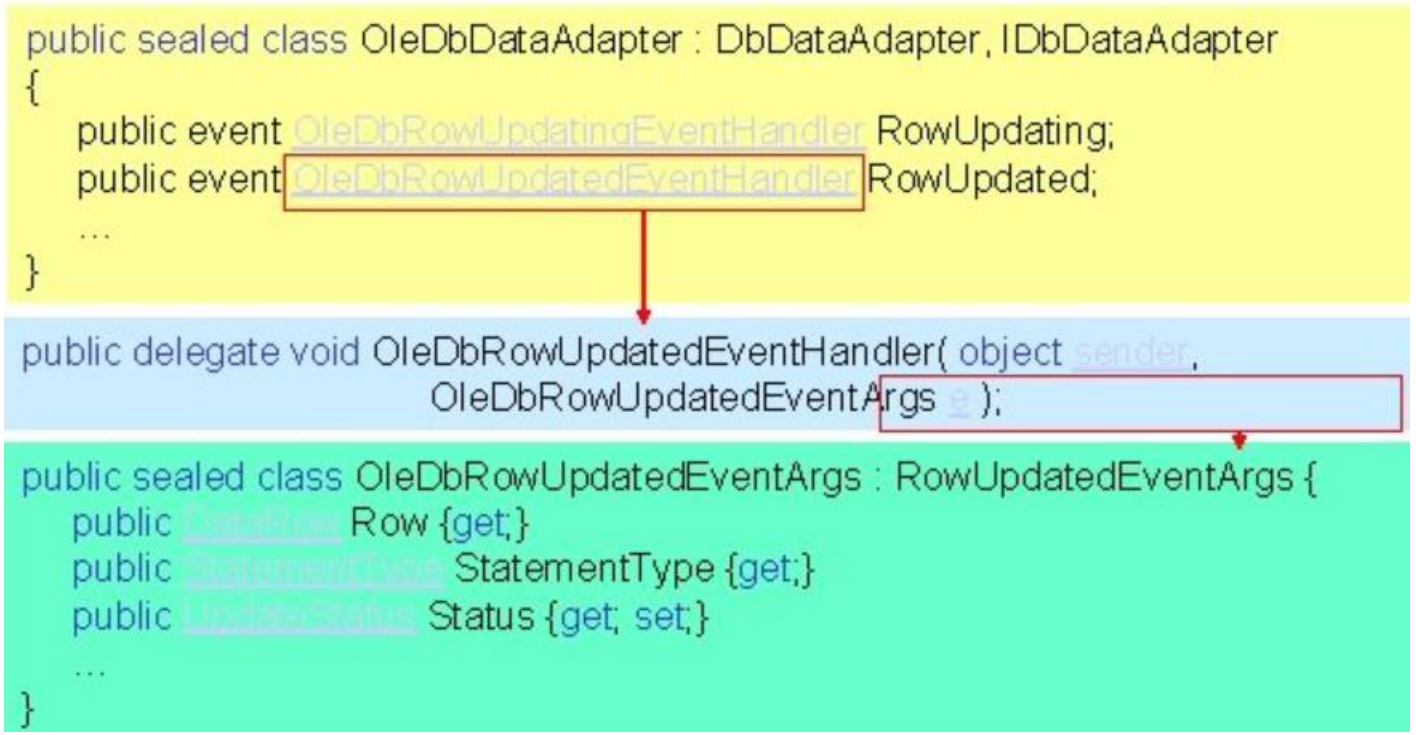
```
OleDbCommandBuilder cmdBuilder = new OleDbCommandBuilder(adapter);
```

- Call Update and handle conflicts

```
try {
    adapter.Update(ds, tableName);
} catch (DBConcurrencyException) {
    // Handle the error, e.g. by reloading the DataSet
}
adapter.Dispose();
```

DataAdapter: Event handling

- Two events signaled on updates for each data row
  - **OnRowUpdating**: just before updating the data source
  - **OnRowUpdated**: just after updating the data source



DataAdapter: Event handling example

- **Define handler methods**

```
private void onRowUpdating(object sender, OleDbRowUpdatedEventArgs args) {
    Console.WriteLn("Updating row for {0}", args.Row[1]);
    ...
}
```

```
private void onRowUpdated(object sender, OleDbRowUpdatedEventArgs args) {
    ...
}
```

- **Add delegates to events of DataAdapter**

```
OleDbDataAdapter adapter = new OleDbDataAdapter();
...
da.RowUpdating += new OleDbRowUpdatingEventHandler(this.OnRowUpdating);
da.RowUpdated += new OleDbRowUpdatingEventHandler(this.OnRowUpdated);
```

Integration with XML

ADO.NET

Integration DataSets & XML

- DataSets and XML are highly integrated
  - serializing DataSets as XML data
  - XML documents as data sources for DataSets
  - schemas for DataSets defined as XML schemas
  - *strongly typed* DataSets generated from XML schemas
  - access to DataSets using XML-DOM interface

- Integration of DataSets and XML used in distributed systems, e.g., web services
  - (see *Microsoft 3-Tier Architecture*)

Writing & reading XML data

- Methods for writing and reading XML data

```
public class DataSet : MarshalByValueComponent, IListSource,
                       ISupportInitialize, ISerializable {
    public void WriteXml( Stream stream );
    public void WriteXml( string fileName );
    public void WriteXml( TextWriter writer);
    public void WriteXml( XmlWriter writer );
    public void WriteXml( Stream stream, XmlWriteMode m );
    public void ReadXml ( Stream stream );
    public void ReadXml ( string fileName );
    public void ReadXml ( TextWriter writer);
    public void ReadXml ( XmlWriter writer );
    public void ReadXml ( Stream stream, XmlReadMode m );
    ...
}
```

```
public enum XmlWriteMode {DiffGram, IgnoreSchema, WriteSchema}
```

```
public enum XmlReadMode {
    Auto, DiffGram, IgnoreSchema, ReadSchema, InferSchema, Fragment }
```

Writing & reading XML data: Example

- Write data to XML file

```
ds.writeXML("personcontact.xml");
```

- Read data from XML
  - with XmlReadMode.Auto a schema is generated automatically

```
DataSet ds = new DataSet();
ds.readXML("personcontact.xml",
           XmlReadMode.Auto);
```

```
<?xml version="1.0" standalone="yes" ?>
<PersonContacts>
  <Person>
    <ID>1</ID>
    <FirstName>Wolfgang</FirstName>
    <Name>Beer</Name>
  </Person>
  <Person>
    <ID>2</ID>
    <FirstName>Dietrich</FirstName>
    <Name>Birngruber</Name>
  </Person>
  <Contact>
    <ID>1</ID>
    <FirstName>Dietrich</FirstName>
    <Name>Birngruber</Name>
    <NickName>Didi</NickName>
    <EMail>didi@dotnet.jku.at</EMail>
    <Phone>7133</Phone>
    <PersonID>2</PersonID>
  </Contact>
  <Contact>
    <ID>2</ID>
    <FirstName>Wolfgang</FirstName>
    <Name>Beer</Name>
    ...
    <PersonID>1</PersonID>
  </Contact>
</PersonContacts>
```

DataSet & XML schema

- **DataSets allow reading and writing XML schemas**
  - WriteXmlSchema: Writes XML schema
  - ReadXmlSchema: Reads XML schema and constructs DataSet
  - InferXmlSchema: Reads XML data and infers schema from the data

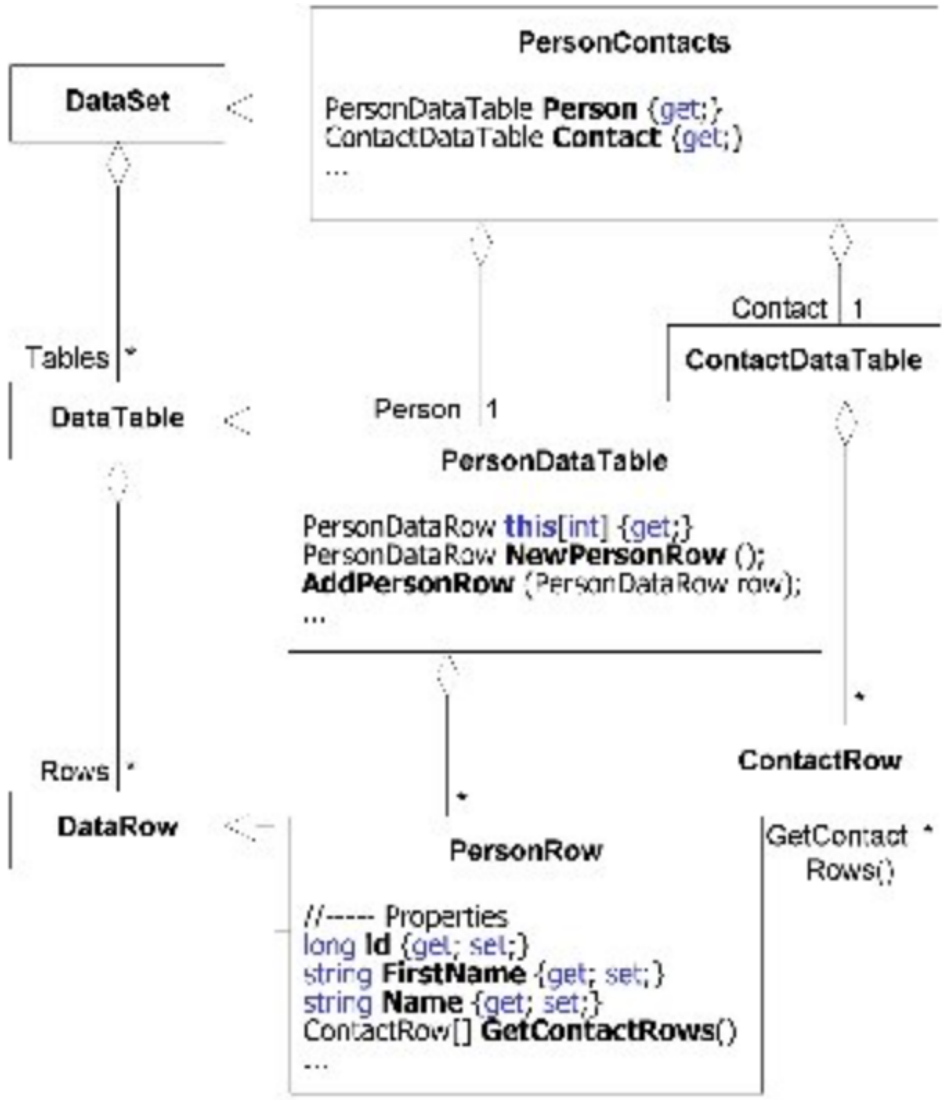
```
...
public void WriteXmlSchema ( Stream stream );
public void WriteXmlSchema ( string fileName );
public void WriteXmlSchema ( TextWriter writer);
public void WriteXmlSchema ( XmlWriter writer );

public void ReadXmlSchema  ( Stream stream );
public void ReadXmlSchema  ( string fileName );
public void ReadXmlSchema  ( TextWriter writer);
public void ReadXmlSchema  ( XmlWriter writer );

public void InferXmlSchema ( Stream stream, string[] namespaces );
public void InferXmlSchema ( string fileName, string[] namespaces );
public void InferXmlSchema ( TextWriter writer, string[] namespaces );
public void InferXmlSchema ( XmlWriter writer, string[] namespaces );
}
```

Typed DataSets

- *Typed DataSets* **provide typed data access**
- **Tool xsd.exe generates classes from XML schema**

```
> xsd.exe personcontact.xsd
/dataset
```

- **Classes define properties for typed access to rows, columns, and relations**



Typed DataSets: Example

- **Data access in conventional DataSet**

```
DataSet ds = new DataSet("PersonContacts");
DataTable personTable = new DataTable("Person");
...
ds.Tables.Add(personTable);
DataRow person = personTable.NewRow();
personTable.Rows.Add(person);
person["Name"] = "Beer";
...
person.GetChildRows("PersonHasContacts")[0]["Name"] = "Beer";
```

- **Data access in typed DataSet**

```
PersonContacts typedDS = new PersonContacts();
PersonTable personTable = typedDS.Person;
Person person = personTable.NewPersonRow();
personTable.AddPersonRow(person);
person.Name = "Beer";
...
person.GetContactRows()[0].Name = "Beer";
```

Access to DataSets using XML-DOM

- XmlDataDocument allows access over XML-DOM interface
- Synchronisation of changes in XmlDataDocument and DataSet

Example:

- Create XmlDataDocument object for DataSet objekt
- Change data in DataSet

```
XmlDataDocument xmlDoc = new XmlDataDocument(ds);
...
DataTable table = ds.Tables["Person"];
table.Rows.Find(3)["Name"] = "Changed Name!";
```

- Access changed data from XmlDataDocument object

```
XmlElement root = xmlDoc.DocumentElement;
XmlNode person = root.SelectSingleNode("descendant::Person[ID='3']");
Console.WriteLine("Access via XML: \n" + person.OuterXml);
```

Preview of ADO.NET 2.0

ADO.NET

ADO.NET 2.0

- Extended interfaces
- Tight coupling with MS SQL Server 9.0 („Yukon“)

New features are (many only available for MS SQL Server 9.0):

- bulk copy operation
- *Multiple Active Result Sets* (*MARS*)
- asynchronous execution of database operations
- batch processing of database updates
- paging through the result of a query
- *ObjectSpaces*

Bulk copy operation

- Inserting a large amount of data in one operation (only for MS SQL Server)
- Provided by class SqlBulkCopyOperation

Example

1. Define data source

```
SqlConnection sourceCon = new SqlConnection(conString); sourceCon.Open();
SqlCommand sourceCmd = new SqlCommand("SELECT * FROM Customers",sourceCon);
IDataReader sourceReader = sourceCmd.ExecuteReader();
```

2. Define target

```
SqlConnection targetCon = new SqlConnection(conString); targetCon.Open();
```

3. Copy data from source to target in one operation

```
SqlBulkCopyOperation bulkCmd = new SqlBulkCopyOperation(targetCon);
bulkCmd.DestinationTableName = "Copy_Customers";
bulkCmd.WriteDataReaderToServer(sourceReader);
```

Multiple Active Result Sets (MARS)

- So far only one DataReader for one connection allowed
- ADO.NET 2.0 allows several DataReaders in parallel

```
SqlConnection con = new SqlConnection(conStr);
con.Open();
SqlCommand custCmd = new SqlCommand("SELECT CustomerId, CompanyName " +
    "FROM Customers ORDER BY CustomerId", con);
SqlCommand ordCmd = new SqlCommand("SELECT CustomerId, OrderId, OrderDate " +
    "FROM Orders ORDER BY CustomerId, OrderDate", con);
SqlDataReader custRdr = custCmd.ExecuteReader();
SqlDataReader ordRdr = ordCmd.ExecuteReader();
string custID = null;
while (custRdr.Read()) { // use the first reader
    custID = custRdr.GetString(0);
    while (ordRdr.Read() && ordRdr.GetString(0) == custID ) {  // use the second reader
        ...
    }
}
...
```

Asynchronous operations

- So far only synchronous execution of commands
- ADO.NET 2.0 supports asynchronous execution mode (similar to asynchronous IO operations)

```
IAsyncResult   BeginExecuteReader (AsyncCallback callback)
IDataReader    EndExecuteReader (AsyncResult result)

IAsyncResult   BeginExecuteNonQuery (AsyncCallback callback)
int            EndExecuteNonQuery (IAsyncResult result)

IAsyncResult   BeginExecuteXmlReader (AsyncCallback callback)
IDataReader    EndExecuteXmlReader (IAsyncResult result)
```

Asynchronous operations: Example

```
...
public class Async {
    SqlCommand cmd; // command to be executed asynchronously
    public void CallCmdAsync() {
        SqlConnection con = new SqlConnection("Data Source=(local)\\NetSDK...");
        cmd = new SqlCommand("MyLongRunningStoredProc", con);
        cmd.CommandType = CommandType.StoredProcedure;
        con.Open();
        // execute the command asynchronously
        cmd.BeginExecuteNonQuery(new AsyncCallback(AsyncCmdEnded), null);
        ...
    }

    // this callback method is executed when the SQL command is finished
    public void AsyncCmdEnded(IAsyncResult result) {
        cmd.EndExecuteNonQuery(result);
        // optionally do some work based on results
        ...
    }
}
```

Batch processing of database updates

- So far rows are updated individually
- With ADO.NET 2.0 several rows can be updated in one batch (only available for MS SQL Server)
- **UpdateBatchSize** can be specified for DataAdapter

```
void UpdateCategories(DataSet ds, SqlConnection con) {
    // create an adapter with select and update commands
    SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM Categories", con);
    // the command builder creates the missing UPDATE, INSERT and DELETE commands
    SqlCommandBuilder cb = new SqlCommandBuilder(da);
    // set the batch size != 1
    da.UpdateBatchSize = 50;
    ...
    // execute the update in batch mode
    da.Update(ds.Tables["Categories"] );
}
```

Paging

- Operation ExecutePageReader allows accessing a subset of rows

```
ExecutePageReader(CommandBehavior b, int startRow, int pageSize)
```

➔ Very useful in combination with user interface controls (e.g. DataGrid)

ObjectSpaces

- ObjectSpaces allow mapping of objects and relational data
- Mapping defined in language *OPath* which (based on *XPath*)

Classes of ObjectSpaces

ObjectSpace: for communication with the data source

ObjectSources: list of connections to the data source

ObjectQuery: for reading objects with OPath

ObjectSet: stores the objets (similar to **DataSet**)

ObjectList and **ObjectHolder**: collections for delayed reading of objects

ObjectSpaces: Example

```
public class Customer { // mapped class
      public string Id; // primary key
      public string Name;
      public string Company;
      public string Phone;
}
public class ObjectSpaceSample {
      public static void Main() {
            // load the mapping and data source information and create the ObjectSpace.
            SqlConnection con = new SqlConnection("Data Source=(local)\\NetSDK; ...");
            ObjectSpace os = new ObjectSpace("map.xml", con);
            // query for objects
            ObjectQuery oQuery = new ObjectQuery(typeof(Customer),"Id >= 'T'", "");
            ObjectReader reader = os.GetObjectReader(oQuery);
            // print result
            foreach (Customer c in reader) {
                  Console.WriteLine(c.GetType() + ":");
                  Console.WriteLine("Id: " + c.Id);
                  Console.WriteLine("Name: " + c.Name);
                  Console.WriteLine("Phone: " + c.Phone);
            }
            reader.Close();
            con.Close();
      }
}
```

MyADO (1)

Program.cs

MyADO | MyADO.Program | Main(string[] args)

```
1   using System;
2   using System.Data;
3
4   namespace MyADO {
        0 references
5       class Program {
            0 references
6           static void Main(string[] args) {
7               // Create DataSet ds & DataTable "Person"
8               DataSet ds = new DataSet("PersonContacts");
9               DataTable personTable = new DataTable("Person");
10              // Define the column "ID" and its properties
11              DataColumn col = new DataColumn();
12              col.DataType = typeof(System.Int64);
13              col.ColumnName = "ID";
14              col.ReadOnly = true;
15              col.Unique = true;
16              col.AutoIncrement = true;
17              col.AutoIncrementSeed = 1;
18              col.AutoIncrementStep = 1;
19              // Add the column to the table and set the PK (Primary Key)
20              personTable.Columns.Add(col);
21              personTable.PrimaryKey = new DataColumn[] { col };
22              // Define the other column and add it to the table
23              col = new DataColumn();
24              col.DataType = typeof(string);
25              col.ColumnName = "FirstName";
26              personTable.Columns.Add(col);
```

MyADO (2)

```
27          // Define the other column and add it to the table
28          col = new DataColumn();
29          col.DataType = typeof(string);
30          col.ColumnName = "LastName";
31          personTable.Columns.Add(col);
32          // Add the table to the DataSet
33          ds.Tables.Add(personTable);
34
35          // Create DataTable "Contact"
36          DataTable contactTable = new DataTable("Contact");
37          // Define the column "ID" and its properties
38          col = new DataColumn();
39          col.DataType = typeof(System.Int64);
40          col.ColumnName = "ID";
41          col.ReadOnly = true;
42          col.Unique = true;
43          col.AutoIncrement = true;
44          col.AutoIncrementSeed = 1;
45          col.AutoIncrementStep = 1;
46          // Add the column to the table and set the PK (Primary Key)
47          contactTable.Columns.Add(col);
48          contactTable.PrimaryKey = new DataColumn[] { col };
49          // Define the other column and add it to the table
50          col = new DataColumn();
51          col.DataType = typeof(string);
52          col.ColumnName = "FirstName";
53          contactTable.Columns.Add(col);
```

MyADO (3)

```
54          // Define the other column and add it to the table
55          col = new DataColumn();
56          col.DataType = typeof(string);
57          col.ColumnName = "LastName";
58          contactTable.Columns.Add(col);
59          // Define the other column and add it to the table
60          col = new DataColumn();
61          col.DataType = typeof(string);
62          col.ColumnName = "NickName";
63          contactTable.Columns.Add(col);
64          // Define the other column and add it to the table
65          col = new DataColumn();
66          col.DataType = typeof(string);
67          col.ColumnName = "Email";
68          contactTable.Columns.Add(col);
69          // Define the other column and add it to the table
70          col = new DataColumn();
71          col.DataType = typeof(string);
72          col.ColumnName = "Phone";
73          contactTable.Columns.Add(col);
74          // Define the column "PersonID" and add it to the table
75          col = new DataColumn();
76          col.DataType = typeof(System.Int64);
77          col.ColumnName = "PersonID";
78          contactTable.Columns.Add(col);
79
80          // Add the table to the DataSet
81          ds.Tables.Add(contactTable);
82
```

MyADO (4)

```
83          // Define the relationship
84          DataColumn parentCol = ds.Tables["Person"].Columns["ID"];
85          DataColumn childCol = ds.Tables["Contact"].Columns["PersonID"];
86          DataRelation rel = new DataRelation("PersonHasContacts",
87              parentCol, childCol);
88          ds.Relations.Add(rel);
89
90          // Data operation: add a data to the table "Person"
91          // Define a new row & its values
92          DataRow personRow = personTable.NewRow();
93          //personRow["FirstName"] = "Wolfgang";
94          personRow[1] = "Wolfgang";
95          personRow["LastName"] = "Beer";
96          // Add the row to the table
97          personTable.Rows.Add(personRow);
98
99          // Data operation: add a data to the table "Contact"
100         // Define a new row & its values
101         DataRow contactRow = contactTable.NewRow();
102         contactRow["FirstName"] = "Wolfgang";
103         contactRow["LastName"] = "Beer";
104         contactRow["Phone"] = "08123456789";
105         contactRow["PersonID"] = (long)personRow["ID"]; // Defines a relation
106                                                         // Add the row to the table
107         contactTable.Rows.Add(contactRow);
```

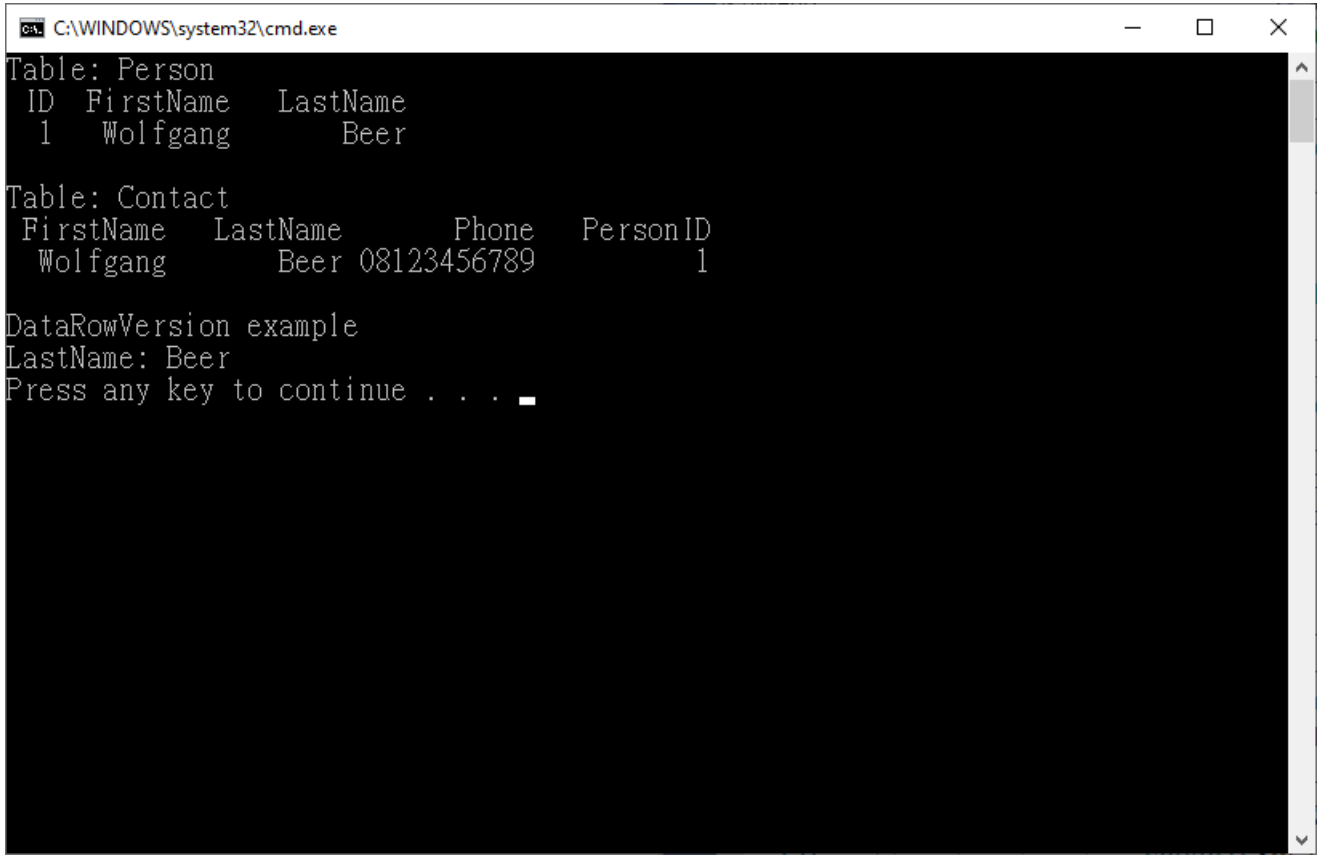
MyADO (5)

```
108         // Accept the changes permanently, if there's no errors
109         if (ds.HasErrors) {
110             ds.RejectChanges();
111         } else {
112             ds.AcceptChanges();
113         }
114
115         // Access the DataSet
116         Console.WriteLine("Table: Person");
117         Console.WriteLine(" ID  FirstName   LastName");
118         foreach (DataRow person in personTable.Rows) {
119             Console.WriteLine("{0,3} {1,10} {2,10}",
120                 person["ID"], person["FirstName"], person["LastName"]);
121         }
122         // Other method
123         Console.WriteLine("\nTable: Contact");
124         Console.WriteLine(" FirstName   LastName       Phone   PersonID");
125         foreach (DataRow contact in personRow.GetChildRows("PersonHasContacts")) {
126             Console.WriteLine("{0,10} {1,10} {2,10} {3,10}",
127                 contact["FirstName"], contact["LastName"],
128                 contact["Phone"], contact["PersonID"]);
129         }
130
```

MyADO (6)

```
131             // DataRowVersion example
132             Console.WriteLine("\nDataRowVersion example");
133             bool hasOriginal =
134                 personRow.HasVersion(DataRowVersion.Original);
135             if (hasOriginal) {
136                 string originalName = (string) personRow["LastName", DataRowVersion.Original];
137                 Console.WriteLine("LastName: " + originalName);
138             }
139         }
140     }
141 }
```

MyADO: Output



MyADO2 (1)

MyADO2.aspx

```
1  <%@ Page Language="C#" AutoEventWireup="true" CodeFile="MyADO2.aspx.cs" Inherits="MyADO2" %>
2
3  <!DOCTYPE html>
4
5  <html xmlns="http://www.w3.org/1999/xhtml">
6  <head runat="server">
7      <title>My ADO2</title>
8  </head>
9  <body>
10     <form id="form1" runat="server">
11         <div>
12             <asp:GridView ID="GridView1" runat="server"
13                 CellPadding="3"
14                 AutoGenerateColumns="False">
15             <Columns>
16                 <asp:BoundField DataField="ID" HeaderText="ID" />
17                 <asp:BoundField DataField="FirstName" HeaderText="First Name" />
18                 <asp:BoundField DataField="LastName" HeaderText="Last Name" />
19             </Columns>
20             </asp:GridView>
21             <br />
22             <asp:Button ID="Button1" runat="server" OnClick="Button1_Click" Text="Button" />
23         </div>
24     </form>
25 </body>
26 </html>
```

100 % No issues found Ln: 27 Ch: 1

ID	First Name	Last Name
Databound	Databound	Databound
Databound	Databound	Databound
Databound	Databound	Databound
Databound	Databound	Databound
Databound	Databound	Databound

Button

MyADO2 (2)

MyADO2.aspx.cs | 2_MyADO2.aspx | MyADO2 | Profile

```
using System;
using System.Data;

2 references
public partial class MyADO2 : System.Web.UI.Page {
    0 references
    protected void Button1_Click(object sender, EventArgs e) {
        // Create DataSet ds & DataTable "Person"
        DataSet ds = new DataSet("PersonContacts");
        DataTable personTable = new DataTable("Person");
        // Define the column "ID" and its properties
        DataColumn col = new DataColumn();
        col.DataType = typeof(System.Int64);
        col.ColumnName = "ID";
        col.ReadOnly = true;
        col.Unique = true;
        col.AutoIncrement = true;
        col.AutoIncrementSeed = 1;
        col.AutoIncrementStep = 1;
        // Add the column to the table and set the PK (Primary Key)
        personTable.Columns.Add(col);
        personTable.PrimaryKey = new DataColumn[] { col };
        // Define the other column and add it to the table
        col = new DataColumn();
        col.DataType = typeof(string);
        col.ColumnName = "FirstName";
        personTable.Columns.Add(col);
        // Define the other column and add it to the table
        col = new DataColumn();
        col.DataType = typeof(string);
        col.ColumnName = "LastName";
        personTable.Columns.Add(col);
```

MyADO2 (3)

```
31          // Add the table to the DataSet
32          ds.Tables.Add(personTable);
33
34          // Create DataTable "Contact"
35          DataTable contactTable = new DataTable("Contact");
36          // Define the column "ID" and its properties
37          col = new DataColumn();
38          col.DataType = typeof(System.Int64);
39          col.ColumnName = "ID";
40          col.ReadOnly = true;
41          col.Unique = true;
42          col.AutoIncrement = true;
43          col.AutoIncrementSeed = -1;
44          col.AutoIncrementStep = -1;
45          // Add the column to the table and set the PK (Primary Key)
46          contactTable.Columns.Add(col);
47          contactTable.PrimaryKey = new DataColumn[] { col };
48          // Define the other column and add it to the table
49          col = new DataColumn();
50          col.DataType = typeof(string);
51          col.ColumnName = "FirstName";
52          contactTable.Columns.Add(col);
53          // Define the other column and add it to the table
54          col = new DataColumn();
55          col.DataType = typeof(string);
56          col.ColumnName = "LastName";
57          contactTable.Columns.Add(col);
```

MyADO2 (4)

```
58          // Define the other column and add it to the table
59          col = new DataColumn();
60          col.DataType = typeof(string);
61          col.ColumnName = "NickName";
62          contactTable.Columns.Add(col);
63          // Define the other column and add it to the table
64          col = new DataColumn();
65          col.DataType = typeof(string);
66          col.ColumnName = "Email";
67          contactTable.Columns.Add(col);
68          // Define the other column and add it to the table
69          col = new DataColumn();
70          col.DataType = typeof(string);
71          col.ColumnName = "Phone";
72          contactTable.Columns.Add(col);
73          // Define the column "PersonID" and add it to the table
74          col = new DataColumn();
75          col.DataType = typeof(System.Int64);
76          col.ColumnName = "PersonID";
77          contactTable.Columns.Add(col);
78
79          // Add the table to the DataSet
80          ds.Tables.Add(contactTable);
81
82          // Define the relationship
83          DataColumn parentCol = ds.Tables["Person"].Columns["ID"];
84          DataColumn childCol = ds.Tables["Contact"].Columns["PersonID"];
85          DataRelation rel = new DataRelation("PersonHasContacts",
86              parentCol, childCol);
87          ds.Relations.Add(rel);
```

MyADO2 (5)

```
88
89          // 1
90          // Data operation: add a data to the table "Person"
91          // Define a new row & its values
92          DataRow personRow = personTable.NewRow();
93          personRow[1] = "Wolfgang";
94          personRow["LastName"] = "Beer";
95          // Add the row to the table
96          personTable.Rows.Add(personRow);
97          // Data operation: add a data to the table "Contact"
98          // Define a new row & its values
99          DataRow contactRow = contactTable.NewRow();
100         contactRow["FirstName"] = "Wolfgang";
101         contactRow["LastName"] = "Beer";
102         contactRow["Phone"] = "08123456789";
103         contactRow["PersonID"] = (long)personRow["ID"]; // Defines a relation
104         // Add the row to the table
105         contactTable.Rows.Add(contactRow);
106         // 2
107         // Add a data to the table "Person"
108         // Define a new row & its values
109         personRow = personTable.NewRow();
110         personRow[1] = "Maria";
111         personRow["LastName"] = "Sharapova";
112         // Add the row to the table
113         personTable.Rows.Add(personRow);
114         // Add a data to the table "Contact"
115         // Define a new row & its values
116         contactRow = contactTable.NewRow();
117         contactRow["FirstName"] = "Maria";
118         contactRow["LastName"] = "Sharapova";
119         contactRow["Phone"] = "08555555555";
120         contactRow["PersonID"] = (long)personRow["ID"];
121         contactTable.Rows.Add(contactRow);
```

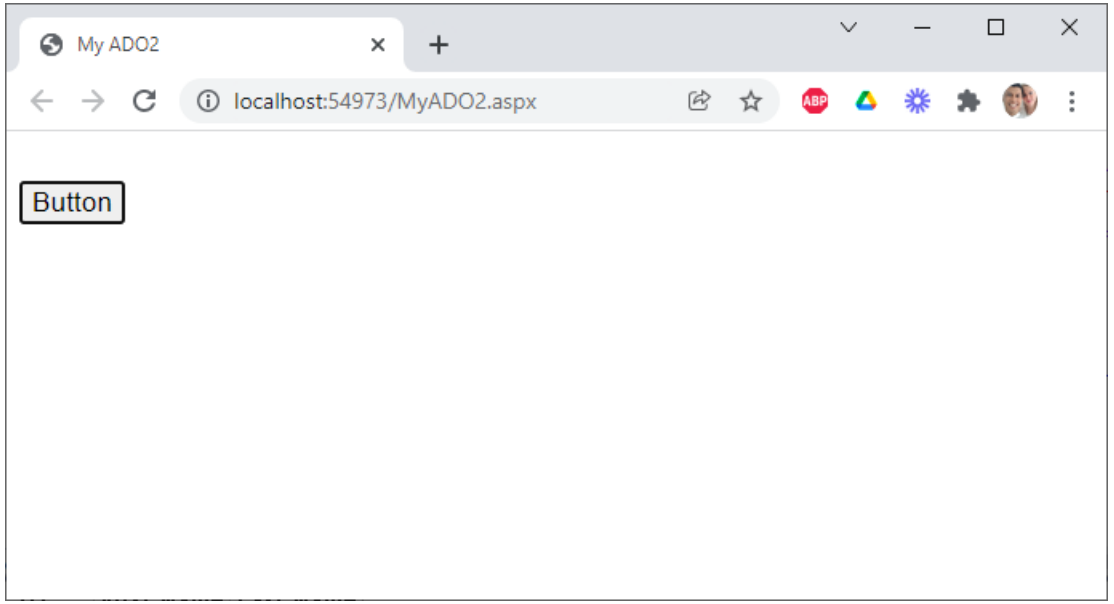
MyADO2 (6)

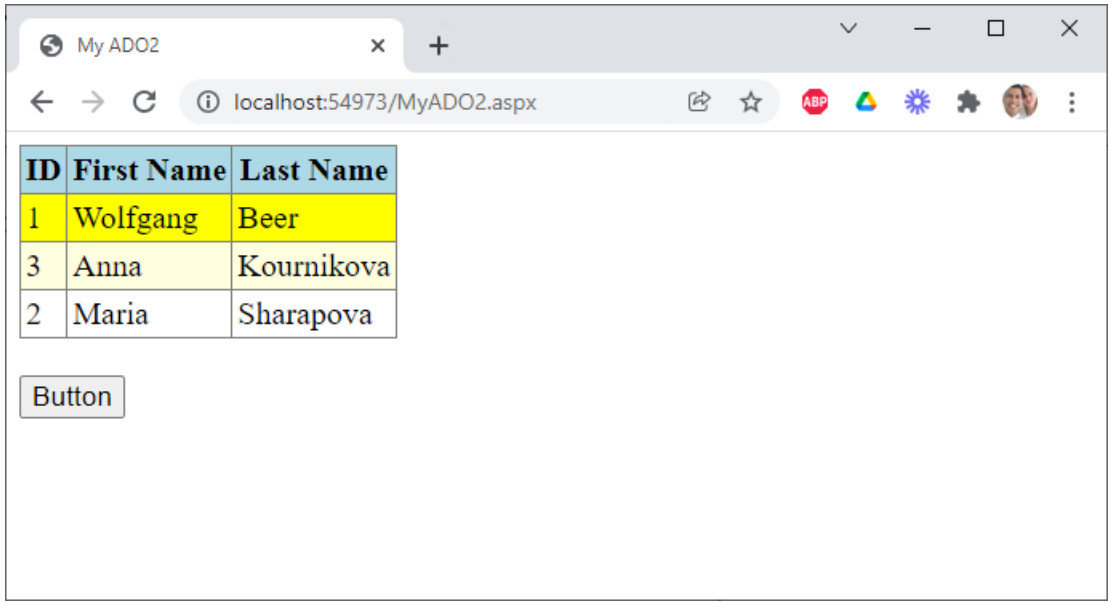
```
122         // 3
123         // Add a data to the table "Person"
124         // Define a new row & its values
125         personRow = personTable.NewRow();
126         personRow["FirstName"] = "Anna";
127         personRow["LastName"] = "Kournikova";
128         // Add the row to the table
129         personTable.Rows.Add(personRow);
130         // Add a data to the table "Contact"
131         // Define a new row & its values
132         contactRow = contactTable.NewRow();
133         contactRow["FirstName"] = "Anna";
134         contactRow["LastName"] = "Kournikova";
135         contactRow["Phone"] = "08333333333";
136         contactRow["PersonID"] = (long)personRow["ID"];
137         contactTable.Rows.Add(contactRow);
138
139         // Accept the changes permanently, if there's no errors
140         if (ds.HasErrors) {
141             ds.RejectChanges();
142         } else {
143             ds.AcceptChanges();
144         }
145
```

MyADO2 (7)

```
146             // DataView
147             DataView dataView = new DataView(personTable);
148             //dataView.RowFilter = "FirstName < 'N'";
149             //dataView.RowStateFilter = DataViewRowState.Added | DataViewRowState.ModifiedCurrent;
150             dataView.Sort = "LastName ASC";
151
152             // GridView
153             GridView1.DataSource = dataView;
154             GridView1.DataBind();
155             GridView1.HeaderStyle.BackColor = System.Drawing.Color.LightBlue;
156             GridView1.AlternatingRowStyle.BackColor = System.Drawing.Color.LightYellow;
157
158             //int i = dataView.Find("Sharapova");
159             //int i = dataView.Find("Kournikova");
160             int i = dataView.Find("Beer");
161             if (i >= 0) {
162                 GridView1.Rows[i].BackColor = System.Drawing.Color.Yellow;
163             }
164
165         }
166     }
```

MyADO2: Output





Summary

ADO.NET

Summary

- Connection-oriented data access model
  - For applications with only a few parallel, short running transactions
  - Object-oriented interface abstracts from data source
  - Access to database by SQL commands
- Connectionless data access model
  - For applications with many parallel, long running transactions
  - DataSet as main memory database
  - DataAdapter is used as connector to the data source
  - Tight integration with XML
  - Well integrated in the .NET Framework, e.g., WebForms, WinForms