
Microsoft®
.net™

Web програмиране с .NET

Модели за работа с данни

- Свързан модел (connected model)
  - Постоянна връзка с данните (online)
  - Случаи на използване
  - Проблеми – лоша скалируемост
- Несвързан модел (disconnected model)
  - Връзката с данните се осъществява offline – за изтегляне на данни и нанасяне на промени по данните
  - Случаи на използване
  - Примери
    - Достъп до данни чрез Web услуга
    - Интеграция с XML

Свързан модел

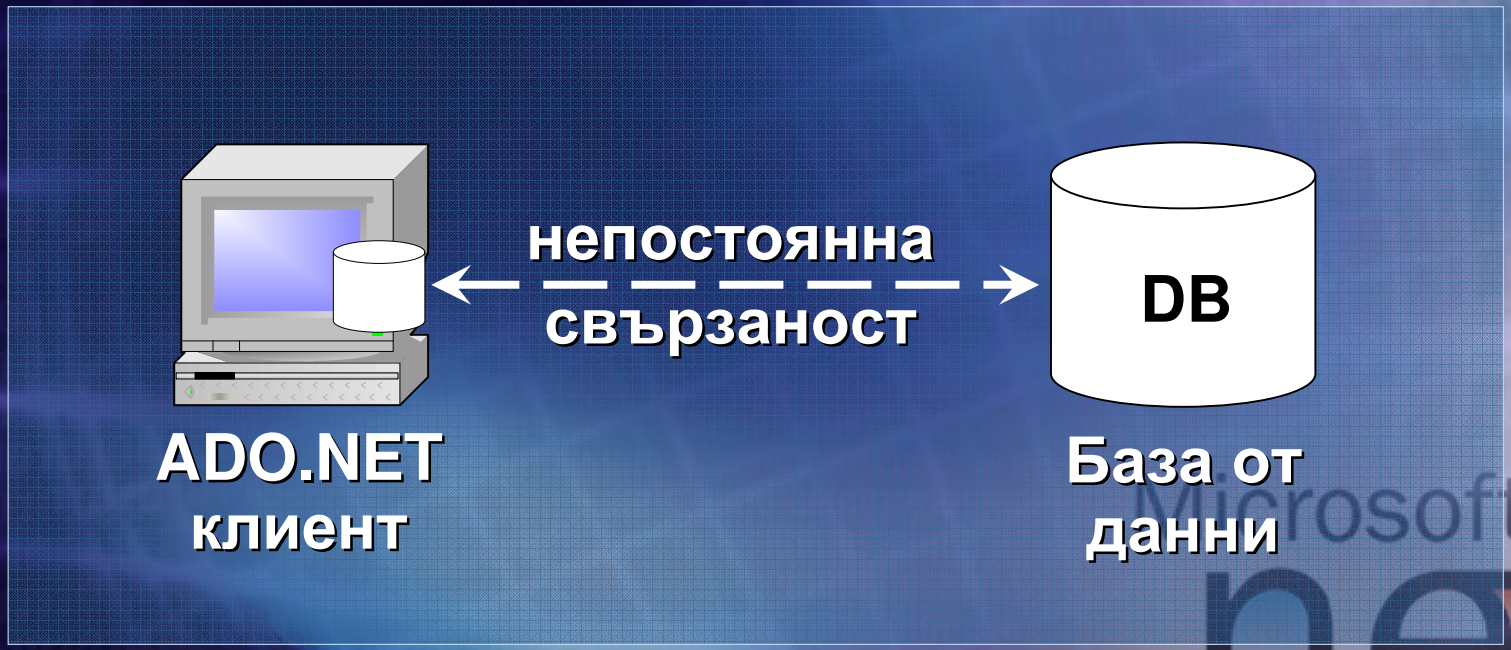
- Реализация на достъп до данни в среда, в която винаги има връзка до източника на данните

Свързан модел – за и против

- Предимства:
  - Средата е по-лесна за подсигуряване (по-малко усилия за разработчика)
  - Контролът върху конкурентният достъп се упражнява по-лесно
  - По-добра вероятност за работа с текущата версия на данните
- Недостатъци:
  - Нужда от постоянна мрежова връзка
  - Проблеми при нужда от скалируемост

Несвързан модел

- Подмножество на данните от централната система за съхранение на данните се копира локално при клиента



Несвързан модел – за и против

- Предимства:
  - Клиентът се свързва, когато има нужда, а в останалото време работи без връзка с централната база от данни
  - Други потребители могат да се свързват междувременно
  - Скалируемостта е доста добра
- Недостатъци:
  - Данните не винаги са текущи
  - Допълнителни усилия за решаване на конфликтите между различните версии

Еволюция на приложенията

- Еднослойни приложения
  - Най-често работи само един потребител
  - Предимства
    - Всички компоненти са на едно място
  - Недостатъци
    - Промяна на функционалността изисква преинсталация
  - Пример
    - Приложение базирано на MS Access

Еволюция на приложенията

- Двуслойни приложения (клиент-сървър)
  - Потребителският интерфейс и бизнес правилата се дефинират на едно място
  - Данните се съхраняват във втория слой
  - Предимства
    - Има разделяне на функционалността
  - Недостатъци
    - Лоша скалируемост – проблеми с поддръжката на голям брой клиенти
  - Примери
    - MS SQL Server ↔ MS Query Analyzer
    - MS Exchange ↔ MS Outlook

Еволюция на приложенията

- Трислойни приложения
  - Различните типове функционалност са в различни слоеве
  - Предимства
    - Отделяне на функционалността между потребителски интерфейс, бизнес правила и съхранение / достъп до данните
  - Недостатъци
    - По-трудна поддръжка
    - Повече усилия за осигуряване на сигурността
  - Пример
    - ASP.NET Web-приложение ↔ ASP.NET Web услуга ↔ MS SQL Server

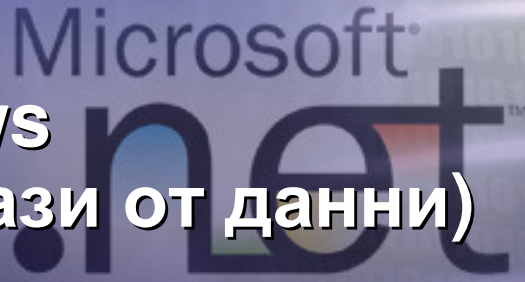
Еволюция на приложенията

- Многослойни приложения
  - Системи с повече от 3 логически слоя
  - Възможност за добавяне на още слоеве за разширяване на функционалността
  - Предимства
    - Възможност за различни приложения да достъпват части от функционалността през отворени протоколи
  - Недостатъци
    - Много труден процес по дефиниране и реализация на правила за сигурен достъп
    - Изискват повече планиране и по-големи срокове за разработка

В търсене на баланса

Какво е ADO.NET?

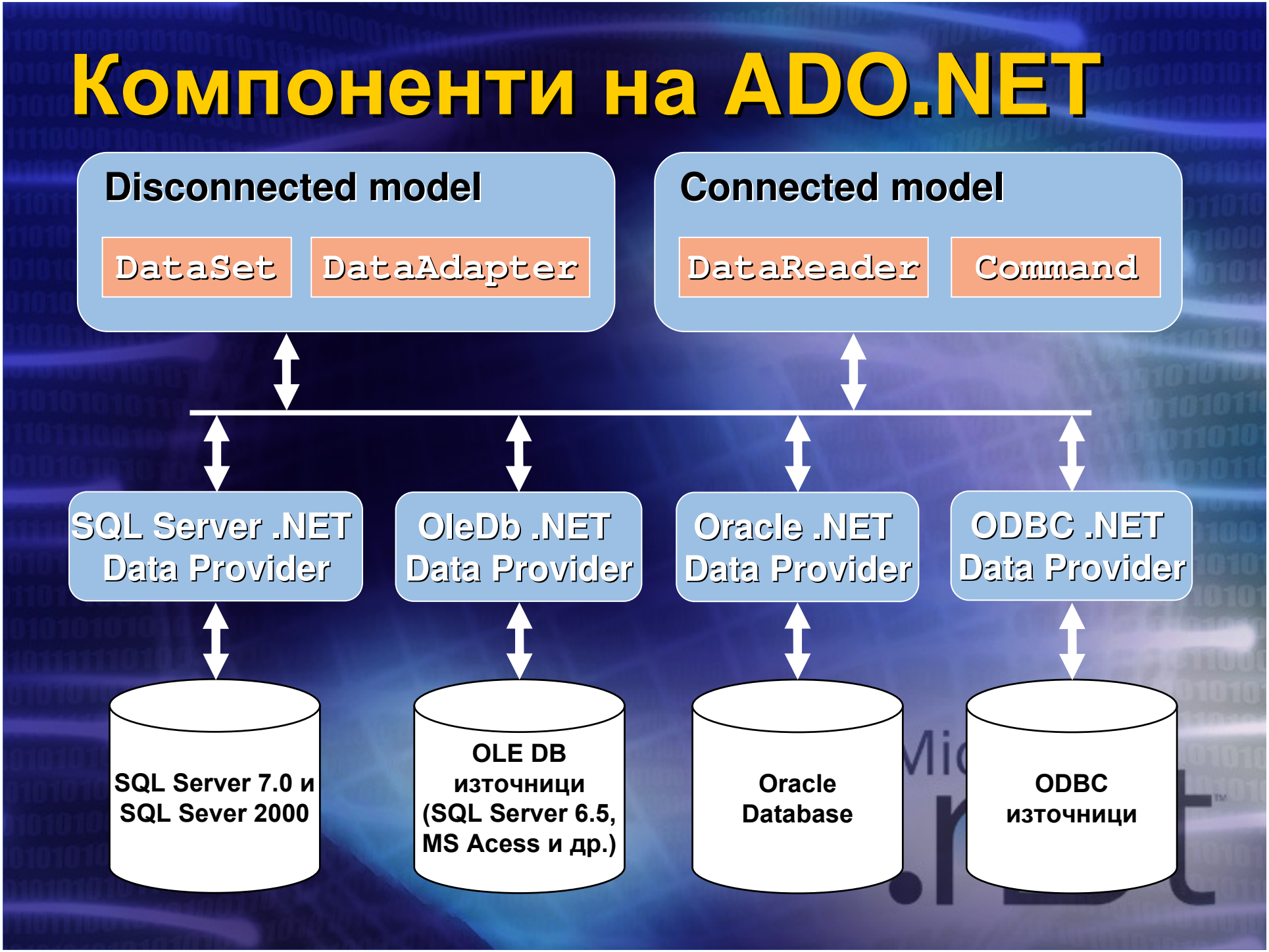
- Набор от класове за работа с данни
  - Набор от класове, интерфейси, структури и други типове за достъп до данни през изцяло .NET базирана реализация
  - Програмен модел за работа с данни
  - Осигурява възможност за работа в несвързана среда
  - Осигурява връзка с XML
  - Наследник на ADO (Windows технология за достъп до бази от данни)



Namespace-и на ADO.NET

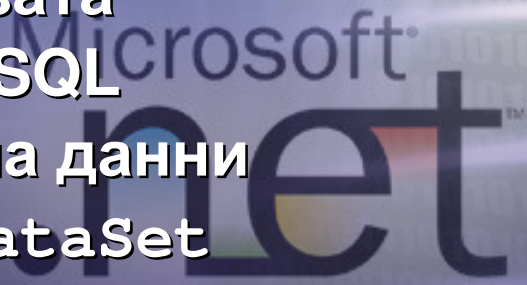
- Пространства от имена от ADO.NET
  - `System.Data` – основни архитектурни класове на ADO.NET
  - `System.Data.Common` – общи класове за всички data Provider-и
  - `System.Data.SqlClient` и `System.Data.SqlTypes` – Data Provider класове за достъп до SQL Server
  - `System.Data.OleDb` – връзка с OleDB
  - `System.Data.Odbc` – връзка с ODBC
  - `System.Xml` – връзка с XML

Компоненти на ADO.NET



Data Provider-и в ADO.NET

- Data Provider-ите са съвкупности от класове, които осигуряват връзка с различни бази от данни
  - За различните RDBMS системи се използват различни Data Provider-и
    - Различните производители използват различни протоколи за връзка със сървърите за данни
  - Дефинират се от 4 основни обекта:
    - `Connection` – за връзка с базата
    - `Command` – за изпълнение на SQL
    - `DataReader` – за извличане на данни
    - `DataAdapter` – за връзка с `DataSet`



Data Provider-и в ADO.NET

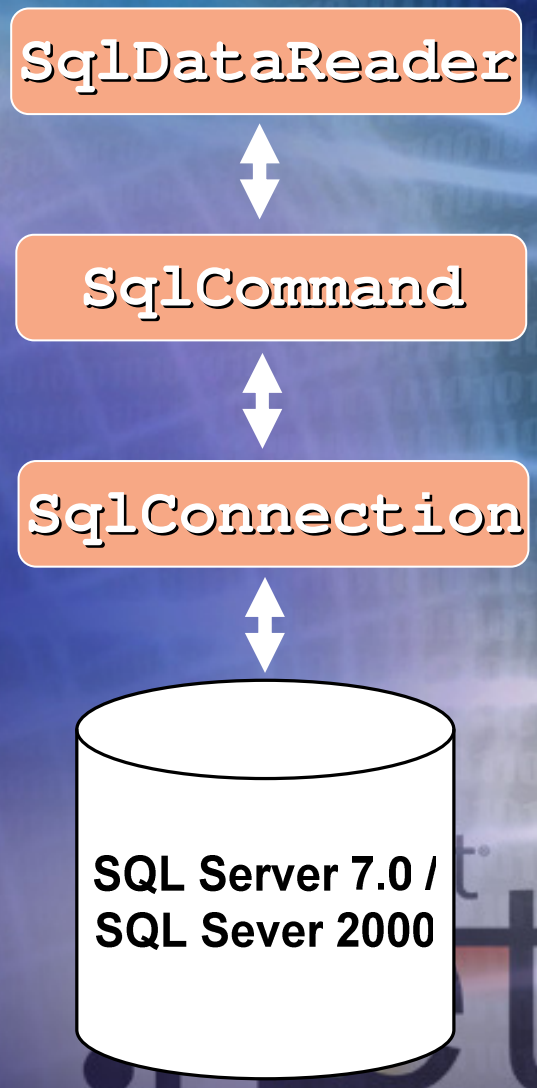
- В ADO.NET има няколко стандартни Data Provider-а
  - `SqlClient` – за връзка със SQL Server
  - `OleDB` – за връзка със стандарта OleDB
  - `Odbc` – за връзка със стандарта ODBC
  - `Oracle` – за връзка с Oracle
- Трети доставчици предлагат Data Provider-и за връзки с други RDBMS:
  - IBM DB2
  - MySQL
  - PostgreSQL
  - Borland Interbase / Firebird

SqlClient Data Provider

- `SqlConnection` – осъществява връзката с MS SQL Server
- `SqlCommand` – изпълнява команди върху SQL Server-а през вече установена връзка
- `SqlDataReader` – служи за извличане на данни от SQL Server-а
  - Данните са резултат от изпълнена команда
- `SqlDataAdapter` – обменя данни между `DataSet` обекти и SQL Server
  - Осигурява зареждане на `DataSet` с данни и обновяване на променени данни
  - Може да се грижи сам за състоянието на връзката с базата данни

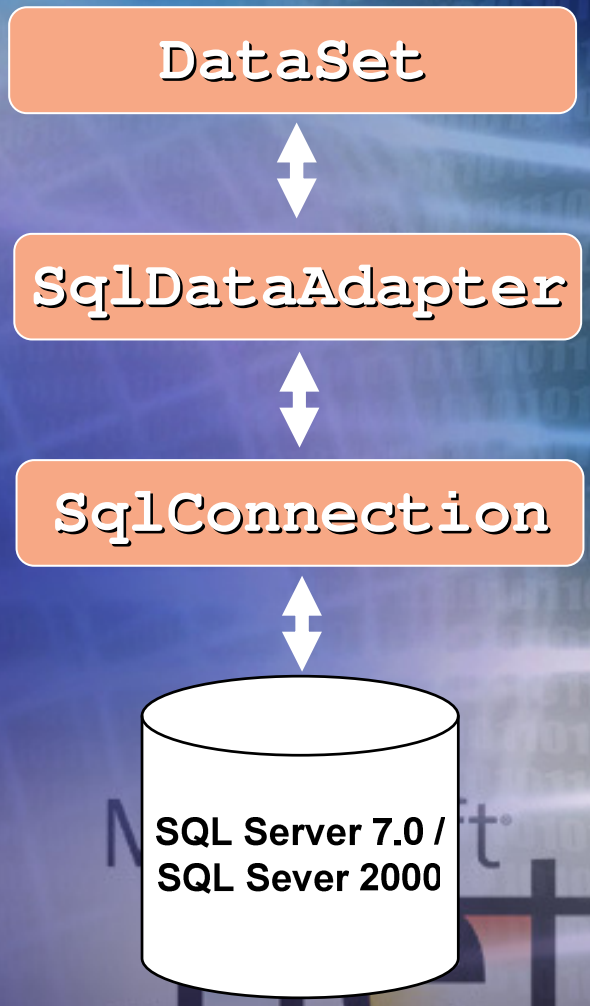
ADO.NET в свързана среда

- Данните са на сървъра до затваряне на връзката
  1. Отваряне на връзка (`SqlConnection`)
  2. Изпълнение на команда / команди (`SqlCommand`)
  3. Обработка на редовете получени като резултат от заявката чрез четец (`SqlDataReader`)
  4. Затваряне на четеца
  5. Затваряне на връзката



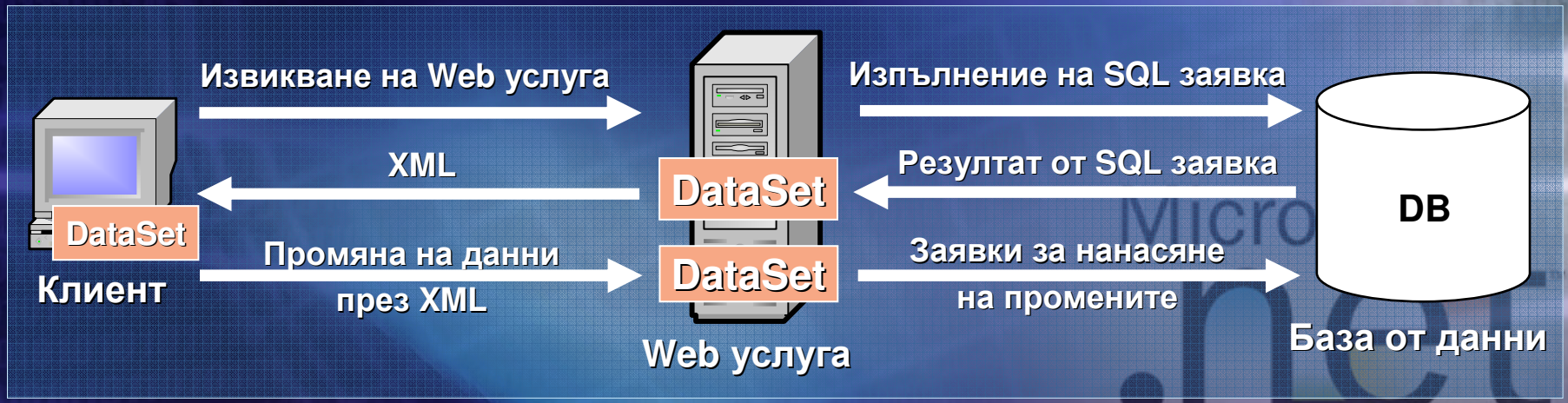
ADO.NET в несвързана среда

- Данните се кешират в `DataSet` обект и връзката се преустановява
  1. Отваряне на връзка (`SqlConnection`)
  2. Пълнене на `DataSet` (чрез `SqlDataAdapter`)
  3. Затваряне на връзката
  4. Работа със `DataSet`-а
  5. Отваряне на връзка
  6. Нанасяне на промени по данните по сървъра
  7. Затваряне на връзката



ADO.NET, XML и Web услуги

- ADO.NET е тясно интегрирано с XML
  - Често се използва в несвързан сценарий посредством Web услуги
  - Web-услугата реализира бизнес слоя на трислойните приложения
    - Извършва бизнес операциите над данните



Какво е XML?

- XML е:
  - универсален език (нотация) за описание на структурирани данни
  - данните се съхраняват заедно с мета-информация за тях
  - прилича на HTML – текстово-базиран, използва тагове и атрибути
  - с него се описват други езици (формати) за представяне на данни
  - световно-утвърден стандарт, поддържан от W3C (www.w3c.org)
  - независим от платформата, езиците за програмиране и операционната система

XML – пример

```
<?xml version="1.0"?>
<library name=".NET Developer's Library">
    <book>
        <title>Programming Microsoft .NET</title>
        <author>Jeff Prosise</author>
        <isbn>0-7356-1376-1</isbn>
    </book>
    <book>
        <title>Microsoft .NET for Programmers</title>
        <author>Fergal Grimes</author>
        <isbn>1-930110-19-7</isbn>
    </book>
</library>
```

XML – пример

```
<?xml version="1.0"?>
<library name=".NET Developer's Library">
    <book>
        <title>Programming Microsoft .NET</title>
        <author>Jeff Prosise</author>
        <isbn>0-7356-1376-1</isbn>
    </book>
    <book>
        <title>Microsoft .NET for Programmers</title>
        <author>Fergal Grimes</author>
        <isbn>1-930110-19-7</isbn>
    </book>
</library>
```

заглавна част (пролог)

атрибут

отварящ таг

елемент

затварящ таг

стойност на елемент

XML и HTML

- Прилики между езиците XML и HTML:
  - и двата са текстово базирани
  - използват тагове и атрибути
- Разлики между езиците XML и HTML:
  - HTML е език, а XML е синтаксис за описание на други езици
  - HTML описва форматирането на информацията, а XML описва структурирана информация
  - XML изисква документите да са добре дефинирани (well-formed)

Добре дефинирани документи

- XML изисква документите да са добре дефинирани (well-formed):
  - таговете винаги да се затварят и то в правилния ред (да не се застъпват)
  - атрибутите винаги да се затварят
  - да има само един основен root елемент
  - ограничения върху имената на таговете и атрибутите
- Пример за лошо-дефиниран документ:

```
<xml>
    <button bug! value="OK name="b1">
        <animation source="demo1.avi"> 1 < 2 < 3
    </click-button>
< / xml >
```

Кога се използва XML?

- XML се използва:
  - за обмяна на информация между различни системи
  - за съхранение на структурирани данни
  - за създаване на собствени езици за описание на информация
- Недостатъци на XML:
  - данните са по-обемисти
  - намалена производителност
  - често пъти е необходима много памет
  - увеличаване на мрежовия трафик

Пространства от имена

- Пространствата от имена (namespaces) в XML документите позволяват дефиниране и използване на тагове с еднакви имена:

```
<?xml version="1.0" encoding="UTF-8"?>
<country:towns xmlns:country="urn:address-com:country"
        xmlns:town="http://www.address.com/town">
    <town:town>
        <town:name>Sofia</town:name>
        <town:population>1 200 000</town:population>
        <country:name>Bulgaria</country:name>
    </town:town>
    <town:town>
        <town:name>Plovdiv</town:name>
        <town:population>700 000</town:population>
        <country:name>Bulgaria</country:name>
    </town:town>
</country:towns>
```

Пространства от имена

```
<?xml version="1.0" encoding="UTF-8"?>
<country:towns xmlns:country="urn:address-com:country"
       xmlns:town="http://www.address.com/town">
    <town:town>
        <town:name>S
        <town:popula
        <country:name>Bulgaria</country:name>
    </town:town>
    <town:town>
        <town:name
        <town:popula>700 000</town:population>
        <country:name>Bulgaria</country:name>
    </town:town>
</country:towns>
```

пространство с префикс "country" и URI идентификатор "urn:address-com:country"

таг с име "name" от пространството "country", т.е. пълното име на тага е "urn: address-com:country:name"

Пространства от имена

- Позволява се използването на пространства по подразбиране:

```
<?xml version="1.0" encoding="windows-1251"?>
<order xmlns="http://www.hranitelni-stoki.com/orders">
    <item>
        <name>бира "Загорка"</name>
        <ammount>8</ammount>
        <measure>бутилка</measure>
        <price>3.76</price>
    </item>
    <item>
        <name>кебапчета</name>
        <ammount>...</ammount>
        <measure>...</measure>
        <price>...</price>
    </item>
</order>
```

пространство по подразбиране

пълното име на тага "item" е "http://www.hranitelnik-stoki.com:item"

XML парсери

- XML парсерите са програмни библиотеки, които улесняват работата с XML
- Служат за:
  - извличане на данни от XML документи
  - построяване на XML документи
  - валидация на XML документи по дадена схема
- По начин на работа биват:
  - DOM (Document Object Model) – представя XML документите като дърво в паметта и позволява лесна обработка
  - SAX (Simple API for XML Processing) – чете XML документите последователно като поток и позволява анализиране на съдържанието им

XML и .NET Framework

- В .NET Framework:
  - Средствата за работа с XML се намират в пространството `System.Xml`
  - Има пълна реализация на DOM модела
    - чрез класовете `XmlDocument`, `XmlNode`, ...
    - XML документът се зарежда целият като дърво в паметта и след това се обработва
  - Има класове, с функционалност подобна на SAX, но няма чиста SAX имплементация
    - класовете `XmlReader` и `XmlWriter` четат и пишат XML документи последователно елемент по елемент

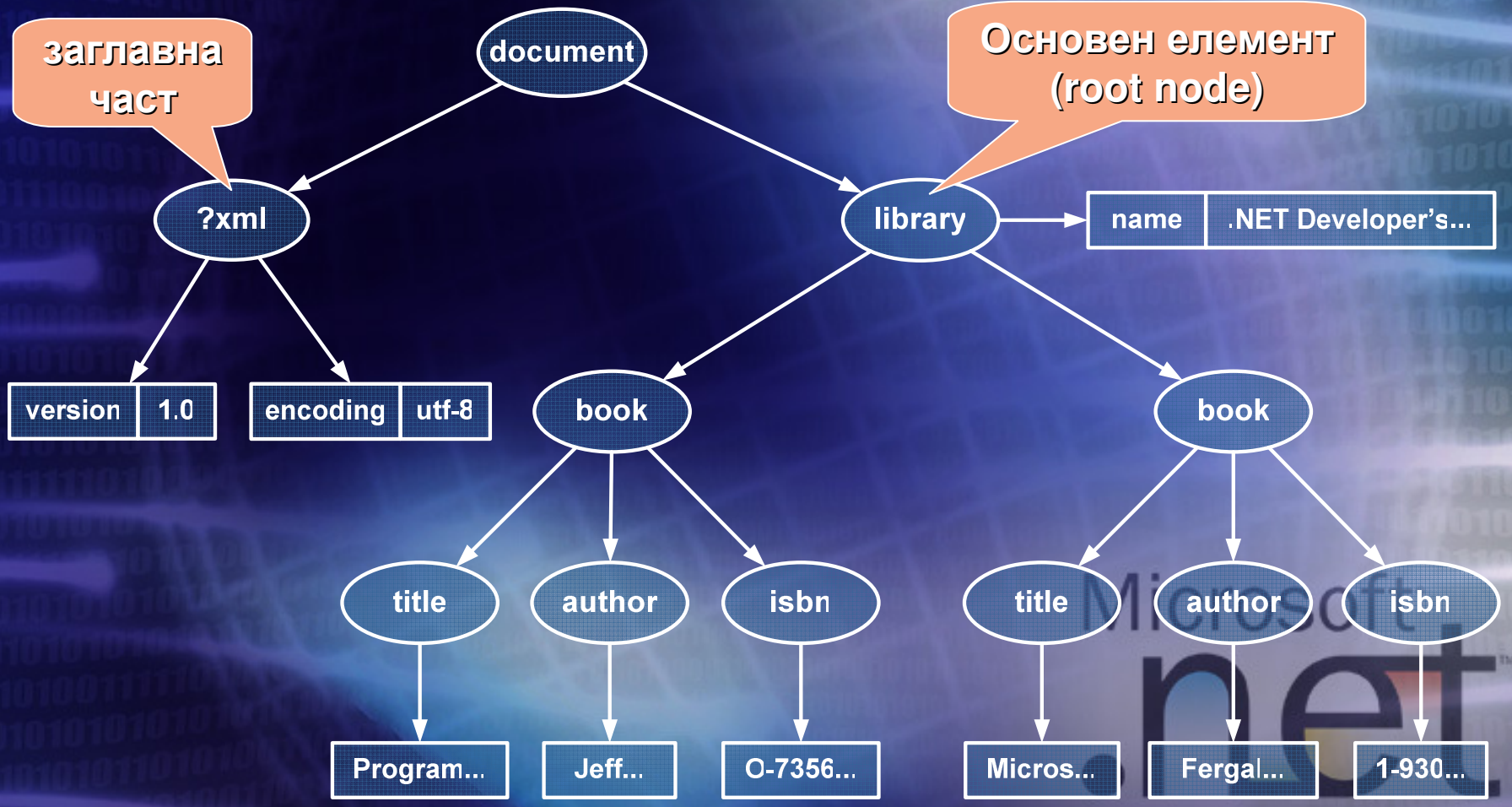
Работа с DOM парсера

- Даден е следния XML документ:

```
<?xml version="1.0"?>
<library name=".NET Developer's Library">
    <book>
        <title>Programming Microsoft .NET</title>
        <author>Jeff Prosise</author>
        <isbn>0-7356-1376-1</isbn>
    </book>
    <book>
        <title>Microsoft .NET for Programmers</title>
        <author>Fergal Grimes</author>
        <isbn>1-930110-19-7</isbn>
    </book>
</library>
```

Работа с DOM парсера

- Този документ се представя в паметта като DOM дърво по следния начин:



DOM парсер – пример

```
XmlDocument doc = new XmlDocument();
doc.Load("library.xml");

XmlNode rootNode = doc.DocumentElement;
Console.WriteLine("Root node: {0}", rootNode.Name);

foreach (XmlAttribute atr in rootNode.Attributes)
{
    Console.WriteLine("Attribute: {0}={1}",
        atr.Name, atr.Value);
}

foreach (XmlNode node in rootNode.ChildNodes)
{
    Console.WriteLine("\nBook title = {0}",
        node["title"].InnerText);
    Console.WriteLine("Book author = {0}",
        node["author"].InnerText);
    Console.WriteLine("Book isbn = {0}",
        node["isbn"].InnerText);
}
```

Класовете за работа с DOM

- За работа с DOM се използват класовете:
  - `XmlNode` – абстрактен базов клас за всички възли в едно DOM дърво
  - `XmlDocument` – съответства на корена на DOM дърво, обикновено съдържа два наследника:
    - заглавна част (пролог) на XML документа
    - елемент-корен на XML документа
  - `XmlElement` – представя XML елемент
  - `XmlAttribute` – представя атрибут на XML елемент (двойка име-стойност)
  - `XmlAttributeCollection` – списък от XML атрибути
  - `XmlNodeList` – списък от възли в DOM дърво

Класът XmlNode

- Класът `System.Xml.XmlNode`:
  - е основополагащ при работата с DOM
  - представлява базов възел, а неговите наследници са типовете DOM възли:
    - `XmlDocument`, `XmlElement`, `XmlAttribute`, …
  - позволява навигация в DOM дървото:
    - `ParentNode` – връща възела-родител (или `null` ако няма)
    - `PreviousSibling` / `NextSibling` – връща левия / десния съсед на текущия възел
    - `FirstChild` / `LastChild` – връща първия / последния наследник на текущия възел
    - `Item` (индексатор в C#) – връща наследник на текущия възел по името му

Класът `XmlNode`

- Класът `System.Xml.XmlNode`:
  - позволява работа с текущия възел:
    - `Name` – връща името на възела (име на елемент, атрибут, ...)
    - `Value` – стойността на възела
    - `Attributes` – списък от атрибутите на възела (като `XmlAttributeCollection`)
    - `HasChildNodes` – дали има наследници
    - `InnerXml`, `OuterXml` – връща частта от XML документа, която описва съдържанието на възела съответно с и без него самия
    - `InnerText` – конкатенация от стойностите на възела и наследниците му рекурсивно
    - `NodeType` – връща типа на възела

Класът `XmlNode`

- Класът `System.Xml.XmlNode` позволява промяна на текущия възел:
  - `AppendChild(…)` / `PrependChild(…)` – добавя нов наследник след / преди всички други наследници на текущия възел
  - `InsertBefore(…)` / `InsertAfter(…)` – вмъква нов наследник преди / след указан наследник
  - `RemoveChild(…)` / `ReplaceChild(…)` – премахва / заменя указания наследник
  - `RemoveAll()` – изтрива всички наследници на текущия възел (атрибути, елементи, ...)
  - `Value`, `InnerText`, `InnerXml` – променя стойността / текста / XML текста на възела

Класът XmlDocument

- Класът `System.Xml.XmlDocument`:
  - съдържа XML документ във вид на DOM дърво
  - позволява зареждане и съхранение на XML документи от/във файл, поток или символен низ (вж. `Load(…)`, `LoadXml(…)`, `Save(…)`)
- По-важни свойства, методи и събития:
  - `DocumentElement` – извлича елемента-корен
  - `PreserveWhitespace` – указва дали празното пространство да бъде запазено при зареждане/записване на документа
  - `CreateElement(…)`, `CreateAttribute(…)`, `CreateTextNode(…)` – създава нов XML елемент, атрибут или стойност на елемент
  - `NodeChanged`, `NodeInserted`, `NodeRemoved` – събития за следене за промени в документа

SAX парсери и `XmlReader`

- В .NET Framework няма имплементация на класически SAX парсер
- Класическите SAX парсери:
  - обхождат документа последователно
  - извикват callback функции при срещане на определени възли
- В .NET последователната обработка на XML документи се извършва с `XmlReader`
- `XmlReader` е абстрактен клас, който:
  - предоставя еднопосочен достъп само за четене до XML данни
  - работи като поток, но чете XML документи
  - прочетените на всяка стъпка данни могат да се извличат и анализират от програмиста

Класът XmlReader

- XmlReader е абстрактен клас
- За работа с него се използват неговите наследници:
  - XmlTextReader – за четене от файл или поток
  - XmlNodeReader – за четене от възел в DOM дърво
  - XmlValidatingReader – за валидация по XSD, DTD или XDR схема при четене от друг XmlReader
- Начин на използване:

```
XmlTextReader reader = new XmlTextReader("some-file.xml");
while (reader.Read()) {
   // Analyze the read node
}
```

Работа с XmlWriter

- Класът `XmlWriter` позволява създаване на XML документи
- Работи като поток, но пише в XML документи
- `XmlWriter` предлага следните методи:
  - `WriteStartDocument()` – добавя пролог частта в началото на документа (`<?xml …`)
  - `WriteStartElement(…)` – добавя отварящ таг
  - `WriteEndElement()` – затваря последния таг
  - `WriteAttributeString(…)` – добавя атрибут в текущия елемент
  - `WriteElementString(…)` – добавя елемент по зададено име и текстова стойност
  - `WriteEndDocument()` – затваря всички тагове и изчиства вътрешните буфери

Работа с XmlWriter

- `XmlWriter` е абстрактен клас и не се инстанцира директно
- `XmlTextWriter` пише XML във файлове и потоци
  - Има допълнителни методи и свойства:
    - `Format`, `Indentation`, `IndentChar` – задават настройките за отместване навътре на вложените тагове
    - `QuoteChar` – задава символа за отделяне на стойностите на атрибутите
  - В конструктора може да се задава кодирането, което да се използва
  - По подразбиране се използва UTF-8

Кога да използваме DOM и SAX?

- Моделът за обработка на XML документи DOM (`XmlDocument`) е подходящ, когато:
  - обработваме малки по обем документи
  - нуждаем се от гъвкавост при навигацията
  - имаме нужда от пряк достъп до отделните възли на документа
  - желаем да променяме документа
- Моделът за обработка на XML документи SAX (`XmlReader`) е подходящ когато:
  - обработваме големи по обем документи
  - скоростта на обработка е важна
  - не е необходимо да променяме възлите на документа

Domain-Driven Design (DDD)

- DDD (наричан още *проблемно-ориентиран подход*) – набор от принципи и схеми за създаване на оптимално организирани системи от обекти;
- Свежда се към създаване на програмни абстракции, наричани *модели на предметната област*;
- Към тях принадлежи *бизнес логиката*, установяваща връзка между програмния код и реалната област на приложение на продукта

Domain-Driven Design (DDD)

- DDD не е конкретна технология или методология, а набор от правила, позволяващи да се вземе правилно проектно решение;
- Този подход позволява значително да се ускори процеса на проектиране на софтуер в непозната предметна област;

Microsoft .net

Domain-Driven Design (DDD)

- DDD е особено полезен в случаите, когато разработчикът не е специалист в предметната област, за която се прави софтуера;
- Пример: програмистът не може да е добре запознат с всички области, за които се разработва софтуер. Но с правилно представена структура, чрез DDD може по-лесно да се проектира приложение, основано на ключови моменти и знания от предметната област;

Основни определения

- **Област (domain)** – предметната област, в която се прилага разработвания софтуер;
- **Модел (model)** – описва отделните аспекти на областта и може да се използва за решаване на проблема;
- **Език за описание** – език, структуриран около описанието на модела, използван от всички членове на екипа за свързване на техните активности по проекта.

DDD концепции

- **Ограничена свързаност (bounded context) –** използване на няколко модела на различни нива на проекта. Този подход се ползва за намаляване на връзките между моделите, което изключва сложността и объркаността на кода. Понякога е неясно в какъв именно контекст следва да бъде ползван даден модел.
- **Решение:** Точно да се определи контекста, в който се ползва модела. Да се определят границите на използване и неговите характеристики.

DDD концепции

- **Цялостност –** когато над проекта работят много хора, има тенденция моделът да се раздроби на по-малки фрагменти, което постепенно води до загуба на цялостта на проекта
- **Решение:** постоянно обединяване на парчетата код от отделните разработчици и обща проверка посредством тестване

DDD концепции

- **Взаимосвързаност** – при работа с няколко отделни модела в голяма група, някои членове на екипа може да не знаят същността на моделите на другите, което затруднява процеса на обединяване на проекта.
- **Решение:** на етапа на проектиране точно да се определи какво именно изпълнява даден модел и как е свързан с другите модели. Като краен резултат трябва да се получи карта на взаимовръзките на проекта.

Web с .NET

Въпроси?