INFORMATION PROCESSING TECHNIQUES

ADO.NET

WEEK11

MURTAZA MUNAWAR FAZAL

Overview

- Overview of ADO.NET
- Disconnected vs. connected data access models
- ADO.NET Architecture
- ADO.NET Core Objects
- Creating a Connection to a Database
- Displaying a DataSet in a List-Bound Control

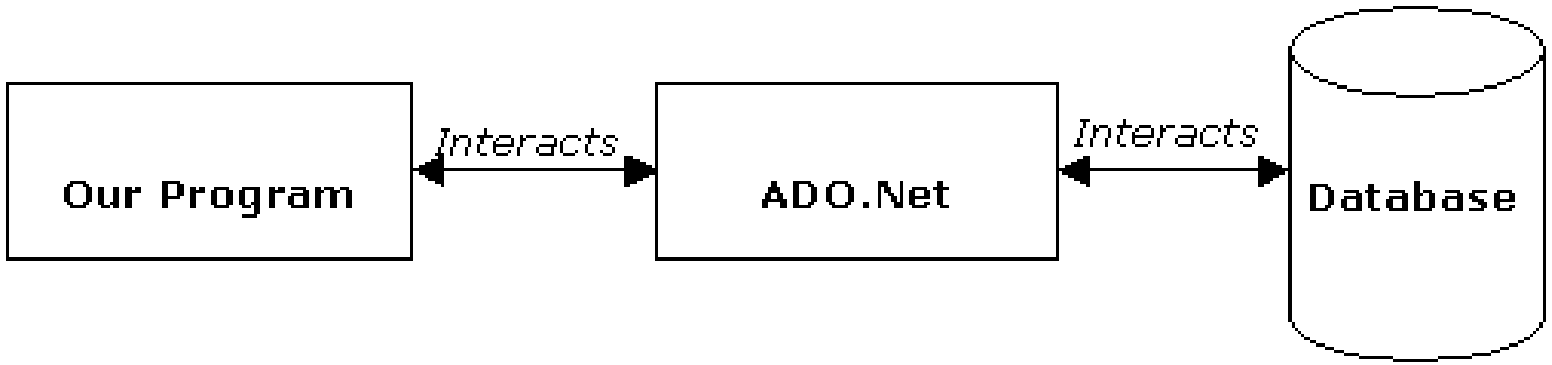
What is ADO.NET?

ADO.NET provides a set of classes for working with data. ADO.NET provides:

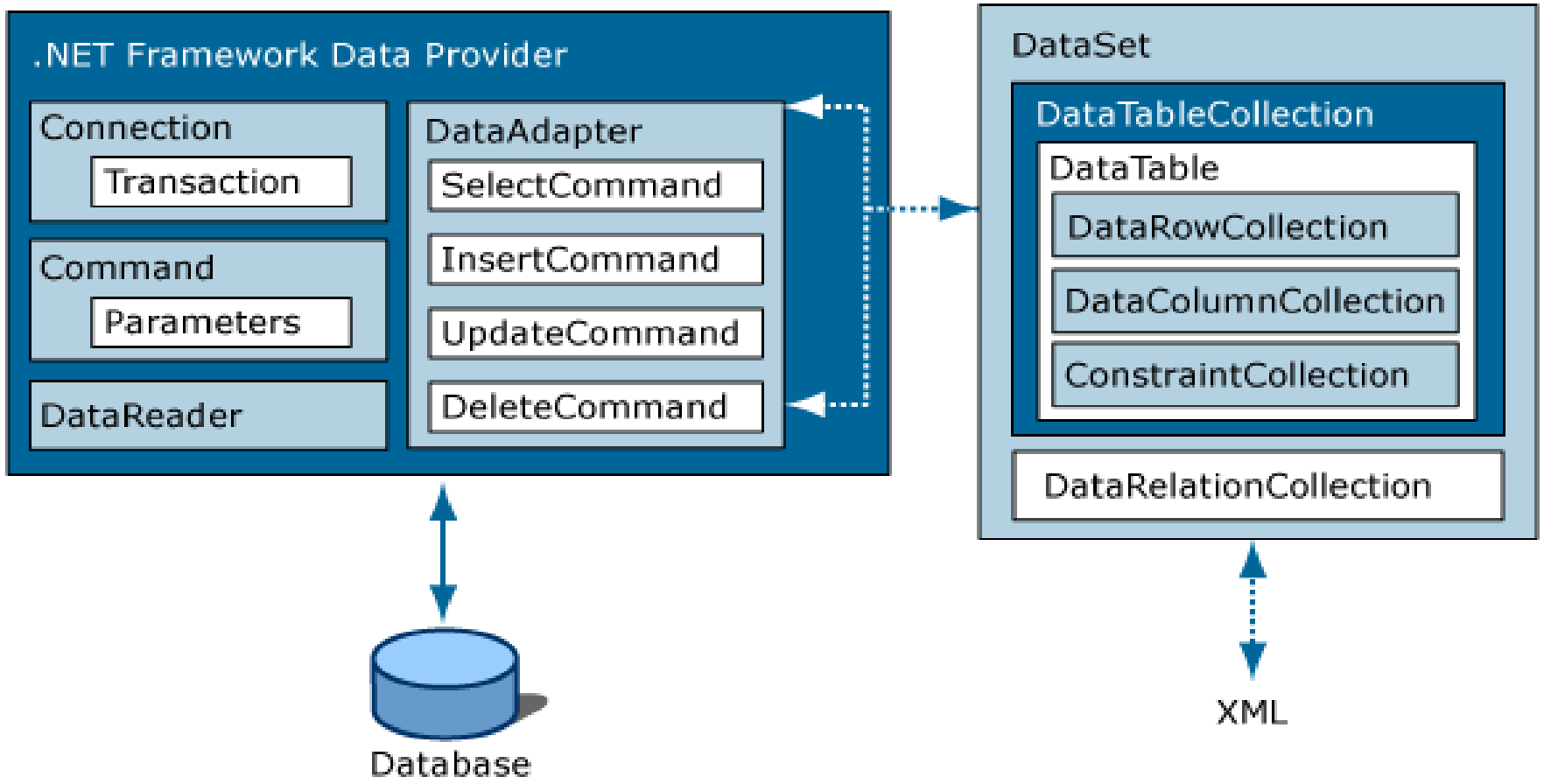
- **An evolutionary, more flexible successor to ADO**
- **A system designed for disconnected environments**
- **A programming model with advanced XML support**
- **A set of classes, interfaces, structures, and enumerations that manage data access from within the .NET Framework**

What is ADO.NET?

An object oriented framework that allows you to interact with database systems



ADO.NET Architecture



ADO.NET Core Objects

Object	Description
Connection	Establishes a connection to a specific data source. (Base class: DbConnection)
Command	Executes a command against a data source. Exposes **Parameters** and can execute within the scope of a **Transaction** from a **Connection**. (The base class: DbCommand)
DataReader	Reads a forward-only, read-only stream of data from a data source. (Base class: DbDataReader)
DataAdapter	Populates a **DataSet** and resolves updates with the data source. (Base class: DbDataAdapter)
DataTable	Has a collection of DataRows and DataColumns representing table data, used in disconnected model
DataSet	Represents a cache of data. Consists of a set of DataTables and relations among them

Connected Environment (Scenario)

1. Open connection
2. Execute command
3. Process rows in reader
4. Close reader
5. Close connection

Connected Environment

Working with data directly via open connection

Advantages

- Simple security realization
- Work with real data
- Simple organization of distributed work

Drawbacks

- Continual connection
- Not available via Internet

Disconnected Environment (Scenarion)

1. Open connection
2. Fill the DataSet
3. Close connection
4. Process the DataSet
5. Open connection
6. Update the data source
7. Close connection

Disconnected Environment

Storage of data local copy from repository

Possibility to update the main data source

Advantages

- Economy of server resources
- Does not require continual connection

Drawbacks

- Demands conflict resolution while data update
- Data is not always up to date

.NET Data Providers

Concept of data provider

Provider types

- SQL .NET Data Provider
- Oracle .NET Data Provider
- OleDB .NET Data Provider
- Odbc .NET Data Provider

How to select data provider

Using Namespaces

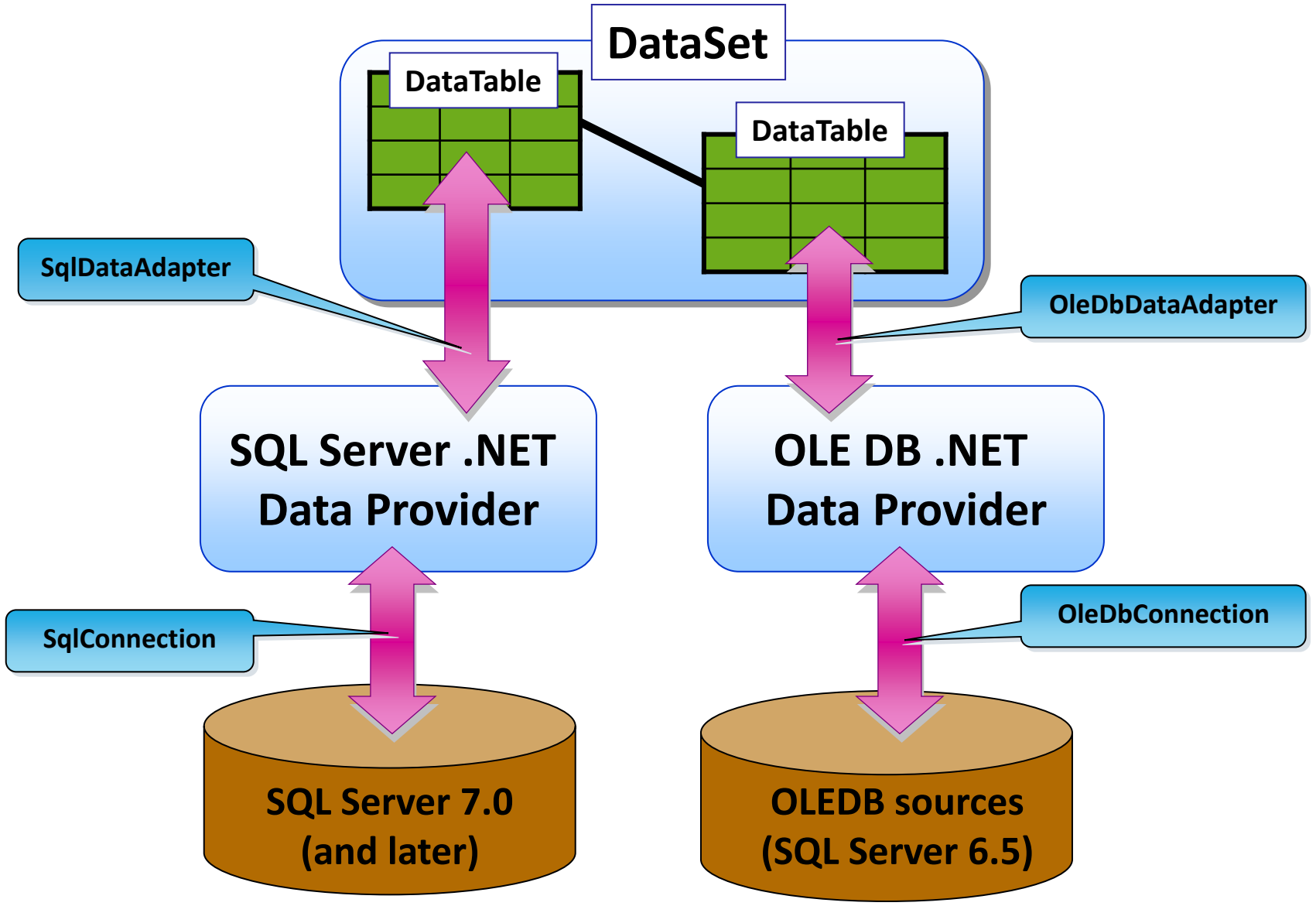
Use the Imports or using statement to import namespaces

```
using System.Data;
using System.Data.SqlClient;
```

Namespaces used with ADO.NET include:

- **System.Data**
- **System.Data.SqlClient**
- **System.Data.OleDb**

The ADO.NET Object Model



Connection

What is Connection?

Define Connection

- SqlConnection conn=new SqlConnection();
- Conn.ConnectionString=“User ID=sa;password=; Data Source=MyServer;Initial Catalog=Northwind;”

ConnectionString Parameters

- Provider
- Data Source
- Initial Catalog
- Integrated Security
- UserID/Password

Connection (Error and Pooling)

System.Data.SqlClient.SqlException

Errors collection

SqlError

- Class
- LineNumber
- Message
- Number

Pooling and Dispose method

Command Object

A command object is a reference to a SQL statement or stored procedure

Properties

- Connection
- CommandType
- CommandText
- Parameters

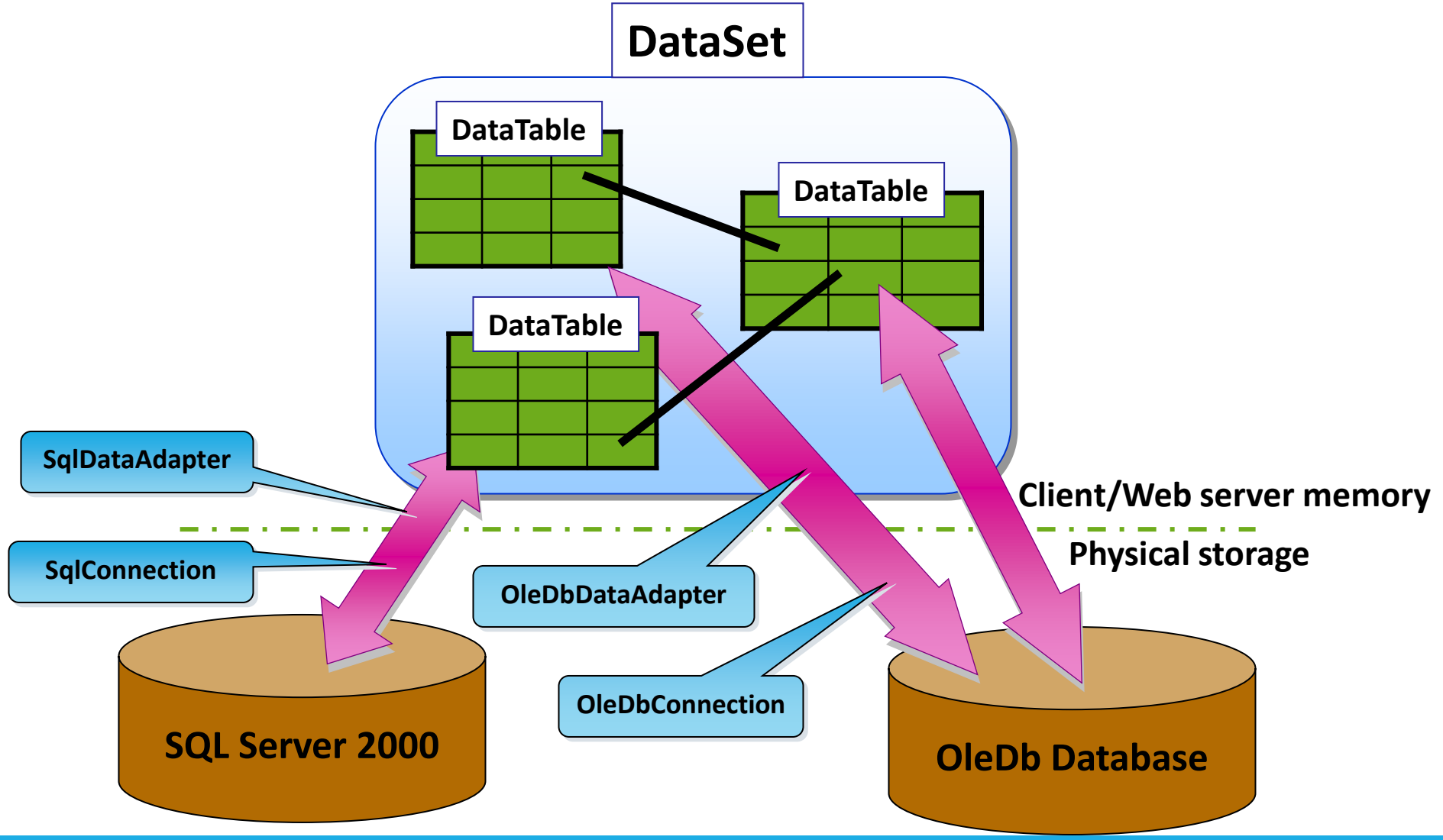
Methods

- ExecuteNonQuery
- ExecuteReader
- ExecuteScalar

DataReader Object

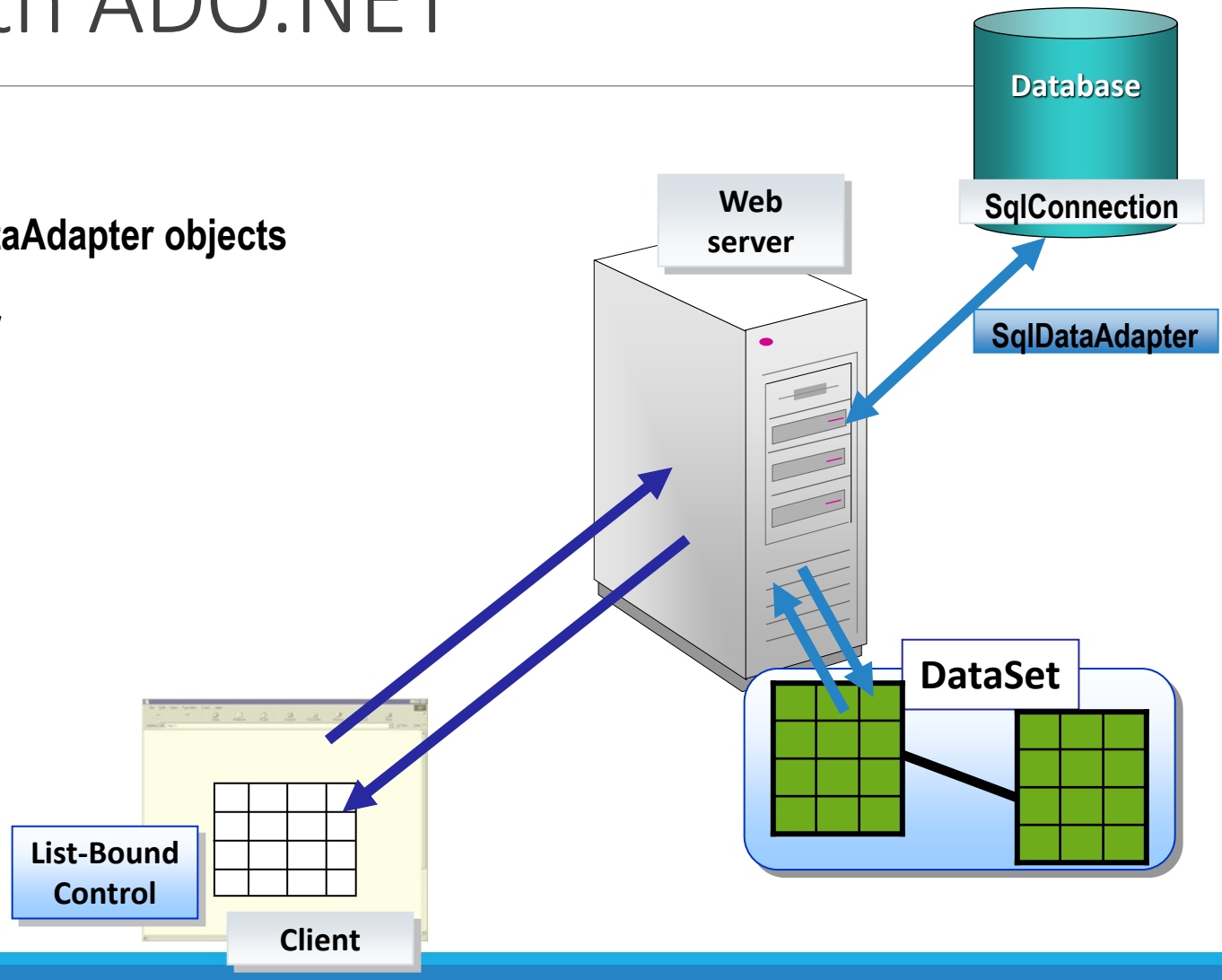
- What is query?
- Forward-only cursor
- Read method
  - Read next record
  - Return true if record is exist
- IsDbNull
- Close method
- NextResult – for multiply select statements

What is a Dataset?

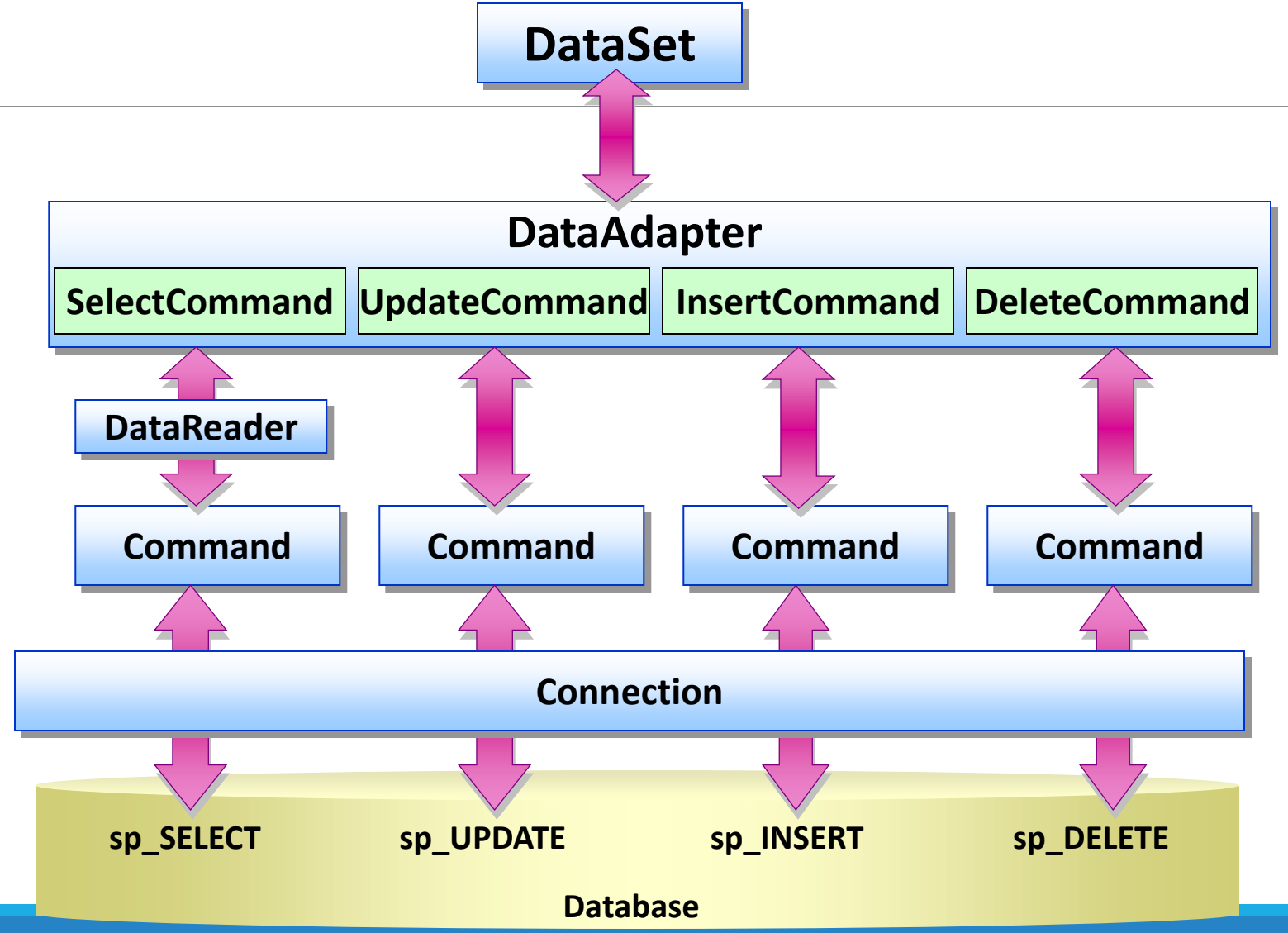


Accessing Data with ADO.NET

1. Client makes request
2. Create the SqlConnection and SqlDataAdapter objects
3. Fill the DataSet from the DataAdapter and close the connection
4. Return the DataSet to the Client
5. Client manipulates the data
6. Update the DataSet
7. Use the SqlDataAdapter to open the SqlConnection, update the database, and close the connection



The DataAdapter Object Model



Creating a DataAdapter

- **Store the query in a DataAdapter**

```
SqlDataAdapter da = new SqlDataAdapter
    ("select * from Authors",conn);
```

- **The DataAdapter constructor sets the SelectCommand property**

```
da.SelectCommand.CommandText;
da.SelectCommand.Connection;
```

- **Set the InsertCommand, UpdateCommand, and DeleteCommand properties if needed**

Generating a DataSet

You can generate a DataSet…

- …through the UI…
  - Creates a **DataSet** that allows you to access data as an object
- …or through code…

```
DataSet ds = new DataSet();
```

and then fill…

```
DataAdapter1.Fill(ds);
DataAdapter2.Fill(ds);
```

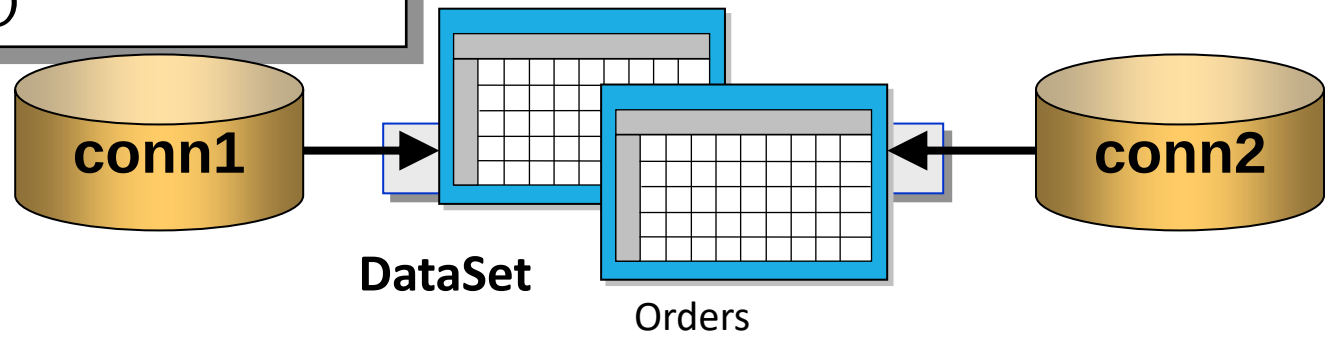
Storing Multiple Tables

- **Add the first table**

```
daCustomers = New SqlDataAdapter _
   ("select * from Customers", conn1)
daCustomers.Fill(ds, "Customers")
```

- **Add the subsequent table(s)**

```
daOrders = New SqlDataAdapter _
   ("select * from Orders", conn2)
daOrders.Fill(ds, "Orders")
```



Getting data

SqlCommand
- ExecuteReader
- ExecuteNonQuery
- ExecuteScalar
- ExecuteXMLReader

SqlDataAdapter
- DataSet

Command Methods

.ExecuteReader() - Returns DataReader

.ExecuteNonQuery() - **Returns # of Rows Affected**

.ExecuteXMLReader() - Returns XMLReader Object to Read XML documentation

.ExecuteScaler() - Returns a Single Value e.g. SQL SUM function.

The DataReader object

DataReader objects are highly optimised for fast, forward only enumeration of data from a data command

A DataReader is **not** disconnected

DataAdapters

Pipeline between DataSets and data sources

Geared towards functionality rather than speed

Disconnected by design

Supports select, insert, delete, update commands and methods

DataAdapters

Must always specify a select command

All other commands can be generated or specified

Choosing a DataReader or a Dataset

The type of functionality application requires should be considered

Use a dataset to:

- Cache data locally in your application so that you can manipulate it
- 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 readonly data is needed use **`DataReader`** to boost performance

Best Practices

- Don’t create a new connection string for every code connecting to DB
- Use app / web config file to keep your connection strings through the application scope
- Accessing settings at runtime
- You can keep any other variable to reach at runtime using this technique

XML Support

ADO.NET is tightly integrated with XML

- Using XML in a disconnected application