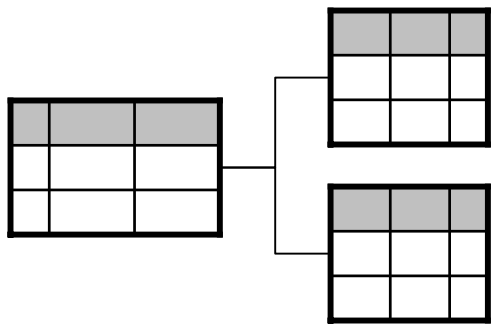
# *ADO.NET*

*Dietrich Birngruber*

*Software Architect*
*TechTalk*
*www.techtalk.at*



# *Contents*

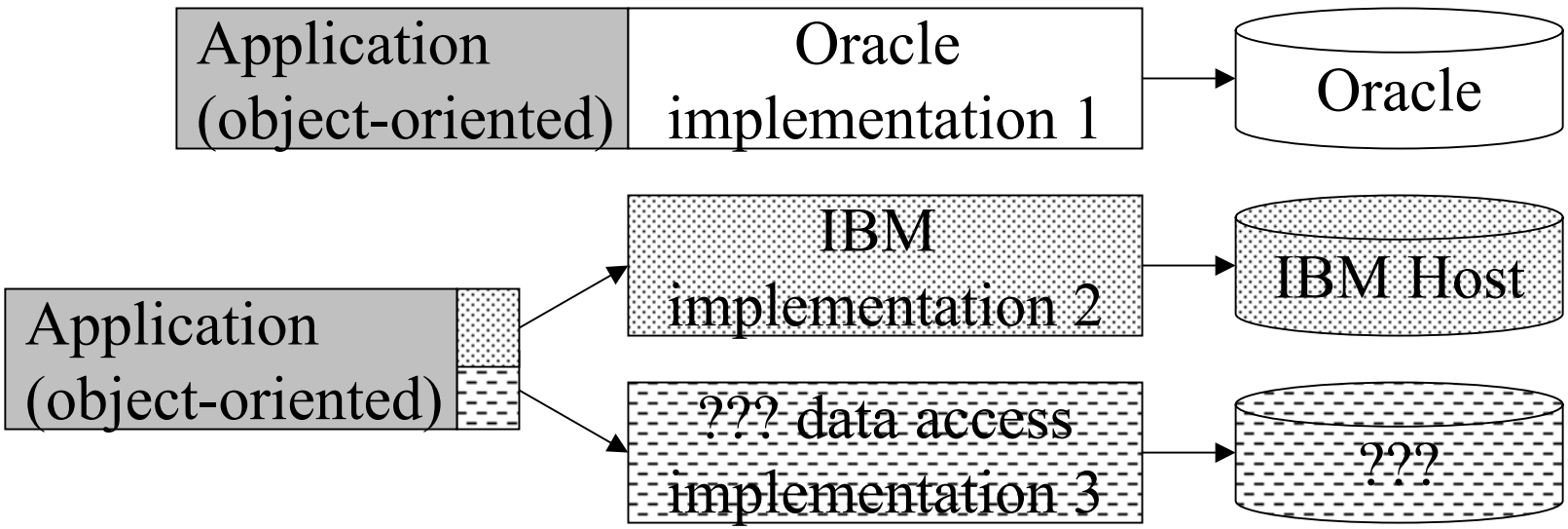
TECH TALK
it works, we care

- **Part I: Basics**
- Part II: Connection-Oriented Scenario
- Part III: Disconnected Scenario
- Part IV: Data Access Layer Sample

# *Motivation*

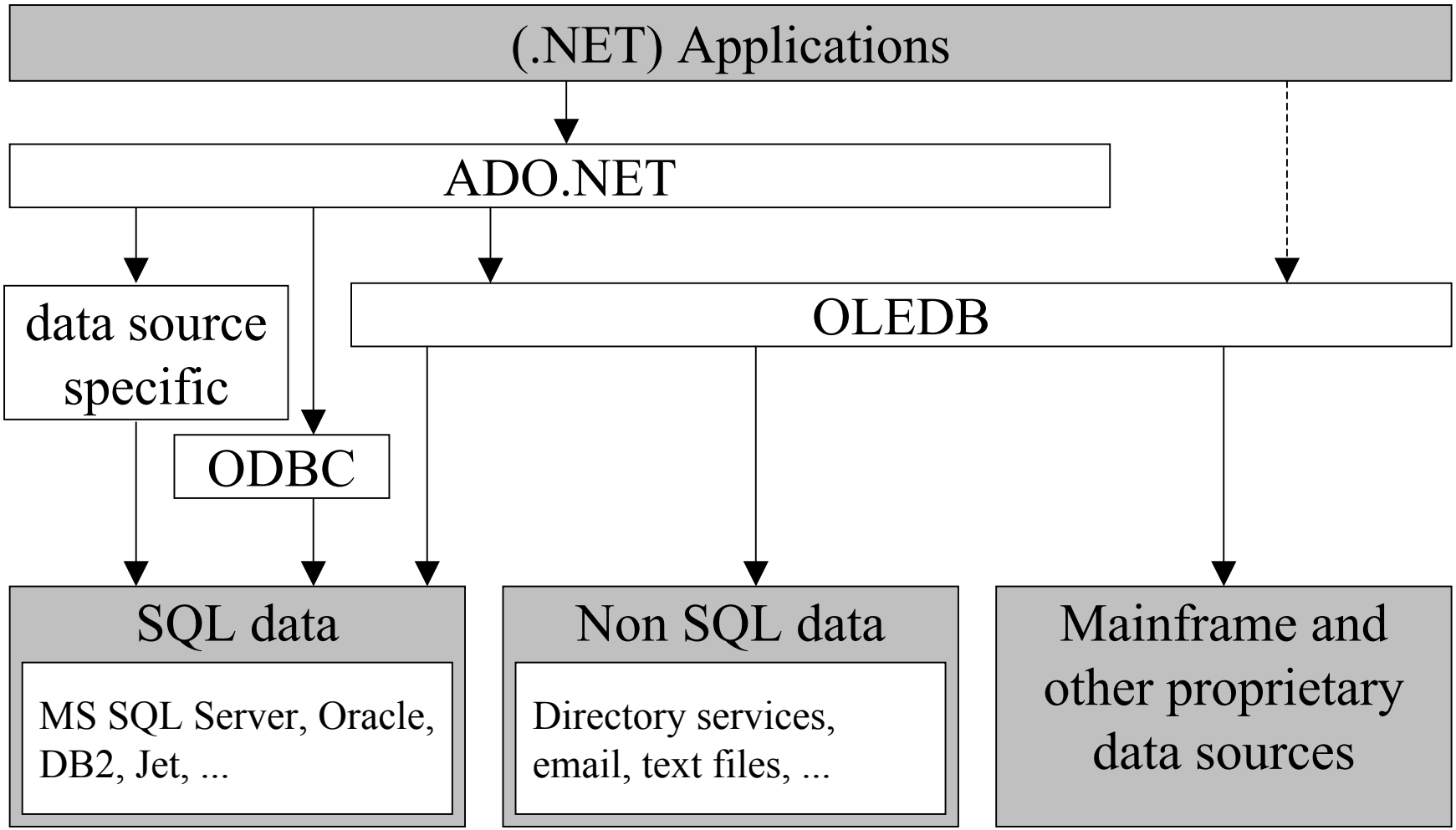
TECH TALK
it works, we care

- How to access different data sources?
  - „back office“, OO/RDBMS, files, (web-)server, ...

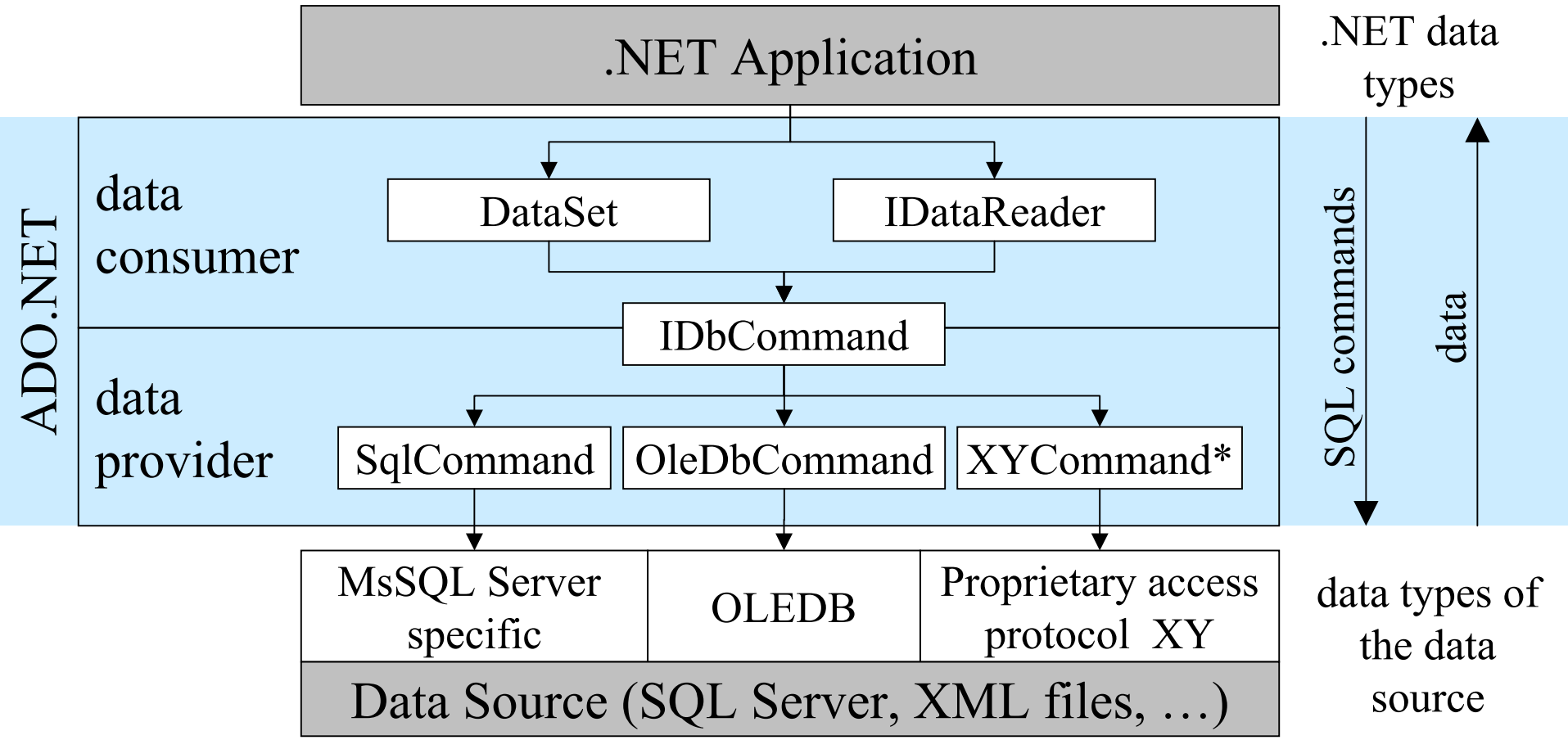



- Is there a unified programming model and API available ?

# *Microsoft Data Access Components*



# *Architectural Overview*

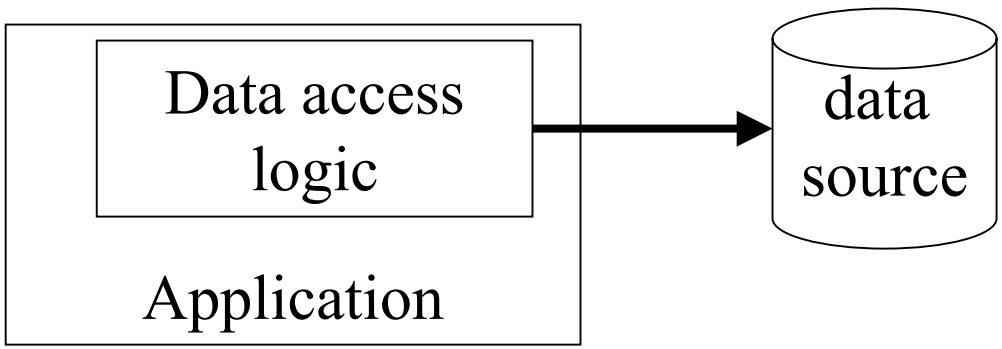



* ODBC, Oracle (v1.1)

# *Connection Oriented Scenario*

- Connection to data source remains alive
- IDataReader for reading

- Scenarios with …
  - low number of concurrent accesses
  - short running transactions
  - data always up to date

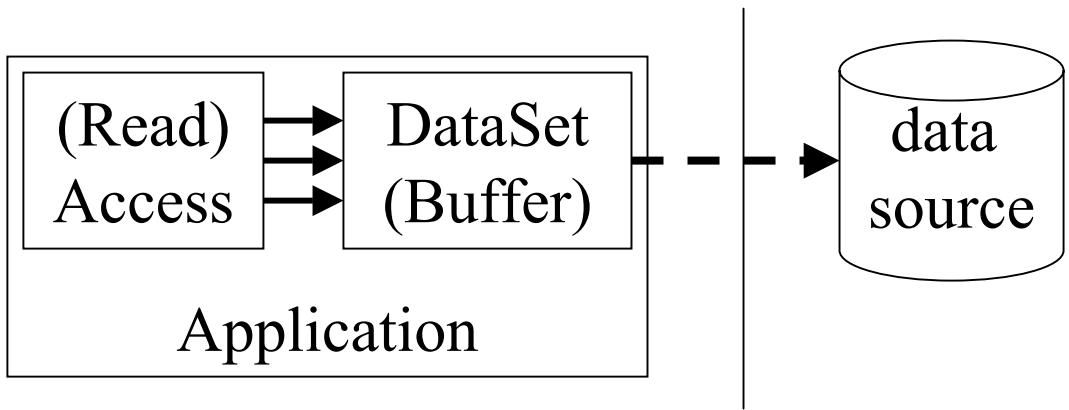


# *Disconnected Scenario*

TECH TALK
it works, we care

- *No permanent* connection to the data source
- Data is cached in DataSet
- Modifications in DataSet ≠ modifications in the data source

- Scenarios with …
  - a lot of concurrent, long running read accesses (e.g. web)



# *Namespaces (System.Data.dll)*

TECH TALK
it works, we care

- *System.Data* (basic types)
- *System.Data.OleDb* (OLEDB provider)
- *System.Data.SqlClient* (Microsoft SQL Server provider)
- System.Data.Common
- System.Data.SqlTypes
- System.Data.Odbc (ODBC provider, since .NET 1.1)
- *System.Data.OracleClient* (Oracle provider, since .NET 1.1)
- System.Data.SqlServerCe (Compact Framework)

- Extra download (prior to NET1.1):
  - Microsoft.Data.Odbc
  - System.Data.OracleClient

# *Contents*

TECH TALK
it works, we care

- Part I: Basics
- **Part II: Connection-Oriented Scenario**
- Part III: Disconnected Scenario
- Part IV: Data Access Layer Sample

# *Example: Northwind*

TECH TALK
it works, we care

- Food & beverages merchant
  - Microsoft sample; part of MSDE (MS SQL Server)
- Task: read table „Employees“
- Steps:
  1. **Connect to database** (IDbConnection)
  2. **Execute SQL command object** (IDbCommand)
  3. **Read and compute** (IDataReader)
  4. **Release resources**

```
| 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
```

# *Typical Program Structure*

TECH TALK
it works, we care

```
using System.Data;
....
1.) Declare connection;
try { // optional
   1.) Create connection to data source;
   2.) Execute SQL commands;
   3.) Compute result;
   4.) Release resources;
} catch ( Exception ) {
   Handle exception or dispatch it;
} finally {
   try { // optional
      4.) Close connection;
   } catch (Exception) { Handle exception; }
}
```

# Code: EmployeeReader

TECH TALK
it works, we care

```
using System;
using System.Data;
using System.Data.OleDb;

public class EmployeeReader {
   public static void Main() {
    //----- Connect to local data base
    string connStr = "provider=SQLOLEDB; data source=(local)\\NetSDK; " +
                     "initial catalog=Northwind; user id=sa; password=; ";
    IDbConnection con = null; // declare connection variable
    try {
     con = new OleDbConnection(connStr);
     con.Open(); //open connection
     //----- Create command object and define SQL command text
     IDbCommand cmd = con.CreateCommand(); //creates an OleDbCommand object
     cmd.CommandText = "SELECT EmployeeID, LastName, FirstName FROM Employees";

     //next slide ...
```

1.)

# Code: EmployeeReader (2)

TECH TALK
it works, we care

```
      //....
      //----- execute command object; it returns an OleDbDataReader
2.)   IDataReader reader = cmd.ExecuteReader();
      object[] dataRow = new object[reader.FieldCount];
      //----- read data row by row ... forward only cursor
3.)   while (reader.Read()) {
        int cols = reader.GetValues(dataRow);
        for (int i = 0; i < cols; i++) Console.Write("| {0} " , dataRow[i]);
        Console.WriteLine();
      }
      //----- close reader and release resources
4.)   reader.Close();
    } catch (Exception e) {
      Console.WriteLine(e.Message);
    } finally {
      try {
4.)     if (con != null)
          con.Close(); // ALWAYS close a connection!
      } catch (Exception ex) { Console.WriteLine(ex.Message); }
    }
  }
}
```

# *IDbConnection Interface*

TECH TALK
it works, we care

For creating a connection to a data source (see step 1. )...

```
public interface IDbConnection : IDisposable {
    //----- properties
    string ConnectionString {get; set;}
    int ConnectionTimeout {get;}
    ...
    //----- methods
    IDbTransaction BeginTransaction();
    IDbTransaction BeginTransaction(IsolationLevel lvl);
    void Close();
    void Open();
    IDbCommand CreateCommand();
    ...
}
```

# *ConnectionString Property*

TECH TALK
it works, we care

- Configures the connection
- Semicolon separated list of key – value pairs
- 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;"
```

# *"Execute" Methods of IDbCommand Interface*

TECH TALK
it works, we care

For executing SQL commands (see step 2. ) …

- IDataReader **ExecuteReader**(...)
  - Property *CommandText* only contains SQL SELECT statements
- int **ExecuteNonQuery**()
  - Property *CommandText* contains INSERT, UPDATE, DELETE, ...

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

# *"Execute" Methods of IDbCommand Interface*

TECH TALK
it works, we care

- object **ExecuteScalar**()
  - Property *CommandText* contains SQL aggregate functions

```
IDbCommand cmd = new SqlCommand("SELECT count(*) FROM Empls",
                                new SqlConnection(connStr));
cmd.Connection.Open();
int count = (int) cmd.ExecuteScalar();
cmd.Connection.Close();
```

# *How To: Parameters And SQL Commands*

TECH TALK
it works, we care

1. Define formal parameters

```
//here we use OLEDB
OleDbCommand cmd = new OleDbCommand();
cmd.CommandText = "DELETE FROM Empls WHERE EmployeeID = ?";
cmd.Parameters.Add(new OleDbParameter("anID", OleDbType.BigInt));
cmd.Connection = ...
```

2. Assign actual parameters and execute command

```
cmd.Parameters["anID"].Value = 1234;   // or cmd.Parameters[0]
cmd.ExecuteNonQuery();
```

# *Contents*

TECH TALK
it works, we care

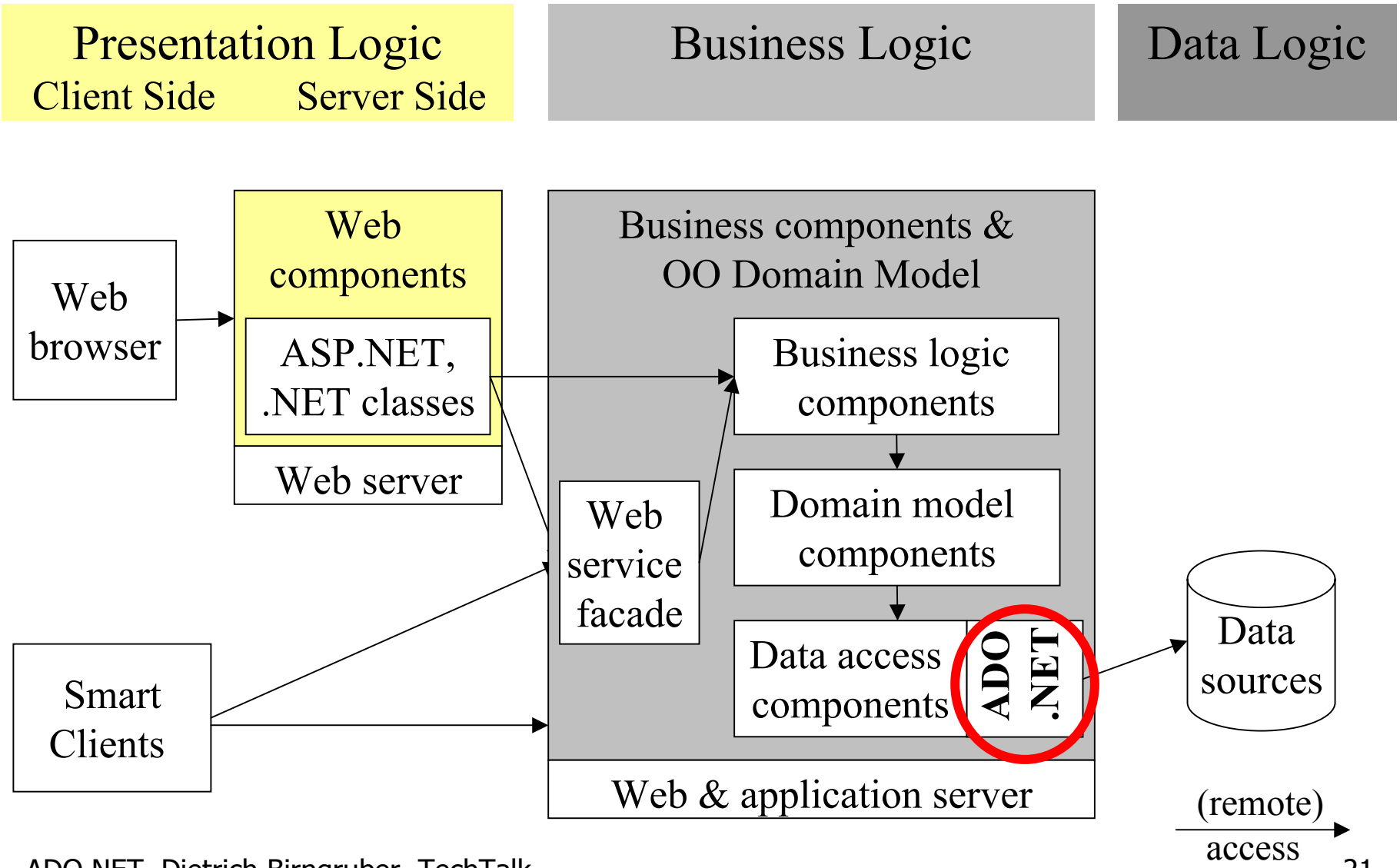
- Part I: Basics
- Part II: Connection-Oriented Scenario
- **Part III: Disconnected Scenario**
- Part IV: Data Access Layer Sample

# *Support for a Disconnected Scenario*

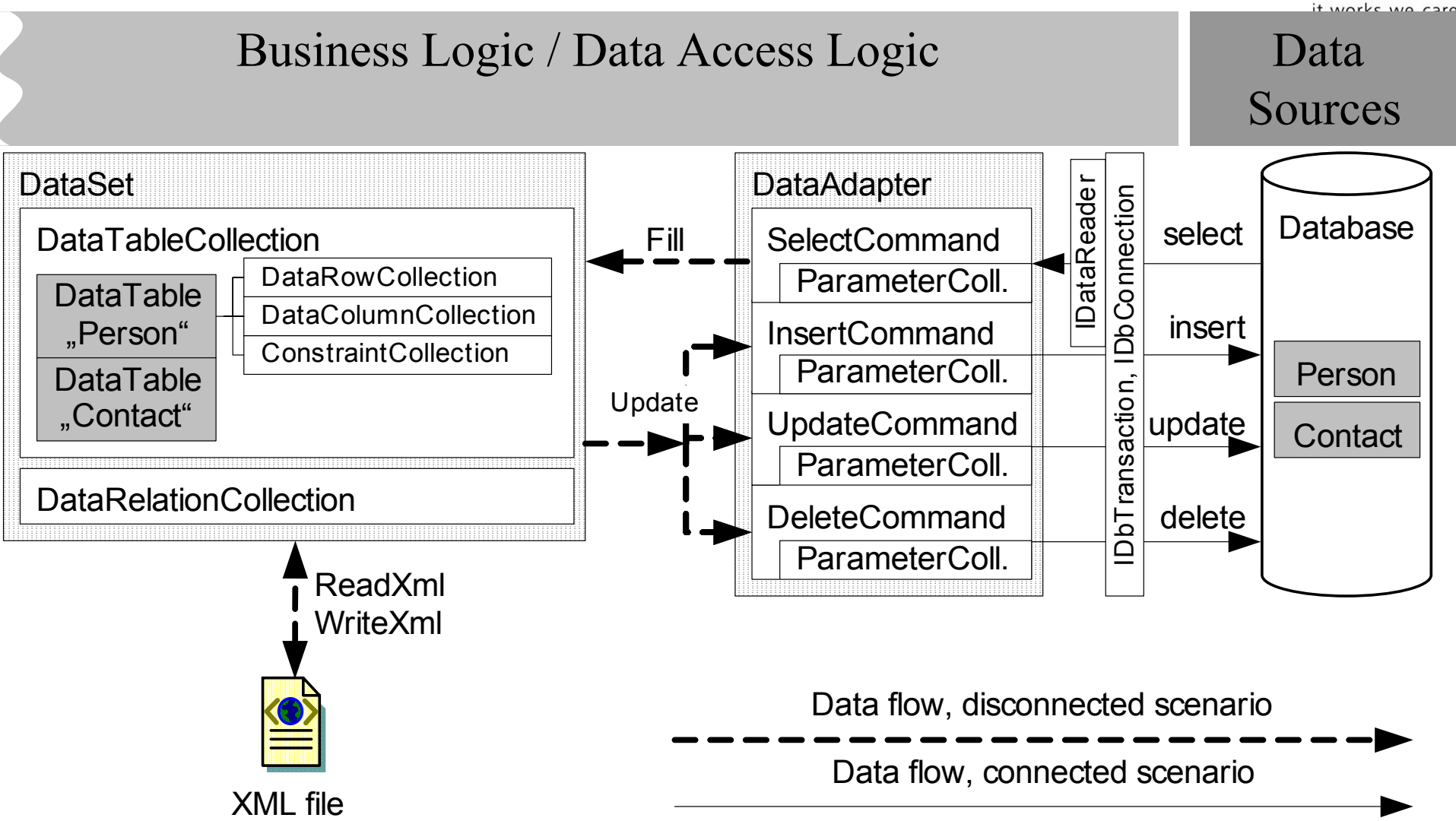
TECH TALK
it works, we care

- For distributed (enterprise) applications
  - A lot of concurrent read accesses
  - Designed for an n-tier client/server architecture in mind
  - E.g.: web shop, product catalog
- *DataSet* caches data
  - Database “snapshot" in memory
  - In memory representation of an XML data and schema
- *DataSet* is independent of a data source
  - Location, including multiple data sources (database, XML file, …)
- *DataSet* is used in .NET framework
  - ASP.NET, Web Services, Win Forms, ...
- *DataAdapter* is used for synchronization with data sources

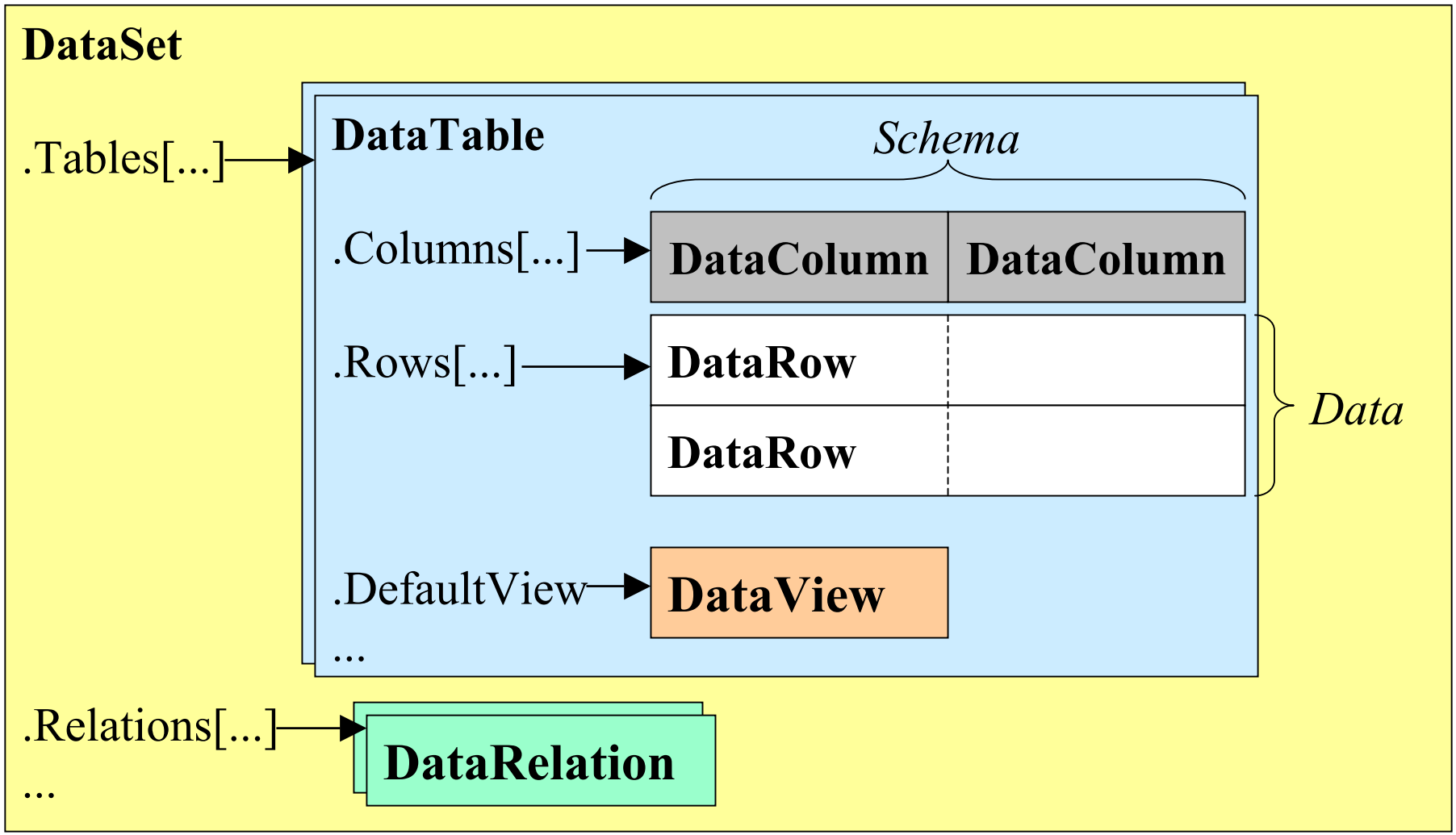
# N-Tier Client/Server Architecture



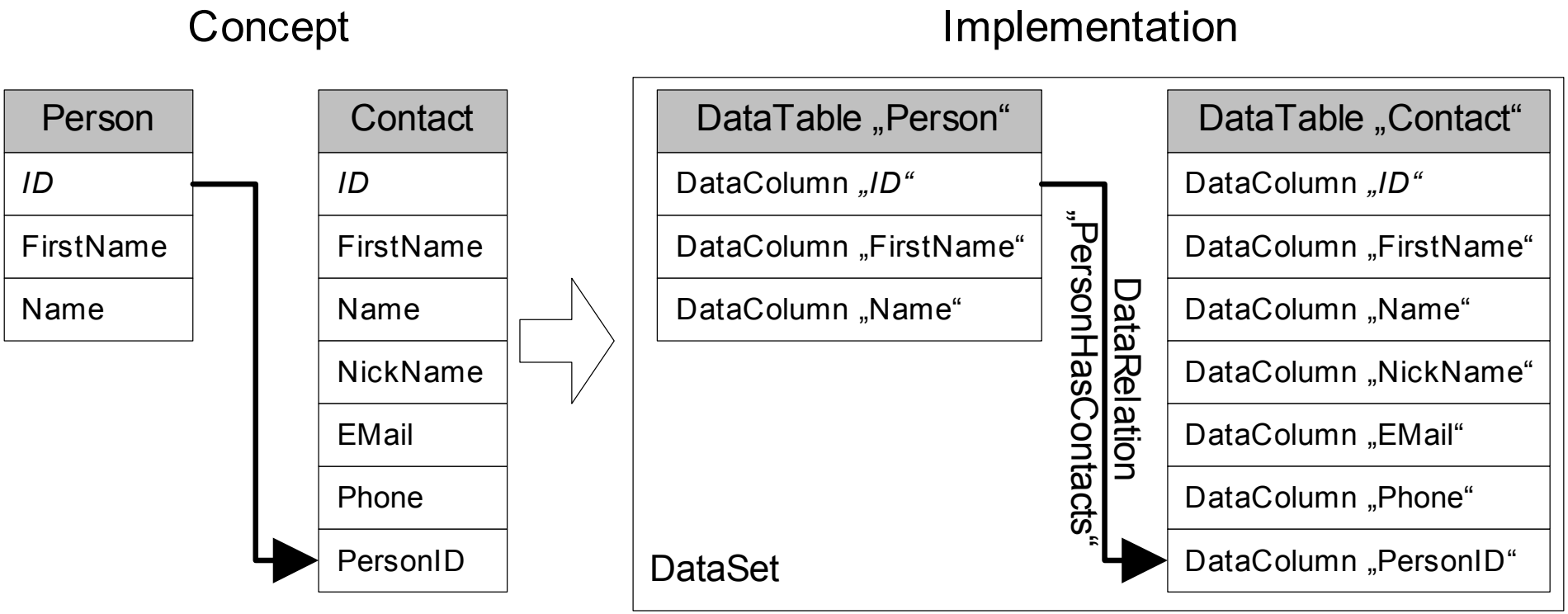
# Zoom In: N-Tier Client/Server Architecture



# *New Data Types*



# *Example: Contacts*




- TODO: to read data and fill a DataSet
- We need a DbDataAdapter or an IDbDataAdapter
  - E.g.: OleDbDataAdapter, SqlDataAdapter, ...

# Beispiel: Kontaktliste (2)

```
DataSet LoadData() {
  //----- define SELECT commands
  OleDbCommand cmd = new OleDbCommand();
  cmd.Connection = new OleDbConnection ("provider=SQLOLEDB; " +
  " data source=(local)\\NetSDK; database=netbook; user id=sa; password=;" );
  cmd.CommandText = "SELECT * FROM Person; SELECT * FROM Contact";

  DataSet ds = new DataSet("PersonContacts");
  IDbDataAdapter adapter = new OleDbDataAdapter();
  adapter.SelectCommand = cmd;
  //----- DataSet is empty and contains no DataTable objects
  adapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;
  //----- Rename automatically created DataTable objects
  adapter.TableMappings.Add("Table", "Person");
  adapter.TableMappings.Add("Table1", "Contact");
  //----- Read data and populate the DataSet
  adapter.Fill(ds);
```

# *Beispiel: Kontaktliste (3)*

TECH TALK
it works, we care

```
  if (ds.HasErrors) {
    ds.RejectChanges();
  } else {
    DefineRelation(ds); //... or read schema information from the data source
    ds.AcceptChanges();
  }
  if (adapter is IDisposable) ((IDisposable)adapter).Dispose();
  return ds;
}

void DefineRelation(DataSet ds) {
  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);
}
```

# *Contents*

TECH TALK
it works, we care

- Part I: Basics
- Part II: Connection-Oriented Scenario
- Part III: Disconnected Scenario
- **Part IV: Data Access Layer Sample**

# *Example: Data Access Layer*

TECH TALK
it works, we care

- Mismatch: object-oriented world vs. relational World
  - Objects, classes, inheritance, object graph, life-time, …
  - Row, tables, relation, constraints, joins, …
- Goal: Mapping OO – relational world
  - Reusable domain model (classes)
  - Transparent persistency mechanism
- Tool for developer
- Object Query Language
  - Set oriented and NO tables!
  - “Class” == type + all objects
- Used in .NET projects at TechTalk

# *Usage Scenario*

TECH TALK
it works, we care

1. Design and implement object-oriented domain model
   - Abstract classes, separate assembly (DLL)

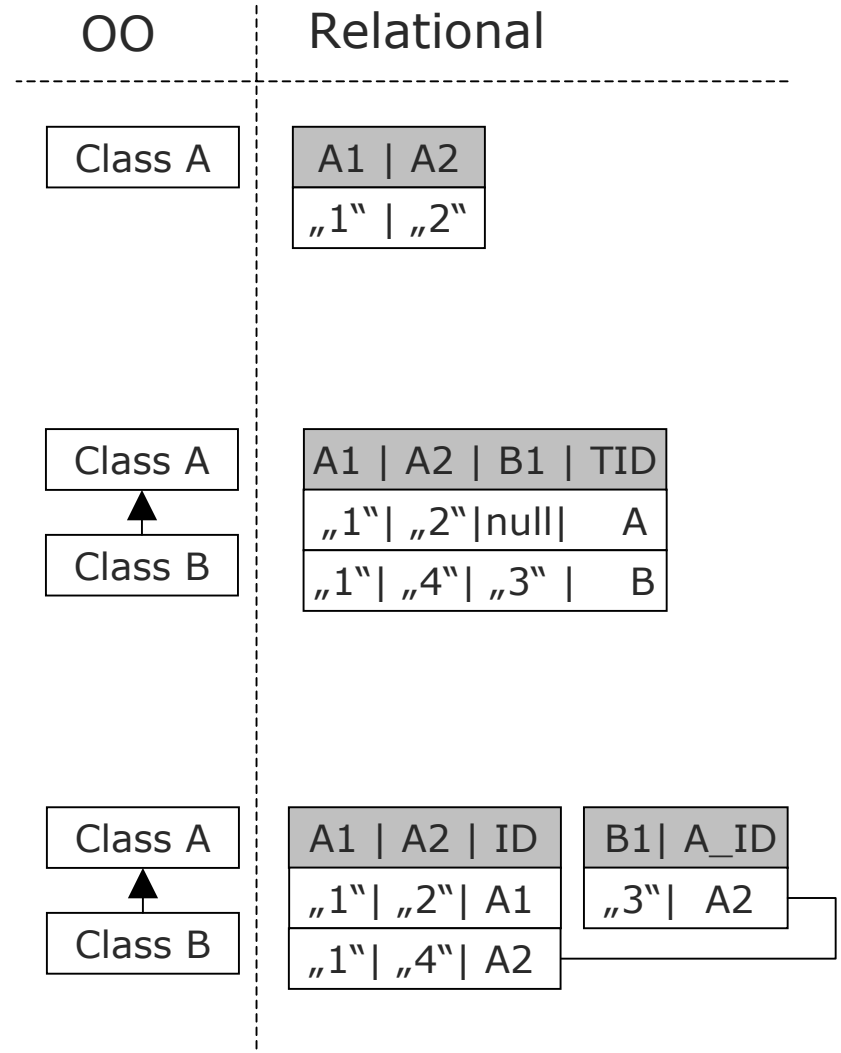
2. Define mapping rules in XML file
   - Classes of the domain model ↔ tables in the relational data base

3. Generate separate mapping assembly (DLL) with a tool
   - Uses ADO.NET and contains SQL commands

4. Test and use generated files
   - With a tool and in your code

# *Supported OO-Relational Mapping Strategies*

- Root Inheritance
  - NO hierarchy
  - Only objects of type „A“
- Shared Inheritance
  - All instances in one table
  - Type discriminator
  - null values in DB,
    but fast read access
- Joined Inheritance
  - 1 type == 1 table
  - Hierarchy == relation (SQL joins)



# *Summary*

TECH TALK
it works, we care

- Connection-oriented scenario
  - Always current data
  - Low number of concurrent data accesses
  - Many write access situations
  - IDbConnection, IDbCommand, IDataReader
- Disconnected scenario
  - Modifications in DataSet ≠ modifications in the data source
  - Many concurrent read access situations
  - DataSet, DataTable, DbDataAdapter
- DAL example
  - Generative programming approach
  - Bridges the gap between OO world and relational world

# *Questions + Contact*

TECH TALK
it works, we care

http://dotnet.jku.at

http://www.ssw.uni-linz.ac.at

http://www.techtalk.at

birngruber@acm.org