

# 1. XML.NET

XML(Extensible Markup Language)은 W3C에서 권장하는 확장 가능한 마크업 언어입니다. XML은 주로 다른 시스템간의 데이터를 쉽게 주고 받을 수 있게 하여 데이터 표준화에 사용할 수 있습니다.

마크업 언어란 내용만 문서에 보관하는 것이 아니라 문서에 보관한 내용이 어떠한 의미를 지니고 있는지 마크를 지정할 수 있게 메타 정보도 표현하는 언어를 말합니다.

이러한 XML은 다양한 언어와 서비스에서 데이터를 주고 받을 수 있게 다양한 기술을 제공하고 있는데 여기에서는 .NET Framework에서 제공하는 다양한 XML 기술을 소개합니다.

.NET Framework에서는 다양한 XML 기술을 제공하고 있습니다. XML.NET 기술을 이용하면 높은 생산성과 표준 기반의 데이터 표준화 및 ADO.NET 기술 등을 효과적으로 사용할 수 있습니다.

XML은 여러분이 아시는 것처럼 W3C 표준으로 XML.NET에서도 W3C에서 권장하는 XML과 스키마, DOM 등의 표준을 따르고 있습니다. 이는 다른 플랫폼에서 운용하는 서비스와 상호 운용성을 보장함을 의미하고 서로 다른 시스템 간의 데이터 표준화할 수 있음을 의미합니다.

XML.NET에서는 정방향으로 빠르게 XML 스키마 및 DTD에 대해 유효성 검사를 수행할 수 있는 XmlReader를 제공하고 있습니다. XmlReader는 표준을 준수하는 XML 파서라고 말할 수 있습니다.

그리고 .NET Framework에서는 XSD 스키마를 프로그래밍 방식으로 빌드할 수 있게 스키마 개체 모델(SOM)을 제공하고 있으며 SmlSchema 클래스를 사용할 수 있습니다. 또한 이러한 스키마를 로딩하거나 저장하기 쉽게 XmlReader와 XmlWriter 클래스를 제공하고 있습니다.

그리고 효과적으로 문서 개체 모델(DOM)을 이용할 수 있게 XmlDocument 클래스를 제공하고 있습니다. 이러한 클래스들 외에도 다양한 클래스를 제공하여 효과적인 작업을 할 수 있습니다.

참고로 이 책에서는 .NET Framework 4를 기반으로 기술하고 있으니 참고하시기 바랍니다.

## 1. 1 XML 문서 구조

XML 문서는 선언부와 루트 요소로 구성하고 있습니다.

선언부는 <?xml로 시작하여 ?>로 끝납니다. 그리고 이처럼 XML 문서에서 <?로 시작해서 ?>로 끝나는 구문을 처리 구문(Processing Instruction)이라 부릅니다.

<?xml	version="1.0"	encoding="euc-kr"	standalone="yes" ?>
1	2	3	4

[그림 1.1] XML 선언부

선언부에는 버전 정보를 필수적으로 입력해야 하고 인코딩 정보나 의존성 정보는 선택적으로 나타낼 수 있습니다. 인코딩 정보는 사용할 언어 코드에 관한 인코딩 정보이며 의존성 정보는 해당 컴퓨터 내에서만 사용할 것인지 여부로 "yes" 혹은 "no"를 사용할 수 있습니다.

XML 문서에 주석을 표현할 때는 <!-- 로 시작해서 -->로 끝나는 사이에 표현합니다.

XML 문서에 요소는 <요소명> 요소 내용 </요소명> 형태로 나타냅니다. 요소명은 숫자나 밑줄이나 XML로 시작할 수 없고 공백 문자를 포함할 수 없습니다. XML 문서에 최상위 요소는 하나만 둘 수 있으며 모든 요소는 최상위 요소 내에 안전한 구조로 작성해야 합니다.

<?xml version="1.0" encoding="euc-kr" standalone="yes" ?>	선언문
<회원정보>	요소시작
<회원아이디>ehclub</회원아이디>	
<회원이름>장문석</회원이름>	
<회원아이디>jejutour</회원아이디>	
<회원이름>송정수</회원이름>	
</회원정보>	요소의 끝

[그림 1.2] XML 요소

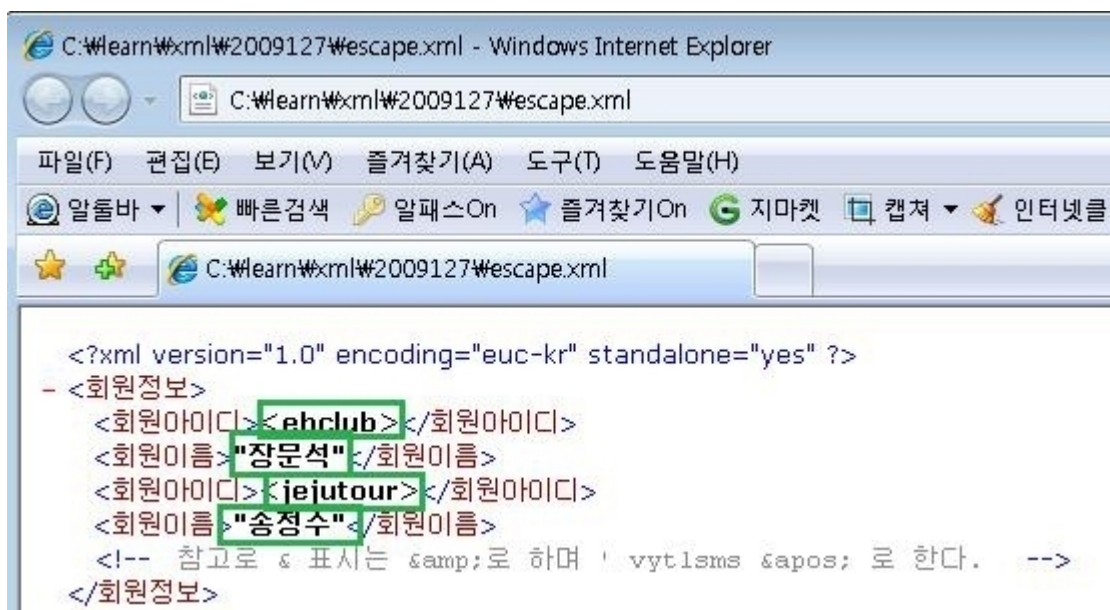
XML 문서에는 일부 문자는 약속한 형태로 표현하는데 이를 escape 문자라 부릅니다.

문자	표현	문자	표현	문자	표현
<	&lt;	'	&apos;	&	&amp;
>	&gt;	"	&quot;		

[표 1.1] escape 문자

```
<?xml version="1.0" encoding="euc-kr" standalone="yes" ?>
<회원정보>
  <회원아이디>&lt;ehclub&gt;</회원아이디>
  <회원이름>&quot;장문석&quot;</회원이름>
  <회원아이디>&lt;jejutour&gt;</회원아이디>
  <회원이름>&quot;송정수&quot;</회원이름>
  <!-- 참고로 & 표시는 &amp;로 하며 ' vtlsms &apos; 로 한다. -->
</회원정보>
```

[문서 1.1] escape 문자를 포함한 XML 문서

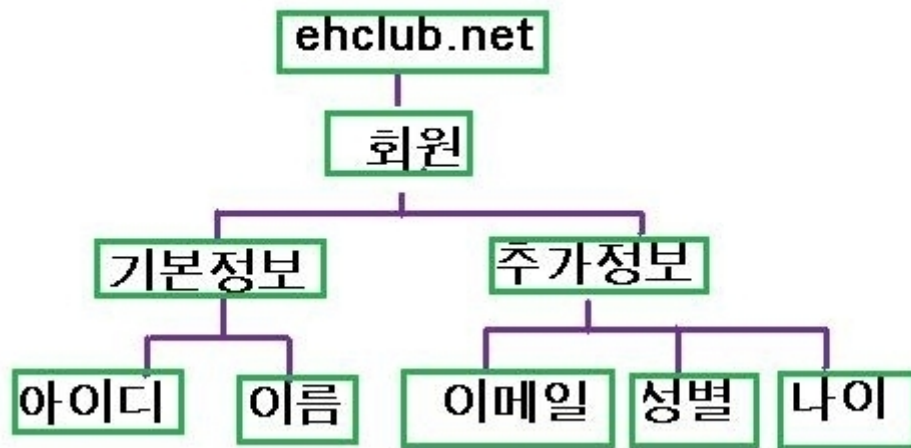


[그림 1.3] [문서 1.1]을 익스플로러로 본 화면

XML 문서를 작성할 때는 먼저 작성할 개체의 요소를 파악하고 분석합니다. 그리고 요소를 추상화한 형태로 작성하고 요소명과 특성명을 구분해서 정의합니다. 이러한 과정을 거친 후에 XML 문서를 작성하세요.

회원 정보를 XML로 표현해 봅시다. 회원의 정보는 아이디, 이름, 이메일, 성별, 나이를 표현하기로 할게요.

먼저 관리할 데이터를 추상화하기로 할게요.



[그림 1.4] 추상화

요소명과 특성명을 구분해서 정의합니다. 간단하게 XML을 정의할 때는 특성을 사용하지 않고 요소만 사용하면 전체 문서가 간단해지고 가독성이 높아집니다.

여기에서는 루트 요소명을 회원 정보로 정의하고 회원 정보 내 자식 요소를 회원 요소라 정할게요. 그리고 회원 요소에는 기본 정보와 추가 정보로 분류하고 기본 정보에는 아이디와 이름을 추가 정보에는 이메일과 성별, 나이를 두기로 할게요.

마지막으로 XML 문서를 작성해 볼게요.

```
<?xml version="1.0" encoding="euc-kr" standalone="yes" ?>
```

```
<회원정보>
```

```
<회원>
```

```
<기본정보>
```

```
<회원아이디>ehclub</회원아이디>
```

```
<회원이름>장문석</회원이름>
```

```
</기본정보>
```

```
<추가정보>
```

```
<이메일>jejutour@daum.net</이메일>
```

```
<나이>38</나이>
```

```
<성별>남</성별>
```

```
</추가정보>
```

```
</회원>
```

```
<회원>
```

```
<기본정보>
```

```
<회원아이디>jejutour</회원아이디>
```

```
<회원이름>송정수</회원이름>
```

```
</기본정보>
```

```
<추가정보>
```

```
<이메일>jejutour@daum.net</이메일>
```

```
<나이>38</나이>
```

```
<성별>남</성별>
```

```
</추가정보>
```

```
</회원>
```

```
</회원정보>
```

[문서 1.2] 회원 정보 XML 문서

특성을 표현할 때는 요소 태그 내부에 특성명="특성값" 형태로 표현합니다.

```
<회원 나이="38" 성별="남">홍길동</회원>
```

## 1. 2 DTD

특정 시스템 내부에서 자신의 시스템에서 사용할 수 있는 유효한 XML 문서 구조를 정의할 때 DTD를 사용할 수 있습니다. 데이터 표준화를 위해 문서 구조를 정의할 때는 DTD 보다 스키마를 선호하지만 하나의 시스템 내부에서만 유효한 데이터 문서인지 구별하여 사용할 때는 DTD 문서를 사용하기도 합니다.

DTD는 Documents Type Definition의 약자로 XML 문서를 표준 문서 포맷으로 추상화하는 것을 말합니다. 그리고 DTD 정의에 맞게 작성한 XML 문서를 유효화 문서(Valid Document)라 부릅니다.

DTD 문서는 <!DOCTYPE DTD 명 [내부 요소들]> 형태로 표시합니다.

그리고 내부 요소는 <!ELEMENT 요소명(내부요소, 내부요소, ...)> 형태로 표시합니다. 만약 내부 요소의 값이 문자형일 때는 #PCDATA 를 명시합니다.

```
<?xml version="1.0" encoding="euc-kr" ?>
<!-- DTD 정의 -->
<!DOCTYPE 회원 [
  <!ELEMENT 회원리스트 (회원)>
    <!ELEMENT 회원 (아이디,이름,이메일)>
      <!ELEMENT 아이디 (#PCDATA)>
      <!ELEMENT 이름 (#PCDATA)>
      <!ELEMENT 이메일 (#PCDATA)>
    ]>
  <!-- DATA 정의 -->
  <회원리스트>
    <회원>
      <아이디>ehclub</아이디>
      <이름>장문석</이름>
      <이메일>jejutour@daum.net</이메일>
    </회원>
    <회원>
      <아이디>jejutour</아이디>
      <이름>송정수</이름>
      <이메일>jejutour@empal.com</이메일>
    </회원>
  </회원리스트>
```

[문서 1.3] XML 문서 내부에 DTD 표현

다음은 XML문서와 DTD문서를 분리한 예입니다.

```
<?xml version="1.0" encoding="euc-kr" ?>
<!ELEMENT 회원리스트 (회원)>
<!ELEMENT 회원 (아이디,이름,이메일)>
<!ELEMENT 아이디 (#PCDATA)>
<!ELEMENT 이름 (#PCDATA)>
<!ELEMENT 이메일 (#PCDATA)>
```

[문서 1.4] XML 문서 외부에 DTD 표현

```
<?xml version="1.0" encoding="euc-kr" ?>
<!DOCTYPE 회원 SYSTEM "회원.dtd">
<회원리스트>
  <회원>
    <아이디>ehclub</아이디>
    <이름>장문석</이름>
    <이메일>jejutour@daum.net</이메일>
  </회원>
  <회원>
    <아이디>jejutour</아이디>
    <이름>송정수</이름>
    <이메일>jejutour@empal.com</이메일>
  </회원>
</회원리스트>
```

[문서 1.5] [문서 1.5] 규격에 유효한 XML 문서

XML 문서는 외형적인 모습은 표현하지 않고 내용만을 표현합니다. 만약 외형적인 모습을 표현할 때는 CSS를 이용하세요.

```
bookname{
  font-family: "돋음체";
  font-size:20px;
  color:#0000ff;
}

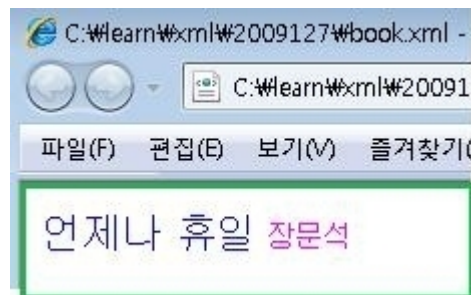
writer{
  font-family: "궁서체";
  font-size:14px;
  color:#ff00ff;
}

.codetype{
  font-family: "궁서체";
  font-size:14px;
  color:#00ff00
}
```

[문서 1.6] book.css 파일

```
<?xml version="1.0" encoding="euc-kr" standalone="yes"?>
<?xml:stylesheet type="text/css" href="book.css"?>
<book>
  <bookname>언제나 휴일</bookname>
  <writer>장문석</writer>
</book>
```

[문서 1.6] book.css 파일



[그림 1.5] CSS를 이용하여 시각화



DTD 문서 타입 선언은 다음과 같습니다.

```
<!DOCTYPE root_element source location1 location2 ... locationN [internal DTD]>
```

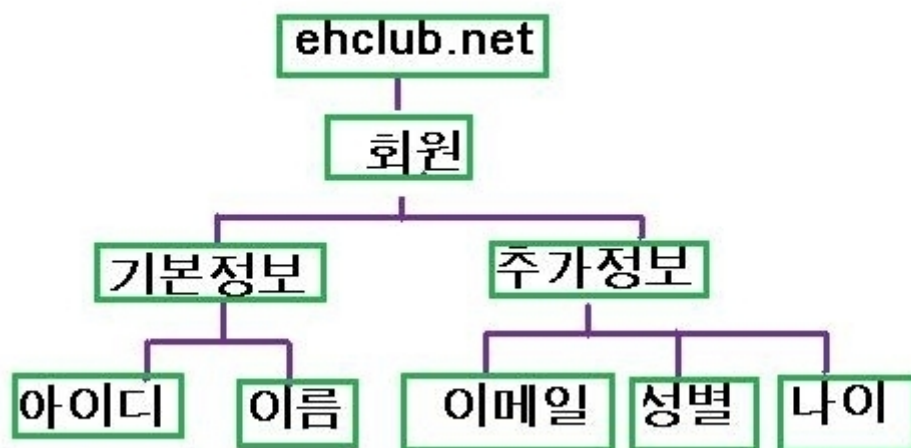
표현	설명설명
DOCTYPE	DTD 문서 정의 예약어
root_element	최상위 요소
source	SYSTEM 혹은 PUBLIC 키워드 사용
location#	외부 선언 DTD 파일 위치
[internal DTD]	내부에서 선언할 DTD 내용들

요소 선언은 다음과 같습니다.

```
<!ELEMENT element_name contents>
```

element\_name 요소 이름

contents 자식 목록들



[그림 1.6] DTD 예제 추상화

```

<!ELEMENT 회원 (기본정보,추가정보)>
<!ELEMENT 기본정보(아이디,이름)>
<!ELEMENT 아이디 (#PCDATA)>
<!ELEMENT 추가정보(이메일,성별,나이)>
<!ELEMENT 이메일 (#PCDATA)>
<!ELEMENT 성별 (#PCDATA)>
<!ELEMENT 나이 (#PCDATA)>
  
```

[문서 1.6] book.css 파일

요소 선언할 때 반복 횟수 및 그룹화는 다음처럼 표시합니다.

기호	의미	사용 예	설명
기호없음	1번	name	name이 한 번
,	순서대로	id,name	id와 name 요소 순으로
	or	id name	id 또는 name 요소
()	그룹화	(id,name)	그룹화
*	0번 이상 반복	meta*	meta 요소가 0번 이상 반복
+	1번 이상 반복	meta+	meta 요소가 1번 이상 반복
?	0번 혹은 1번	manager?	manager 요소가 0번 혹은 한 번

[표 1.2] 반복 횟수

<!ELEMENT element\_name ANY> : 어떠한 데이터 표현도 가능

<!ELEMENT element\_name EMPTY> : 데이터를 갖지 않음

특성 선언은 다음과 같이 표현합니다.

<!ATTLIST ele\_name, att\_name, att\_type, att\_defvalue>

표현	설명설명
ATTLIST	속성임을 나타내는 예약어
ele_name	요소 이름
att_name	특성 이름
att_type	열거형, 문자열, 토큰
att_defvalue	특성 초기값

[표 1.3] 특성 선언 표현

att\_type은 열거형, 문자열, 토큰 타입이 올 수 있고 토큰 타입은 예약어를 사용합니다.

att_type 종류	설명설명
문자열	CDATA 예약어 사용 DTD 정의: <!ATTLIST member birthday CATA "72-08-20"> 사용 예: <member birthday="72-08-20">홍길동</member>
열거형	DTD내에 열거한 값 중 하나 선택 DTD 정의: <!ATTLIST member major(cs,es,korea)#IMPLIED> 사용 예: <member major="korea">홍길동</member>
ENTITY, ENTITIES	내부 개체를 값으로 둘 수 있다. DTD정의: <!ATTLIST member picture ENTITY default> 사용 예: <member picture="image1">홍길동</member>
ID	요소 식별값으로 유일해야 한다. 숫자만으로 구성할 수 없음 DTD 정의: <!ATTLIST member idnum ID default> 사용 예: <member idnum="jejutour">홍길동</member>
IDREF, IDREFS	내부에 ID를 참조
NMTOKEN, NMTOKENS	문자열이 값으로 오며 공백을 허용하지 않음 DTD 정의: <!ATTLIST member credit NMTOKEN default> 사용 예: <member credit="720820-1234567">홍길동</member>

[표 1.4] 특성 타입

## 1. 3 XML 스키마

스키마는 DTD처럼 XML문서의 구조를 표현하기 위해서 사용합니다.

W3C에서는 DTD나 XML 스키마를 정의하고 이것에 맞게 XML을 작성하는 것을 Well Formed XML이라고 말합니다.

DTD는 하나의 시스템 내에서 사용할 XML 데이터의 구조를 정의하여 유효성을 점검할 때 사용하며 스키마는 서로 다른 시스템 사이의 데이터를 주고 받아 사용할 수 있게 데이터 표준화를 위해 제공하고 있습니다.

프로그램 방식으로 데이터를 처리할 때는 명확하게 문서의 구조를 정의하고 이를 기반으로 데이터 소스를 목적에 맞게 사용할 때 DTD나 스키마를 통해 안정성을 제공할 수 있습니다.

DTD로 XML 문서 구조를 정의하면 표준화에 사용하기 힘든 이유는 다음과 같습니다.

DTD는 XML 문서 구조와 다른 문법을 사용하므로 별도의 처리 시스템이 필요합니다.

DTD는 네임 스페이스를 제공하지 않습니다.

DTD는 하나의 문서에만 적용할 수 있습니다.

DTD는 지원하는 데이터 타입의 종류가 한정되어 있습니다.

DTD는 상속 개념을 제공하지 않습니다.

DTD는 XML문법과 달라서 DOM을 지원하지 않습니다.

이러한 한계점을 극복하기 위해 W3C에서는 XML 스키마 표준을 정의하고 이를 통해 XML 문서 구조를 정의할 수 있게 제공하고 있습니다.

	DTD	스키마
문법	XML과 유사한 문법	XML 1.0 표준에 만족
DOM 지원	DOM 기술을 지원하지 않음	DOM을 통한 조작 가능
컨텐츠 모델	순차, 선택 리스트만 제공	순차, 리스트 복합적으로 사용 가능
데이터 타입	문자열, 토큰, ID와 그 외에 한정된 타입 지원	문장려, 숫자, 날짜/시간 및 새로운 형식 정의를 할 수 있음
Namespace	전역 이름만 사용	전역/ 로컬 이름 모두 사용 가능
상속성	제공하지 않음	제공
확장성	한계가 있음	제한 없음
기본 제약 조건	있음	없음
동적 스키마	불가능 - 읽기만 가능	가능 - 런타임에 상호작용으로 변경 가능

[표 1.5] DTD와 스키마 비교

### 1.3.1 XML 스키마 문법

XML 스키마는 문서의 구조와 데이터를 기술하기 위하여 구조와 데이터 타입으로 나누어져 있습니다.

구조는 데이터 타입들로부터 만들어질 수 있는 조합을 의미합니다.

데이터 타입은 기본 데이터 타입과 파생 데이터 타입이 있습니다.

XML 스키마 선언은 <xsd:schema> 요소로 시작하며 최상위 요소는 항상 스키마여야 합니다.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns="schema.xsd"
            targetNamespace = "schema.xsd"
            elementFormDefault = "qualified">
```

특성	의미
xmlns:xsd	XML 스키마의 네임스페이스 지정
xmlns	예약되어 있는 키워드로 네임스페이스 정의
targetNamespace	스키마에 정의한 요소나 속성들을 다른 XML 문서에서 식별할 수 있게 네임스페이스 지정
elementFormDefault	elementFormDefault 값이 qualified일 때는 모든 요소는 targetNamespace에서 선언한 네임스페이스를 사용하고 unqualified일 때는 targetNamespace에서 선언한 네임스페이스를 사용하지 않아도 됨

[표 1.6] 스키마 선언에서 사용하는 특성

스키마에서 요소를 정의할 때는 element 태그를 이용해서 정의합니다. Type 속성을 이용하여 요소에 어떠한 데이터 형식을 포함할 수 있는지 나타낼 수 있습니다.

```
<?xml version="1.0" encoding="euc-kr"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="book" type="xsd:string"/>
</xsd:schema>
```

스키마에서 특성은 attribute 태그를 사용하여 정의합니다. 자세한 문법 사항은 W3C 권고안을 참고하세요.

```
<xsd:element name="price">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:int">
        <xsd:attribute name="단위" type="xsd:string" use="optional" default="원"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

다음은 책 목록을 표현하기 위한 스키마 파일의 내용과 책 목록 데이터를 표현한 파일의 내용입니다.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="urn:books-schema"
  elementFormDefault="qualified"
  targetNamespace="urn:books-schema">

  <xsd:element name="books" type="booksType"/>

  <xsd:complexType name="booksType">
    <xsd:sequence maxOccurs="unbounded">
      <xsd:element name="book" type="bookType"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="bookType">
    <xsd:sequence>
      <xsd:element name="title" type="xsd:string"/>
      <xsd:element name="price" type="xsd:int"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

[문서 1.1] data.xsd 문서 내용

```
<?xml version="1.0" encoding="utf-8"?>
<books xmlns="urn:books-schema">
  <book>
    <title>XML.NET 과 ADO.NET</title>
    <price>22000</price>
  </book>
</books>
```

[문서 1.2] data.xml 문서 내용

## 2. XmlWriter

XmlWriter 클래스는 추상 클래스로 정방향으로만 XML데이터를 쓸 수 있습니다.

XmlWriter 개체를 이용하면 파일이나 스트림 혹은 TextReader 및 문자열에 XML 데이터를 작성할 수 있습니다.

XML 표준 문서를 정방향으로 데이터를 판독하는 파서입니다. 정방향으로만 데이터를 판독하기 때문에 메모리 캐시를 사용하지 않으며 빠른 성능을 제공합니다.

▷ 클래스 상속 계층

System.Object

System.Xml.XmlWriter

System.Xml.XmlDictionaryWriter

System.Xml.XmlTextWriter

System.Xml.Xsl.Runtime.XmlQueryOutput

▷ 네임스페이스: System.Xml

▷ 어셈블리: System.Xml(System.Xml.dll)

### 2. 1 XmlWriter 개체 만들기

XmlWriter 개체는 XmlWriter 클래스의 정적 메서드인 Create를 호출하여 만듭니다. Create 메서드를 이용하여 XmlWriter 개체를 생성할 때는 출력할 대상을 파일이나 스트림, TextWriter, XmlWriter로 지정할 수 있으며 문자 검사 및 세부 규칙 등을 설정한 XmlWriterSettings 개체를 추가로 전달할 수 있습니다.

▷ Create 메서드 중복 정의

Create(Stream)

Create(Stream, XmlWriterSettings)

Create(String)

Create(String, XmlWriterSettings)

Create(StringBuilder)

Create(StringBuilder, XmlWriterSettings)

Create(TextWriter)

Create(TextWriter, XmlWriterSettings)

Create(XmlWriter)

Create(XmlWriter, XmlWriterSettings)

### 2.1.1 XmlWriterSettings

XmlWriter 개체를 만들 때 문자 검사 및 세부 규칙을 정할 때 XmlWriterSettings 개체를 이용할 수 있습니다.

XmlWriterSettings 개체는 들여쓰기 규칙이나 인코딩 형식 및 줄 바꿈에 사용할 문자, XML 선언을 작성할 것인지 여부 등을 지정할 수 있는 속성을 제공하고 있습니다.

속성	초기값	설명
CheckCharacters	true	문자 검사 수행 여부 ( <a href="http://www.w3.org/TR/REC-xml#charsets">http://www.w3.org/TR/REC-xml#charsets</a> )
CloseOutput	false	Close 메서드 호출할 때 XmlWriter가 내부 스트림 혹은 TextWriter도 함께 닫을지 여부
ConformanceLevel	ConformanceLevel.Document	규칙 수준 (ConformanceLevel 값 중 하나를 지정)
Encoding	Encoding.UTF8	사용할 인코딩 형식
Indent	false	요소의 들여쓰기 여부
IndentChars	" "(공백 두 개)	들여쓰기에 사용할 문자열
NamespaceHandling	Default	중복 네임스페이스 선언을 제거할 지 여부 (NamespaceHandling 열거형)
NewLineChars	"\r\n"	줄 바꿈에 사용할 문자열
NewLineHandling	Replace	줄 바꿈을 정규화할 지 여부 (NewLineHandling 값 중 하나)
NewLineOnAttributes	false	특성을 새 줄에 쓸지 여부
OmitXmlDeclaration	false	XML 선언을 생략할 지 여부
OutputMethod	Xml	출력을 직렬화하는데 사용할 메서드를 지정 (XmlOutputMethod 중 하나)

[표 2.1] XmlWriterSettings 속성



### 2.1.2 XmlWriter 개체 만들기 예제

다음 예제 코드는 자동 들여쓰기 규칙을 적용하여 XmlWriter 개체를 만들고 주석을 쓴 후에 닫는 간단한 예제입니다.

```
static void Main(string[] args)
{
    XmlWriterSettings xsettings = new XmlWriterSettings();
    xsettings.Indent = true;

    XmlWriter xwriter = XmlWriter.Create("data.xml", xsettings);
    xwriter.WriteComment("XmlWriter 개체 만들기 실습 예제");
    xwriter.WriteStartElement("books");
    xwriter.WriteStartElement("book");
    xwriter.WriteValue("ADO.NET");
    xwriter.WriteEndElement();
    xwriter.WriteStartElement("book");
    xwriter.WriteValue("XML.NET");
    xwriter.WriteEndElement();
    xwriter.WriteEndElement();
    xwriter.Close();
}
```

[소스 2.1] XmlWriter 개체 만들기 예제 코드

다음은 [소스 2.1]로 작성한 예제를 실행했을 때 만들어지는 "data.xml" 파일의 내용입니다.

```
<?xml version="1.0" encoding="utf-8"?>
<!--XmlWriter 개체 만들기 실습 예제-->
<books>
  <book>ADO.NET</book>
  <book>XML.NET</book>
</books>
```

[문서 2.1] [소스 2.1]을 실행했을 때 만들어지는 "data.xml" 파일 내용

## 2. 2 XmlWriter 개체로 XML 데이터 작성

XmlWriter 개체는 XML 데이터를 작성할 때 사용할 수 있는 다양한 Write 메서드를 제공하고 있습니다.

XmlWriter 개체에서 제공하는 Write 메서드의 종류에는 요소 쓰기, 특성 쓰기, 형식화된 데이터 쓰기 등이 있습니다.

### 2.2.1 요소 쓰기

XmlWriter 개체를 이용하여 요소를 쓸 때는 WriteStartElement, WriteElementString, WriteNode 메서드를 호출합니다.

WriteStartElement 메서드는 요소의 시작 태그를 쓸 때 사용합니다. 이 메서드를 이용하여 요소를 쓴 다음에는 요소의 끝을 쓰는 WriteEndElement 메서드를 호출해야 합니다.

```
public void WriteStartElement(string name);
public void WriteStartElement(string name, string ns);
public void WriteStartElement(string prefix, string name, string ns);
public void WriteEndElement();
```

WriteElementString 메서드는 문자열 값을 갖는 요소를 쓸 때 사용합니다.

```
public void WriteElementString(string name, string value);
public void WriteElementString (string name, string ns, string value);
public void WriteElementString (string prefix, string name, string ns, string value);
```

WriteNode 메서드는 XmlReader 개체 혹은 XPathNavigator 개체를 복사하여 쓰기 작업에 사용합니다.

```
public void WriteNode(XmlReader reader, bool defattr); //defattr: 기본 특성을 복사 여부
public void WriteNode(XPathNavigator navi, bool defattr);
```

다음의 예제 코드는 세 가지 방법으로 요소 쓰기를 사용하는 예제입니다. 먼저 "data.xml" 파일을 출력 파일로 설정한 XmlWriter 개체를 생성하여 books 루트 요소를 쓰고 두 개의 자식 요소 book을 쓰기 작업합니다. 첫 번째 book 요소의 자식 요소인 title을 쓸 때는 WriteStartElement 메서드를 이용하였고 두 번째 book 요소의 자식 요소인 title을 쓸 때는 WriteElementString 메서드를 이용하였습니다.

그리고 작성한 "data.xml" 파일을 소스 파일로 설정한 XmlReader 개체를 생성하고 Console.Out을 출력 파일로 설정한 XmlWriter 개체를 생성한 후에 XmlReader 개체를 이용하여 데이터를 복사하여 콘솔 화면에 출력할 때 WriteNode 메서드를 이용하였습니다.

```

static void Main(string[] args)
{
    XmlWriterSettings settings = new XmlWriterSettings();
    settings.Indent = true;
    XmlWriter writer = XmlWriter.Create("data.xml", settings);

    writer.WriteComment("XmlWriter 개체로 요소 쓰기");
    writer.WriteStartElement("books"); //루트 요소 쓰기

    writer.WriteStartElement("book");//book 요소 쓰기
    writer.WriteStartElement("title");//title 요소 쓰기
    writer.WriteName("XML.NET");
    writer.WriteEndElement();//title 요소 닫기
    writer.WriteStartElement("가격");//가격 요소 쓰기
    writer.SetValue(12000);
    writer.WriteEndElement();//가격 요소 닫기
    writer.WriteEndElement();//book 요소 닫기

    writer.WriteStartElement("book");//book 요소 쓰기
    writer.WriteElementString("title","ADO.NET");//title 요소와 값 쓰기
    writer.WriteStartElement("가격");//가격 요소 쓰기
    writer.SetValue(15000);
    writer.WriteEndElement();//가격 요소 닫기
    writer.WriteEndElement();//book 요소 닫기

    writer.WriteEndElement();//루트 요소 닫기
    writer.Close();

    XmlReader xreader = XmlReader.Create("data.xml"); //XmlReader 개체 생성
    XmlWriter xwriter = XmlWriter.Create(Console.Out); //콘솔 출력 스트림으로 XmlWriter 개체 생성
    xwriter.WriteNode(xreader, false); //xreader 개체로 읽어온 데이터를 xwriter 개체에 복사
    xwriter.Close();
    xreader.Close();
}

```

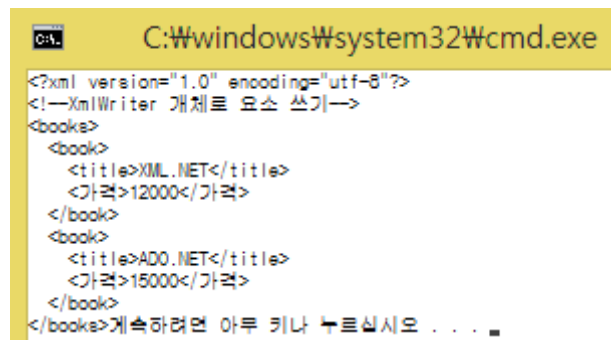
[소스 2.2] XmlWriter 개체로 요소 쓰기 예제 코드

```

<?xml version="1.0" encoding="utf-8"?>
<!--XmlWriter 개체로 요소 쓰기-->
<books>
  <book>
    <title>XML.NET</title>
    <가격>12000</가격>
  </book>
  <book>
    <title>ADO.NET</title>
    <가격>15000</가격>
  </book>
</books>

```

[문서 2.2] [소스 2.2]를 실행했을 때 만들어지는 "data.xml" 파일 내용



```

C:\windows\system32\cmd.exe
<?xml version="1.0" encoding="utf-8"?>
<!--XmlWriter 개체로 요소 쓰기-->
<books>
  <book>
    <title>XML.NET</title>
    <가격>12000</가격>
  </book>
  <book>
    <title>ADO.NET</title>
    <가격>15000</가격>
  </book>
</books>계속하려면 아무 키나 누르십시오 . . .

```

[그림 2.1] [소스 2.2] 실행 화면

### 2.2.2 특성 쓰기

요소의 특성을 쓸 때는 WriteStartElement, WriteAttributeString, WriteAttributes 메서드를 이용합니다.

WriteStartElement 메서드는 특성의 시작을 작성할 때 사용하는데 WriteEndElement 메서드를 이용하여 특성의 끝을 작성해야 합니다.

```
public void WriteStartElement (string name);  
public void WriteStartElement (string name, string ns);  
public void WriteStartElement (string prefix, string name, string ns);  
public void WriteEndElement ( );
```

WriteAttributeString 메서드를 이용하면 값이 있는 특성을 쉽게 작성할 수 있습니다.

```
public void WriteAttributeString (string name, string value);  
public void WriteAttributeString (string name, string ns, string value);  
public void WriteAttributeString (string prefix, string name, string ns, string value);
```

WriteAttributes 메서드를 이용하면 XmlReader의 현재 위치에 있는 모든 특성을 작성할 수 있습니다.

```
public void WriteAttributes (XmlReader reader, bool defattr);
```

다음의 예제 코드는 세 가지 방법으로 특성 쓰기를 사용하는 예제입니다. 먼저 "data.xml" 파일을 출력 파일로 설정한 XmlWriter 개체를 생성하여 books 루트 요소를 쓰고 두 개의 자식 요소 book을 쓰기 작업합니다. 첫 번째 book 요소의 title 특성을 쓸 때는 WriteStartElement 메서드를 이용하였습니다. 두 번째 book 요소의 title 특성을 쓸 때는 WriteAttributeString 메서드를 이용하였습니다.

그리고 작성한 "data.xml" 파일을 소스 파일로 설정한 XmlReader 개체를 생성하고 Console.Out을 출력 파일로 설정한 XmlWriter 개체를 생성한 후에 XmlReader 개체의 Read 메서드를 반복하여 호출하여 읽어온 노드 타입이 요소일 때 XmlReader 개체의 현재 위치에 있는 모든 특성 정보를 WriteAttributes 메서드를 이용하여 출력하였습니다.

```
static void Main(string[] args)  
{  
    XmlWriterSettings settings = new XmlWriterSettings();  
    settings.Indent = true;  
    XmlWriter writer = XmlWriter.Create("data.xml", settings);  
    writer.WriteComment("XmlWriter 개체로 특성 쓰기");  
    writer.WriteStartElement("books"); //루트 요소 쓰기  
    writer.WriteStartElement("book"); //book 요소 쓰기  
    writer.WriteStartElement("title"); //title 특성 쓰기  
    writer.WriteString("XML.NET"); //title 특성 값 쓰기  
    writer.WriteEndElement(); //title 특성 닫기  
    writer.WriteStartElement("가격"); //가격 특성 쓰기
```

```

writer.WriteValue(12000); //가격 특성 값 쓰기
writer.WriteEndAttribute(); //가격 특성 닫기
writer.WriteEndElement(); //book 요소 닫기

writer.WriteStartElement("book");//book 요소 쓰기
writer.WriteAttributeString("title", "ADO.NET");//title 특성과 값 쓰기
writer.WriteStartAttribute("가격");//가격 특성 쓰기
writer.WriteValue(15000);//가격 특성 값 쓰기
writer.WriteEndAttribute();//가격 특성 닫기
writer.WriteEndElement();//book 요소 닫기

writer.WriteEndElement();//루트 요소 닫기
writer.Close();

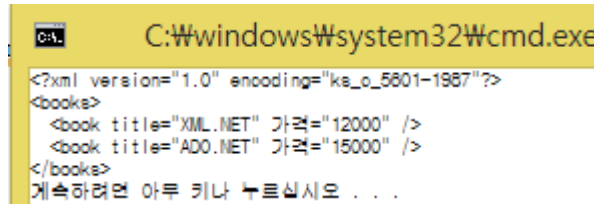
XmlReader xreader = XmlReader.Create("data.xml"); //XmlReader 개체 생성
XmlWriter xwriter = XmlWriter.Create(Console.Out,settings); //XmlWriter 개체 생성
while (xreader.Read())
{
    if (xreader.NodeType == XmlNodeType.Element)
    {
        xwriter.WriteStartElement(xreader.Name);
        xwriter.WriteAttributes(xreader, false); //xreader의 현재 특성을 쓰기
        if (xreader.IsEmptyElement)
        {
            xwriter.WriteEndElement();
        }
    }
    else if (xreader.NodeType == XmlNodeType.EndElement)
    {
        xwriter.WriteEndElement();
    }
}
xwriter.Close();
xreader.Close();
Console.WriteLine();
}

```

[소스 2.3] XmlWriter 개체로 특성 쓰기 예제 코드

```
<?xml version="1.0" encoding="utf-8"?>
<!--XmlWriter 개체로 특성 쓰기-->
<books>
  <book title="XML.NET" 가격="12000" />
  <book title="ADO.NET" 가격="15000" />
</books>
```

[문서 2.3] [소스 2.3]을 실행했을 때 만들어지는 "data.xml" 파일 내용



```
<?xml version="1.0" encoding="ks_e_5601-1987"?>
<books>
  <book title="XML.NET" 가격="12000" />
  <book title="ADO.NET" 가격="15000" />
</books>
계속하려면 아무 키나 누르십시오 . . .
```

[그림 2.2] [소스 2.3] 실행 화면

### 2.2.3 형식화된 데이터 쓰기

XmlWriter 클래스에서는 WriteValue 메서드를 제공하고 있어 형식화된 값을 쉽게 작성할 수 있습니다. 앞에서 작성했던 [소스 2.2]와 [소스 2.3]에서 가격 정보를 쓸 때 WriteValue 메서드를 사용했으니 참고하시기 바랍니다.

이 외에도 XmlWriter 클래스에서 파생한 XmlTextWriter 클래스를 사용하면 W3C XML 1.0 및 XML 내의 네임스페이스 권장 사항에 따르는 XML 데이터를 스트림이나 파일에 작성할 수 있습니다.

## 3. XmlReader

XmlReader 클래스는 XML 표준 문서를 정방향으로 데이터를 판독하는 파서입니다. 정방향으로만 데이터를 판독하기 때문에 메모리 캐시를 사용하지 않으며 빠른 성능을 제공합니다.

▷ 클래스 상속 계층

System.Object

System.Xml.XmlReader

System.Xml.XmlDictionaryReader

System.Xml.XmlNodeReader

System.Xml.XmlTextReader

System.Xml.XmlValidatingReader

▷ 네임스페이스: System.Xml

▷ 어셈블리: System.Xml(System.Xml.dll)

XmlReader 클래스를 사용할 때는 먼저 XmlReader 개체를 생성하는 것에서 출발합니다. 그리고 유효성 및 규칙을 설정하고 필요한 요소 및 특성 및 내용을 읽는 작업을 수행할 수 있습니다.

이 책에서는 XmlReader 개체를 만드는 방법과 유효성을 검사하는 방법 및 XML 데이터의 요소, 특성, 내용을 읽는 방법과 형식화된 데이터를 읽는 방법에 관하여 소개할 것입니다.

### 3. 1 XmlReader 개체 생성

XmlReader 개체를 생성할 때는 XmlReader 클래스에서 제공하는 정적 메서드 Create를 이용합니다.

```
public static XmlReader Create(Stream input);
public static XmlReader Create(Stream input, XmlReaderSettings settings);
public static XmlReader Create(Stream input, XmlReaderSettings settings, string baseuri);
public static XmlReader Create(Stream input, XmlReaderSettings settings, XmlParserContext context);
public static XmlReader Create(string uri);
public static XmlReader Create(string uri, XmlReaderSettings settings);
public static XmlReader Create(string uri, XmlReaderSettings settings, XmlParserContext context);
public static XmlReader Create(TextReader input);
public static XmlReader Create(TextReader input, XmlReaderSettings settings);
public static XmlReader Create(TextReader input, XmlReaderSettings settings, string baseuri);
public static XmlReader Create(TextReader input, XmlReaderSettings settings, XmlParserContext context);
public static XmlReader Create(XmlReader reader, XmlReaderSettings settings);
```



### 3.1.1 XmlReaderSettings

XmlReader 클래스의 정적 메서드 Create를 이용하여 생성할 XmlReader 개체에 지원할 기능 집합을 설정하기 위해 XmlReaderSettings 개체를 이용합니다.

속성	초기값	설명
CheckCharacters	true	문자 검사 수행 여부 ( <a href="http://www.w3.org/TR/REC-xml#charsets">http://www.w3.org/TR/REC-xml#charsets</a> )
CloseInput	false	Close 메서드 호출할 때 XmlReader가 내부 스트림 혹은 TextReader도 함께 닫을지 여부
ConformanceLevel	ConformanceLevel.Document	규칙 수준 (ConformanceLevel 값 중 하나를 지정)
DtdProcessing	Prohibit	DtdProcessing 값 중 하나를 지정
IgnoreComments	false	주석을 무시할지 여부
IgnoreProcessingInstructions	false	처리 명령을 무시할지 여부
IgnoreWhitespace	false	유효하지 않은 공백을 무시할지 여부
LineNumberOffset	0	줄 번호 오프셋
LinePositionOffset	0	줄 위치 오프셋
MaxCharactersFromEntities	0(제한 없음을 의미)	엔터티 확장 후 최대 허용 문자 수
MaxCharactersInDocument	0(제한 없음을 의미)	최대 허용 문자 수
NameTable	null	문자열을 비교할 때 사용할 이름 테이블
ProhibitDtd	true	DTD 프로세스 금지 여부 (현재는 사용하지 않음)
Schemas	빈 XmlSchemaSet 개체	스키마 유효성 검사를 수용할 때 사용할 XmlSchemaSet 개체
ValidationFlags	AllowXmlAttributes	스키마 유효성 검사 설정 ValidationType 속성이 Schema로 설정된 개체에 적용 XmlSchemaValidationFlags 값의 집합
ValidataionType	ValidataionType.None	유효성 검사 또는 형식 할당 수행 여부 ValidataionType 값 중 하나
XmlResolver	자격 증명이 없는 새 XmlUrlResover 개체	외부 문서에 액세스할 때 사용할 XmlResolver 개체

[표 3.1] XmlReaderSettings 속성

### 3.1.2 XmlReader 개체 만들기

이번에는 간단한 예제를 통해 XmlReader 개체를 만드는 방법을 살펴봅시다. 먼저 12가지 Create 메서드 중에 4가지 방법을 사용하는 예를 보여드리고 난 후에 외부 데이터를 원본으로 XmlReader 개체를 생성하는 예제를 보여드릴게요. 그리고 마지막으로 XmlSchemaSet 개체를 이용하여 유효성 검사를 하는 예제를 보여드릴게요.

먼저 예제에 사용할 원본 XML 파일인 "data.xml"의 내용을 살펴봅시다.

```
<?xml version="1.0" encoding="utf-8"?>
<!--XmlReader 개체 만들기-->
<books>
  <book>
    <title>XML.NET 과 ADO.NET</title>
    <가격>22000</가격>
  </book>
</books>
```

[문서 3.1] data.xml 문서 내용

다음의 예제 코드는 다양한 형태로 XmlReader 개체를 생성하여 입력 스트림의 내용을 XmlWriter 개체를 이용하여 콘솔 화면에 출력하는 소스입니다.

여기에서는 네 가지 방법으로 XmlReader 개체를 생성하는 것을 보여주고 있습니다. 하나는 입력 스트림을 전달하여 XmlReader 개체를 생성하는 것이고 두 번째는 입력 스트림과 XmlReaderSettings 개체를 이용, 세 번째는 소스 파일의 uri를 문자열로 전달하여 생성, 네 번째는 소스 파일의 uri와 XmlReaderSettings 개체를 이용하는 것입니다.

참고로 예제에 사용한 XmlReaderSettings 개체는 주석을 무시하도록 설정하였습니다.

```
static void Main(string[] args)
{
    //Create(Stream input);
    Console.WriteLine("-----Start Test1-----");
    FileStream fs = new FileStream("data.xml",
        FileMode.OpenOrCreate, FileAccess.Read, FileShare.Read);
    XmlReader reader1 = XmlReader.Create(fs);
    WirteConsole(reader1);
    reader1.Close();
    fs.Close();
    Console.WriteLine("----- End Test1 -----");
}
```

```

//Create(Stream input, XmlReaderSettings settings);
Console.WriteLine("-----Start Test2-----");
fs = new FileStream("data.xml",
    FileMode.OpenOrCreate, FileAccess.Read, FileShare.Read);
XmlReaderSettings settings = new XmlReaderSettings();
settings.IgnoreComments = true;
XmlReader reader2 = XmlReader.Create(fs, settings);
WirteConsole(reader2);
reader2.Close();
fs.Close();
Console.WriteLine("----- End Test2 -----");
//Create(string uri);
Console.WriteLine("-----Start Test3-----");
settings.IgnoreComments = true;
XmlReader reader3 = XmlReader.Create("data.xml");
WirteConsole(reader3);
reader3.Close();
Console.WriteLine("----- End Test3 -----");
//Create(string uri, XmlReaderSettings settings);
Console.WriteLine("-----Start Test4-----");
settings.IgnoreComments = true;
XmlReader reader4 = XmlReader.Create("data.xml", settings);
WirteConsole(reader4);
reader4.Close();
Console.WriteLine("----- End Test4 -----");
}

private static void WirteConsole(XmlReader reader)
{
    XmlWriter xwriter = XmlWriter.Create(Console.Out);
    xwriter.WriteNode(reader, false);
    xwriter.Close();
    Console.WriteLine();
}

```

[소스 3.1] XmlReader 개체 만들기 예제 코드

```
C:\Windows\system32\cmd.exe

-----Start Test1-----
<?xml version="1.0" encoding="utf-8"?>
<!--XmlReader 개체 만들기-->
<books>
  <book>
    <title>XML.NET 과 ADO.NET</title>
    <가격>22000</가격>
  </book>
</books>
----- End Test1 -----
-----Start Test2-----
<?xml version="1.0" encoding="utf-8"?>

<books>
  <book>
    <title>XML.NET 과 ADO.NET</title>
    <가격>22000</가격>
  </book>
</books>
----- End Test2 -----
-----Start Test3-----
<?xml version="1.0" encoding="utf-8"?>
<!--XmlReader 개체 만들기-->
<books>
  <book>
    <title>XML.NET 과 ADO.NET</title>
    <가격>22000</가격>
  </book>
</books>
----- End Test3 -----
-----Start Test4-----
<?xml version="1.0" encoding="utf-8"?>

<books>
  <book>
    <title>XML.NET 과 ADO.NET</title>
    <가격>22000</가격>
  </book>
</books>
----- End Test4 -----
계속하려면 아무 키나 누르십시오 . . .
```

[그림 3.1] [소스 3.1] 실행 화면

이번에는 외부 자원을 소스로 하는 XmlReader 개체를 만드는 예제를 살펴봅시다. 여기에서는 제가 운영하는 커뮤니티 사이트에 있는 게시판 rss("<http://cafe.daum.net/xml/rss/ehclub.net/Svc5>")를 소스로 할게요.

외부 자원을 소스로 할 때는 XmlUrlResolver 개체를 생성하여 XmlReaderSettings 개체의 XmlResolver 속성에 설정하고 XmlReader 개체를 생성할 때 XmlReaderSettings 개체를 전달하면 됩니다.

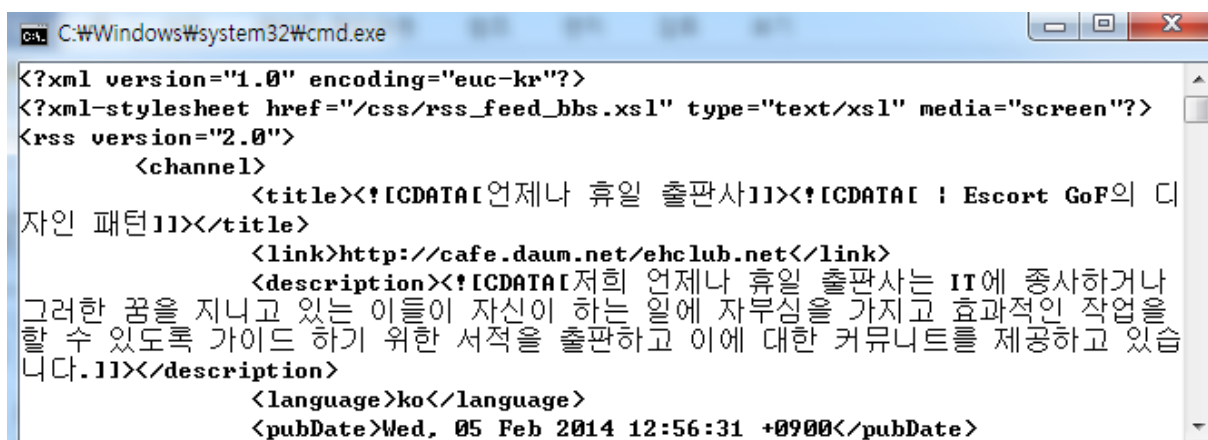
```
static void Main(string[] args)
{
    XmlUrlResolver resolver = new XmlUrlResolver();
    resolver.Credentials = System.Net.CredentialCache.DefaultCredentials;

    XmlReaderSettings settings = new XmlReaderSettings();
    settings.XmlResolver = resolver;

    XmlReader reader = XmlReader.Create("http://cafe.daum.net/xml/rss/ehclub.net/Svc5", settings);
    WirteConsole(reader);
    reader.Close();
}

private static void WirteConsole(XmlReader reader)
{
    XmlWriter xwriter = XmlWriter.Create(Console.Out);
    xwriter.WriteNode(reader, false);
    xwriter.Close();
    Console.WriteLine();
}
```

[소스 3.2] 외부 자원을 소스로 하는 XmlReader 개체 만들기 예제 코드



[그림 3.2] [소스 3.2] 실행 화면

이번에는 XmlSchemaSet 개체로 유효성 검사를 할 수 있게 XmlReader 개체를 생성하는 방법을 살펴볼게요.

예제에서는 XmlSchemaSet 개체를 생성하여 이미 작성한 스키마 파일 "data.xsd"의 내용을 XmlSchemaSet에 추가하고 XmlReaderSettings 개체의 Schemas 속성에 설정합니다. 그리고 XmlReader 개체를 이용하여 데이터 소스에 유효성이 위배한 부분을 발견할 때 처리하기 위한 이벤트 핸들러를 추가합니다.

이와 같은 작업을 한 후에 XmlReaderSettings 개체를 전달하여 XmlReader 개체를 생성하면 이후에 읽기 작업 등에서 데이터 소스에 유효성이 위배한 부분을 발견하면 설정한 이벤트 핸들러가 동작합니다.

```
static void Main(string[] args)
{
    XmlSchemaSet sc = new XmlSchemaSet();
    sc.Add("urn:books-schema", "data.xsd");

    XmlReaderSettings settings = new XmlReaderSettings();
    settings.ValidationType = ValidationType.Schema;
    settings.Schemas = sc;
    settings.ValidationEventHandler += new ValidationEventHandler (ValidationCallBack);

    XmlReader reader = XmlReader.Create("data.xml", settings);
    WirteConsole(reader);
    reader.Close();
}

private static void ValidationCallBack(object sender, ValidationEventArgs e)
{
    Console.WriteLine("유효성 위배: {0}", e.Message);
}

private static void WirteConsole(XmlReader reader)
{
    XmlWriter xwriter = XmlWriter.Create(Console.Out);
    xwriter.WriteNode(reader, false);
    xwriter.Close();
    Console.WriteLine();
}
```

[소스 3.3] 유효성 검사를 설정한 XmlReader 개체 만들기 예제 코드

유효성 검사는 XmlSchemaSet 개체를 이용하는 방법 외에도 DTD를 사용하거나 래핑된 XmlReader 개체를 사용하는 방법 등이 있으니 MSDN을 참고하시기 바랍니다.

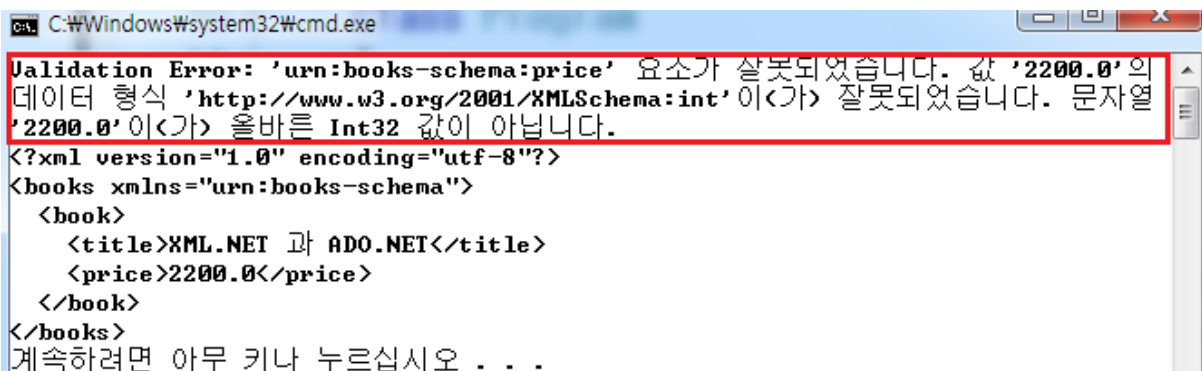
다음은 미리 작성한 스키마 파일 "data.xsd"와 데이터 소스로 사용할 "data.xml" 파일의 내용입니다.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="urn:books-schema"
    elementFormDefault="qualified" targetNamespace="urn:books-schema">
  <xsd:element name="books" type="booksType"/>
  <xsd:complexType name="booksType">
    <xsd:sequence maxOccurs="unbounded">
      <xsd:element name="book" type="bookType"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="bookType">
    <xsd:sequence>
      <xsd:element name="title" type="xsd:string"/>
      <xsd:element name="price" type="xsd:int"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

[문서 3.2] data.xsd 문서 내용

```
<?xml version="1.0" encoding="utf-8"?>
<books xmlns="urn:books-schema">
  <book>
    <title>XML.NET 과 ADO.NET</title>
    <price>2200.0</price>
  </book>
</books>
```

[문서 3.3] data.xml 문서 내용



[그림 3.3] [소스 3.3] 실행 화면

## 3. 2 XmlReader 개체로 데이터 분석

XmlReader 개체는 정방향으로 XML 데이터 소스를 분석하기 위함입니다.

구조화되어 있는 XML 데이터 소스를 정방향으로 읽기 작업을 하여 원하는 프로그램 데이터로 가공하기 위해서는 현재 위치의 노드가 어떠한 형식인지 알 수 있어야 합니다. 그리고 현재 노드 형식에 따라 요소 읽기와 특성 읽기, 값 읽기 등을 할 수 있어야 할 것입니다.

### 3.2.1 현재 위치의 노드 형식 알아내기

XmlReader 개체는 정방향으로만 읽기 작업할 수 있습니다. 따라서 XmlReader 개체로 Read 메서드를 호출하면 다음 노드로 위치가 변합니다. XML 데이터는 구조적인 데이터이므로 현재 분석한 노드가 무엇인지에 따라 처리를 다르게 해야 할 때가 많습니다.

XmlReader 클래스에서는 개체의 멤버 속성으로 NodeType을 제공하고 있어 현재 위치의 노드가 어떠한 형식인지 쉽게 확인할 수 있습니다.

```
static void Main(string[] args)
{
    XmlReader reader = XmlReader.Create("data.xml");
    reader.MoveToContent();

    while (reader.Read())
    {
        WriteNodeInfo(reader);
    }
}

private static void WriteNodeInfo(XmlReader reader)
{
    Console.WriteLine("노드 형:{0}", reader.NodeType);
    Console.WriteLine(" > 노드 이름:{0}", reader.Name);
    Console.WriteLine(" > 노드 데이터:{0}", reader.Value);
}
```

[소스 3.4] 현재 위치의 노드 형식 알아내기 예제 코드



### 3.2.2 요소 읽기

XmlReader 클래스에서는 요소 읽기 작업에 관한 6가지 메서드와 1가지 속성을 제공하고 있습니다. 먼저 이들의 역할과 원형을 간략히 살펴봅시다.

현재 노드가 시작 요소인지 확인할 때는 IsStartElement 메서드를 사용합니다.

```
public bool IsStartElement ();  
public bool IsStartElement (string name);  
public bool IsStartElement (string local_name, string ns);
```

현재 노드가 시작 요소일 때 요소를 읽기 위해 ReadStartElement 메서드를 제공합니다.

```
public void ReadStartElement ();  
public void ReadStartElement (string name);  
public void ReadStartElement (string local_name, string ns);
```

현재 노드가 종료 요소일 때 읽기 위해 ReadEndElement 메서드를 제공합니다.

```
public void ReadEndElement ();
```

현재 노드가 시작 요소이고 값이 문자열일 때 값을 읽기 위해 ReadElementString 메서드를 제공합니다.

```
public string ReadElementString ( );  
public string ReadElementString (string name );  
public string ReadElementString (string local_name, string ns );
```

원하는 하위 요소로 이동할 때는 ReadToDescendant 메서드를 사용합니다.

```
public bool ReadToDescendant (string name);  
public bool ReadToDescendant (string local_name, string ns_uri);
```

원하는 형제 요소로 이동할 때는 ReadToNextSibling 메서드를 사용합니다.

```
public bool ReadToNextSibling(string name);  
public bool ReadToNextSibling(string name, string ns_uri);
```

마지막으로 비어있는 요소인지 확인할 때 IsEmptyElement 속성을 사용합니다.

```
public bool IsEmptyElement{ get; }
```

다음은 XmlReader 개체를 이용하여 요소를 읽는 예제입니다. 예제 코드에서는 도서 목록이 작성되어 있는 원본 XML 파일에서 도서 정보를 읽어와서 도서 개체를 만들고 있습니다.

먼저 XmlReader 개체의 Read 메서드를 반복 호출하여 "book" 요소가 시작하는 태그인지 확인하여 맞다면 Book 클래스의 정적 메서드인 MakeBook 메서드를 통해 Book 개체를 생성합니다. 이 때 "book" 요소가 시작하는 태그인지 확인하기 위해 IsStartElement 메서드를 호출하고 있습니다.

MakeBook 메서드에서는 하위 요소 중에 "title" 위치로 이동할 때 ReadToDescendant 메서드를 호출하고 있으며 "title" 요소는 값으로 문자열을 갖고 있어서 ReadElementString 메서드 호출을 통해 도서 제목을 얻어옵니다.

그리고 "title"의 형제 요소인 "price" 요소로 이동하기 위해 ReadToNextSibling 메서드를 호출하고 있으며 시작 요소를 읽기 위해 ReadStartElement 메서드를 호출합니다. 그리고 값을 얻어오기 위해 XmlReader 개체의 속성 Value를 사용합니다.

```
class Book
{
    internal string Title{    get;    private set;    }
    internal int Price {    get;    private set;    }
    internal static Book MakeBook(XmlReader xr)
    {
        string title = string.Empty;
        int price = 0;
        xr.ReadToDescendant("title");
        title = xr.ReadElementString("title");
        xr.ReadToNextSibling("price");
        xr.ReadStartElement("price");
        price = int.Parse(xr.Value);
        return new Book(title, price);
    }
    Book(string title, int price)
    {
        Title = title;
        Price = price;
    }
    public override string ToString()
    {
        return Title;
    }
}
```

```

class Program
{
    static void Main(string[] args)
    {
        ArrayList ar = new ArrayList();
        XmlReader reader = XmlReader.Create("data.xml");
        while (reader.Read())
        {
            if (reader.IsStartElement("book"))
            {
                Book book = Book.MakeBook(reader);
                if (book != null) { ar.Add(book); }
            }
        }
        Console.WriteLine("도서 개수:{0}", ar.Count);
        foreach (Book book in ar)
        {
            Console.WriteLine("도서명:{0} 가격:{1}", book.Title, book.Price);
        }
    }
}

```

[소스 3.5] XmlReader 개체로 요소 읽기 예제 코드

```

<?xml version="1.0" encoding="utf-8"?>
<books xmlns="urn:books-schema">
  <book>
    <title>XML.NET</title>
    <price>12000</price>
  </book>
  <book>
    <title>ADO.NET</title>
    <price>15000</price>
  </book>
</books>

```

[문서 3.4] data.xml 문서 내용

### 3.2.3 특성 읽기

XmlReader 클래스는 요소의 특성을 읽기 위해 6가지 메서드와 3가지 속성과 인덱서를 제공합니다. 먼저 이들의 역할과 원형을 살펴봅시다.

특성 값을 얻어올 때 GetAttribute 메서드를 이용합니다.

```
public string GetAttribute (int i); //i는 인덱스
public string GetAttribute (string name);
public string GetAttribute (string name, string ns_uri);
```

XmlReader 개체의 현재 위치를 지정한 특성으로 이동할 때 MoveToAttribute 메서드를 이용합니다.

```
public void MoveToAttribute (int i); //i는 인덱스
public void MoveToAttribute (string name);
public void MoveToAttribute (string name, string ns);
```

현재 특성 노드를 포함하는 요소로 이동할 때 MoveToElement 메서드를 이용합니다.

```
public bool MoveToElement ( );
```

첫 번째 특성으로 이동할 때 MoveToFirstAttribute 메서드를 이용합니다.

```
public bool MoveToFirstAttribute ( );
```

다음 특성으로 이동할 때 MoveToNextAttribute 메서드를 이용합니다.

```
public bool MoveToNextAttribute ( );
```

특성 값을 분석할 때는 ReadAttributeValue 메서드를 이용합니다.

```
public bool ReadAttributeValue ( );
```

현재 노드의 특성 개수를 얻어올 때는 AttributeCount 속성을 이용합니다.

```
public int AttributeCount { get; }
```

현재 노드에 특성이 있는지 확인할 때는 HasAttributes 속성을 이용합니다.

```
public bool HasAttributes { get; }
```

현재 노드가 유효성 검사에 사용할 DTD나 스키마에 정의한 기본값에서 생성한 값을 가진 특성인지 확인할 때는 IsDefault 속성을 사용합니다.

```
public bool IsDefault { get; }
```

특성의 값을 얻어올 때 인덱서를 사용할 수 있습니다.

```
public string this[int i] { get; } //i:인덱스
public string this[string name] { get; }
public string this[string name, string ns_uri] { get; }
```

다음처럼 XML 문서 파일 "data.xml"이 있을 때 간단하게 XmlReader 개체로 특성 읽기에 관한 예제 코드를 살펴봅시다.

```
<?xml version="1.0" encoding="utf-8"?>
<books category=".NET">
  <book title="XML.NET" price="12000" count="50"/>
  <book title="ADO.NET" price="15000" count="60"/>
</books>
```

[문서 3.5] data.xml 문서 내용

예제에서는 먼저 "data.xml" 문서 파일을 인자로 XmlReader 개체를 생성합니다. 그리고 XmlReader 개체로 읽기 작업을 반복하고 현재 위치의 노드에 대한 처리를 하는 구조로 작성하였습니다.

현재 위치에 있는 노드가 "books" 시작 요소일 때는 특성이 있는지 HasAttributes로 확인한 후에 category 특성 정보를 읽어와서 콘솔 화면에 출력합니다.

```
if (reader.HasAttributes)
{
    string category = reader.GetAttribute("category");
    Console.WriteLine("카테고리:{0}", category);
}
```

현재 위치에 있는 노드가 "book" 시작 요소일 때는 AttributeCount 속성을 이용하여 특성 개수를 얻어온 후에 for문에서 각 인덱스에 있는 특성을 콘솔 화면에 출력합니다. XmlReader 개체가 정방향으로만 이동 가능하다고 했는데 XmlReader 개체의 속성을 사용한다고 해서 위치가 변하지 않습니다.

```
int attr_cnt=reader.AttributeCount;
for (int i = 0; i < attr_cnt; i++)
{
    Console.Write("{0} ", reader[i]);
}
```

그리고 또 다른 방법으로 GetAttribute 메서드에 원하는 특성 항목을 얻어와서 콘솔화면에 출력합니다. 마찬가지로 GetAttribute 메서드를 호출하여도 XmlReader 개체의 위치는 변하지 않습니다.

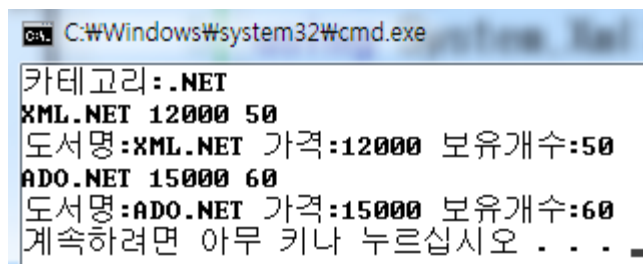
```
string title = reader.GetAttribute("title");
string price = reader.GetAttribute("price");
int count = int.Parse(reader.GetAttribute("count"));
Console.WriteLine("도서명:{0} 가격:{1} 보유개수:{2}", title, price, count);
```

```

static void Main(string[] args)
{
    XmlReader reader = XmlReader.Create("data.xml");
    while (reader.Read())
    {
        if (reader.IsStartElement("books"))
        {
            if (reader.HasAttributes)
            {
                string category = reader.GetAttribute("category");
                Console.WriteLine("카테고리:{0}", category);
            }
        }
        if (reader.IsStartElement("book"))
        {
            int attr_cnt=reader.AttributeCount;
            for (int i = 0; i < attr_cnt; i++)
            {
                Console.Write("{0} ", reader[i]);
            }
            Console.WriteLine();
            string title = reader.GetAttribute("title");
            string price = reader.GetAttribute("price");
            int count = int.Parse(reader.GetAttribute("count"));
            Console.WriteLine("도서명:{0} 가격:{1} 보유개수:{2}", title, price, count);
        }
    }
}

```

[소스 3.6] XmlReader 개체로 특성 읽기 예제 코드



```

C:\Windows\system32\cmd.exe
카테고리: .NET
XML.NET 12000 50
도서명:XML.NET 가격:12000 보유개수:50
ADO.NET 15000 60
도서명:ADO.NET 가격:15000 보유개수:60
계속하려면 아무 키나 누르십시오 . . .

```

[그림 3.4] [소스 3.6] 실행 화면

### 3.2.4 기타

XmlReader 클래스에서는 요소 읽기와 특성 읽기 외에도 XML 데이터 소스에 있는 다양한 내용을 읽기 위한 메서드와 속성을 제공하고 있습니다. 여기에서는 이러한 부가적인 기능과 속성을 개괄적으로 소개할게요.

현재 노드의 내용을 문자열로 반환하는 ReadString 메서드를 제공하고 있습니다. ReadString 메서드를 이용하면 새로운 태그가 나타나기 전의 내용을 문자열 형태로 반환합니다.

```
public string ReadString ( );
```

```
if (reader.IsStartElement("book"))
{
    Console.WriteLine(reader.ReadString());
}
```

태그 내부에 있는 모든 내용을 문자열로 읽을 때는 ReadInnerXml 메서드를 이용합니다.

```
public string ReadInnerXml ( );
```

그리고 태그를 포함하여 현재 위치의 노드와 모든 자식 노드의 정보를 얻고자 할 때는 ReadOuterXml 메서드를 사용합니다.

```
public string ReadOuterXml ( );
```

다음의 XML 데이터 파일을 소스로 XmlReader 개체를 만들어서 ReadInnerXml 메서드와 ReadOuterXml 메서드의 결과가 어떻게 다른지 간단히 살펴봅시다.

```
<?xml version="1.0" encoding="utf-8"?>
<books>
  <book price="12000" count="50">
    <title>ADO.NET </title>
  </book>
  <book price="15000" count="60">
    <title>XML.NET </title>
  </book>
</books>
```

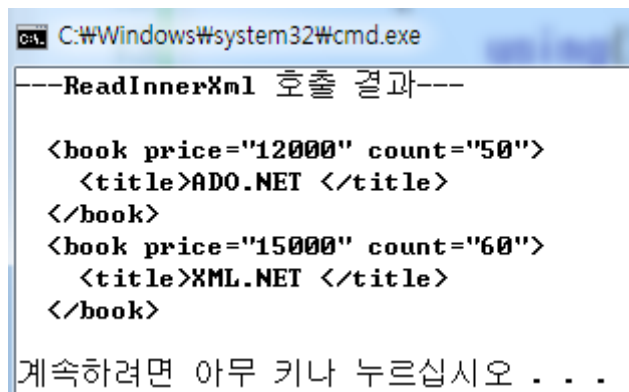
[문서 3.6] data.xml 문서 내용

```

static void Main(string[] args)
{
    using(XmlReader reader = XmlReader.Create("data.xml"))
    {
        while(reader.Read())
        {
            if (reader.NodeType == XmlNodeType.Element)
            {
                Console.WriteLine("---ReadInnerXml 호출 결과---");
                Console.WriteLine(reader.ReadInnerXml());
            }
        }
    }
}

```

[소스 3.7] ReadInnerXml 메서드 호출 예제 코드



```

C:\Windows\system32\cmd.exe
---ReadInnerXml 호출 결과---

<book price="12000" count="50">
  <title>ADO.NET </title>
</book>
<book price="15000" count="60">
  <title>XML.NET </title>
</book>
계속하려면 아무 키나 누르십시오 . . .

```

[그림 3.5] [소스 3.7] 실행 화면

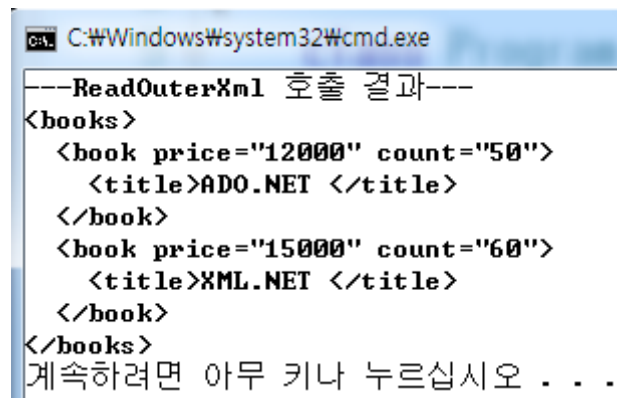


```

static void Main(string[] args)
{
    using(XmlReader reader = XmlReader.Create("data.xml"))
    {
        while(reader.Read())
        {
            if (reader.NodeType == XmlNodeType.Element)
            {
                Console.WriteLine("---ReadOuterXml 호출 결과---");
                Console.WriteLine(reader.ReadOuterXml());
            }
        }
    }
}

```

[소스 3.8] ReadOuterXml 메서드 호출 예제 코드



```

C:\Windows\system32\cmd.exe
---ReadOuterXml 호출 결과---
<books>
  <book price="12000" count="50">
    <title>ADO.NET </title>
  </book>
  <book price="15000" count="60">
    <title>XML.NET </title>
  </book>
</books>
계속하려면 아무 키나 누르십시오 . . .

```

[그림 3.6] [소스 3.8] 실행 화면

이 외에 처리 명령이나 문서 형식, 주석, 공백 같은 형식의 노드를 건너뛰고 실제 내용이 있는 노드로 이동하는 MoveToContent 메서드를 제공하고 있습니다. MoveToContent 메서드에서는 현재 위치의 노드 형식을 반환합니다.

```
public XmlNodeType MoveToContent ( );
```

그리고 ReadSubtree 메서드를 이용하면 현재 노드 위치부터 자식 노드들을 읽을 때 사용할 XmlReader 개체를 반환받아 사용할 수 있습니다.

```
public virtual XmlReader ReadSubtree ( );
```

## 4. XmlSchema

XML 스키마는 표준 XML 문서 구조의 정의입니다. 이를 이용하면 유효성 검사를 할 수 있고 판독하는 등의 강력한 기능을 사용할 수 있습니다.

.NET Framework에서는 스키마를 만들거나 유효성을 검사할 수 있는 스키마 개체 모델(SOM) API를 제공하고 있습니다.

### 4. 1 XML 스키마 작성 및 판독

XML 스키마 파일을 작성할 때는 XmlSchema 클래스를 비롯하여 다양한 클래스를 사용합니다.

프로그램 메모리에 XML 스키마를 만들 때는 XmlSchema 개체를 생성합니다. 그리고 XML 스키마 개체에 스키마 요소를 표현하기 위해 XmlSchemaElement 개체를 만들어 원하는 내용으로 설정하고 XmlSchema 개체의 멤버 Items 컬렉션 속성에 추가합니다.

```
XmlSchema schema = new XmlSchema();  
// <xs:element name="book" type="string"/>  
XmlSchemaElement elem_book = new XmlSchemaElement();  
schema.Items.Add(elem_book);  
elem_book.Name = "book";  
elem_book.SchemaTypeName = new XmlQualifiedName("string", "http://www.w3.org/2001/XMLSchema");
```

그리고 이렇게 만든 XmlSchema 개체들을 XmlSchemaSet에 추가하여 컴파일 과정을 거쳐 유효성 테스트를 할 수 있습니다.

```
XmlSchemaSet schemaSet = new XmlSchemaSet();  
schemaSet.ValidationEventHandler += new ValidationEventHandler(ValidationCallbackOne);  
schemaSet.Add(schema);  
schemaSet.Compile();
```

다음은 유효성 검사에서 위배했을 때 처리하는 이벤트 핸들러의 예입니다.

```
public static void ValidationCallbackOne(object sender, ValidationEventArgs args)  
{  
    Console.WriteLine(args.Message);  
}
```

그리고 XmlSchema 개체는 Write 메서드를 제공하고 있어 XML 스키마 파일을 만들 수 있습니다.

```
FileStream fs = new FileStream("data.xsd", FileMode.Create);  
compiledSchema.Write(fs);  
fs.Close();
```

다음의 예제 코드는 책과 앨범을 포함하는 콘텐츠를 스키마로 표현한 예제 코드입니다.

예제 코드에서는 XmlSchema 개체를 생성한 후에 책 요소와 앨범 요소에 관한 XmlSchemaElement 개체를 생성하여 XmlSchema 개체의 Items 컬렉션에 추가하는 것으로 출발합니다. 그리고 책 요소와 앨범 요소를 하위 요소로 포함할 수 있는 콘텐츠 집합 요소에 관한 XmlSchemaElement 개체를 생성하여 XmlSchema 개체의 Items 컬렉션에 추가합니다.

콘텐츠 집합 요소에 관한 XmlSchemaElement 개체는 복합 타입임을 지정하고 자식 요소로 책 요소와 앨범 요소를 포함할 수 있음을 지정합니다. 이 후에 XmlSchemaSet 개체를 생성하여 XmlSchema 개체를 추가한 후에 컴파일을 통해 유효성 검사를 수행합니다. 마지막으로 유효성 검사에 통과한 XmlSchema 개체를 파일로 저장하고 있습니다.

```
static void Main(string[] args)
{
    XmlSchema schema = new XmlSchema();

    // <xs:element name="book" type="string"/>
    XmlSchemaElement elem_book = new XmlSchemaElement();
    schema.Items.Add(elem_book);
    elem_book.Name = "book";
    elem_book.SchemaTypeName =
        new XmlQualifiedName("string", "http://www.w3.org/2001/XMLSchema");

    // <xs:element name="albumn" type="string"/>
    XmlSchemaElement elem_albumn = new XmlSchemaElement();
    schema.Items.Add(elem_albumn);
    elem_albumn.Name = "albumn";
    elem_albumn.SchemaTypeName =
        new XmlQualifiedName("string", "http://www.w3.org/2001/XMLSchema");

    // <xs:element name="contents">
    XmlSchemaElement elem_contents = new XmlSchemaElement();
    schema.Items.Add(elem_contents);
    elem_contents.Name = "contents";

    // <xs:complexType>
    XmlSchemaComplexType complexType = new XmlSchemaComplexType();
    elem_contents.SchemaType = complexType;
```

```

// <xs:choice minOccurs="0" maxOccurs="unbounded">
XmlSchemaChoice choice = new XmlSchemaChoice();
complexType.Particle = choice;
choice.MinOccurs = 0;
choice.MaxOccursString = "unbounded";

// <xs:element ref="book"/>
XmlSchemaElement ref_book = new XmlSchemaElement();
choice.Items.Add(ref_book);
ref_book.RefName = new XmlQualifiedName("book");
// <xs:element ref="albumn"/>
XmlSchemaElement ref_albumn = new XmlSchemaElement();
choice.Items.Add(ref_albumn);
ref_albumn.RefName = new XmlQualifiedName("albumn");

XmlSchemaSet schemaSet = new XmlSchemaSet();
schemaSet.ValidationEventHandler += new ValidationEventHandler(ValidationCallbackOne);
schemaSet.Add(schema);
schemaSet.Compile();

XmlSchema compiledSchema = null;
foreach (XmlSchema schema1 in schemaSet.Schemas())
{
    compiledSchema = schema1;
}

XmlNamespaceManager nsmgr = new XmlNamespaceManager(new NameTable());
nsmgr.AddNamespace("xs", "http://www.w3.org/2001/XMLSchema");
FileStream fs = new FileStream("data.xsd", FileMode.Create);
compiledSchema.Write(fs);
fs.Close();
}

public static void ValidationCallbackOne(object sender, ValidationEventArgs args)
{
    Console.WriteLine(args.Message);
}

```

[소스 4.1] XML 스키마 파일 작성 예제 코드

다음은 [소스 4.1] 예제 코드로 만들어진 "data.xsd" 파일의 내용입니다.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="book" type="xs:string" />
  <xs:element name="albumn" type="xs:string" />
  <xs:element name="contents">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="book" />
        <xs:element ref="albumn" />
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

[문서 4.1] data.xsd 문서 내용

이렇게 작성한 XML 스키마 파일은 XML 데이터 파일을 로딩할 때 유효성 검사에 사용할 수 있습니다. 이미 3장에서 XML 스키마 파일을 이용하여 유효성 검사를 하는 예제를 보여준 적이 있는데 다시 한 번 확인해 봅시다. 여기에서는 스키마 파일인 "data.xsd"에 정의한 스키마에 맞게 작성한 "data.xml" 파일과 스키마에 위배한 요소가 있는 "data2.xml" 파일의 내용을 XmlReader 개체로 읽어오는 예제를 보여드릴게요.

```
<?xml version="1.0" encoding="utf-8"?>
<contents xmlns="xs">
  <book>XML.NET</book>
  <albumn>My Way</albumn>
  <book>ADO.NET</book>
</contents>
```

[문서 4.2] data.xml 문서 내용

```
<?xml version="1.0" encoding="utf-8"?>
<contents xmlns="xs">
  <book>XML.NET</book>
  <albumn>My Way</albumn>
  <books>ADO.NET</books>
</contents>
```

[문서 4.3] data2.xml 문서 내용

```

static void Main(string[] args)
{
    XmlSchemaSet sc = new XmlSchemaSet();
    sc.Add("xs", "data.xsd");
    XmlReaderSettings settings = new XmlReaderSettings();
    settings.ValidationType = ValidationType.Schema;
    settings.Schemas = sc;
    settings.ValidationEventHandler += new ValidationEventHandler(ValidationCallBack);
    TestLoading("data.xml", settings);
    TestLoading("data2.xml", settings);
}

private static void TestLoading(string path, XmlReaderSettings settings)
{
    XmlReader reader = XmlReader.Create(path, settings);
    XmlWriter xwriter = XmlWriter.Create(Console.Out);
    xwriter.WriteNode(reader, false);
    xwriter.Close();
    Console.WriteLine();
    reader.Close();
}

private static void ValidationCallBack(object sender, ValidationEventArgs e)
{
    Console.WriteLine("유효성 오류: {0}", e.Message);
}

```

[소스 4.2] XML 스키마 파일에 의거해서 XML 읽기 예제 코드

```

C:\Windows\system32\cmd.exe
<?xml version="1.0" encoding="utf-8"?>
<contents xmlns="xs">
  <book>XML.NET</book>
  <albumn>My Way</albumn>
  <book>ADO.NET</book>
</contents>
Validation Error: 요소 네임스페이스 'xs'의 'contents'에 잘못된 사식 요소 네임스
페이스 'xs'의 'books'이(가) 있습니다. 필요한 요소 목록: 네임스페이스 'xs'의 'boo
k, albumn'
<?xml version="1.0" encoding="utf-8"?>
<contents xmlns="xs">
  <book>XML.NET</book>
  <albumn>My Way</albumn>
  <books>ADO.NET</books>
</contents>
계속하려면 아무 키나 누르십시오 . . .

```

[그림 4.1] [소스 4.2] 실행 화면

## 5. DOM 모델

.NET Framework에서는 문서 기반으로 XML을 처리할 수 있는 DOM 모델을 제공하고 있습니다. DOM 모델은 XML 데이터를 프로그램 상의 메모리에 문서 형태로 표현한 것입니다. XmlReader 개체로 XML 데이터를 읽을 때는 메모리에 캐시하지 않고 정방향으로만 이동하면서 읽습니다. 하지만 DOM 모델에서는 메모리에 데이터를 캐시하기 때문에 효과적으로 읽고 조작하고 수정할 수 있습니다.

### 5. 1 DOM 모델의 형식들

DOM모델에서는 XML 문서를 노드 트리 형태로 메모리에 캐시합니다. 이를 위해 다양한 노드 형식을 제공하고 있는데 이들의 형식은 W3C 표준에 맞게 제공하고 있습니다.

다음은 DOM 모델로 XML 문서를 메모리에 캐시할 때 사용하는 대표적인 노드 형식들입니다.

XML 문서를 트리 형태로 메모리에 캐시하는 프로그램 메모리 상의 XML 문서를 위해 XmlDocument 클래스를 제공하고 있습니다.

XML 요소를 표현하기 위해 XmlElement 클래스와 특성을 표현하기 위해 XmlAttribute 클래스를 제공합니다.

요소나 특성에 속한 텍스트를 표현할 때는 XmlText 클래스를 사용하고 주석을 표현할 때는 XmlComment 클래스를 사용합니다.

CDATA를 나타낼 때 XmlCDataSection 클래스, DTD 하위 집합 또는 외부 DTD 및 매개 변수 엔티티를 표현할 때 XmlEntity 클래스를 사용합니다. 그리고 DTD의 노테이션을 나타낼 때 XmlNotation을 제공하며 문서 형식을 나타낼 때 XmlDocumentType 클래스를 사용합니다.

그리고 트리 구조없이 하나 이상의 노드를 포함하는 임시 노드를 표현하기 위해 XmlDocumentFragment 클래스를 제공합니다.

이 외에도 W3C에는 정의되어 있지 않지만 .NET Framework에서는 추가적으로 노드 선언할 때 사용할 수 있는 XmlDeclaration, 혼합 내용에 들어가는 유효 공백을 나타내는 XmlSignificantWhitespace 및 요소 내용의 공백을 나타내는 XmlWhitespace 클래스 등을 제공하고 있습니다.

## 5. 2 DOM 모델로 XML 문서 만들기

DOM 모델에서는 프로그램 상의 메모리에 문서를 XmlDocument 클래스로 표현하고 있습니다. 프로그램 상의 메모리에 있는 XmlDocument 개체는 물리적인 저장소에 있는 XML 문서를 로딩하여 메모리에 XML 데이터를 캐시할 수 있고 데이터 내용을 변경 및 작성도 할 수 있습니다.

여기에서는 XmlDocument 개체를 이용하여 데이터 내용을 추가한 후에 물리적인 저장소에 파일로 XML 문서를 만드는 방법을 간략하게 알아보겠습니다. 먼저 XmlDocument 개체를 생성합니다. 그리고 XML 데이터를 추가합니다. 마지막으로 XmlDocument 개체의 데이터를 파일로 저장합니다.

```
static void Main(string[] args)
{
    XmlDocument doc = new XmlDocument();
    doc.LoadXml("<?xml version=W\"1.0W\" encoding=W\"utf-8W\"?>"+
        "<books>"+
        "<book>"+
        "<title> ADO.NET </title>"+
        "</book>"+
        "</books>");
    doc.Save("data.xml");
}
```

[소스 5.1] DOM 개체로 XML 문서 만들기 예제 코드

```
<?xml version="1.0" encoding="utf-8"?>
<books>
  <book>
    <title> ADO.NET </title>
  </book>
</books>
```

[문서 5.1] data.xml 문서 내용



## 5. 3 노드 추가 및 제거, 선택

XmlDocuemnt 클래스에는 필요한 형식의 노드를 생성할 수 있는 Create 메서드를 제공하고 있습니다.

노드의 선언을 만들 때는 CreateXmlDeclaration 메서드를 사용합니다.

```
public XmlDeclaration CreateXmlDeclaration ( string version, string encoding, string standalone);
```

버전은 항상 "1.0"이어야 하고 인코딩을 Encoding 클래스에서 지원하는 문자열로 설정합니다. 만약 인코딩을 null로 지정하면 기본 인코딩을 사용합니다. 그리고 standalone의 값은 "yes" 혹은 "no"을 사용할 수 있고 null이나 string.Empty를 사용하면 선언에 기록하지 않습니다.

XmlDeclartaion 개체는 XmlDocument 개체의 DocumentElement 속성 앞에 추가합니다.

```
XmlDeclaration decl = doc.CreateXmlDeclaration("1.0", null, null);
XmlElement root = doc.DocumentElement;
doc.InsertBefore(decl, doc.DocumentElement);
```

요소를 만들 때는 CreateElement 메서드를 사용합니다.

```
public XmlElement CreateElement ( string name);
public XmlElement CreateElement ( string name, string ns);
public XmlElement CreateElement ( string prefix, string name, string ns);
```

```
XmlElement root = doc.CreateElement("books");
doc.AppendChild(root);
XmlElement elem = doc.CreateElement("book");
elem.InnerText = "XML.NET";
doc.DocumentElement.AppendChild(elem);
```

특성을 추가할 때는 이미 만들어진 요소에 추가합니다. 특성을 추가할 요소 개체에 SetAttribute 메서드를 이용하여 특성을 추가할 수 있습니다. 또 다른 방법으로는 CreateAttribute 메서드를 이용해 XmlAttribute 노드를 만들고 나서 추가할 요소 개체의 특성 컬렉션에 추가할 수도 있습니다.

```
public XmlAttribute CreateAttribute (string name);
public XmlAttribute CreateAttribute (string name, string ns);
public XmlAttribute CreateAttribute (string prefix, string name, string ns);
```

```
elem.SetAttribute("price", "12000");
XmlAttribute attr = doc.CreateAttribute("count");
attr.Value = "50";
elem.Attributes.Append(attr);
```

주석을 만들 때는 CreateComment 메서드를 사용합니다.

```
public XmlComment CreateComment(string data);
```

```
XmlComment comment;
```

```
comment = doc.CreateComment("XML 노드 삽입");
```

```
doc.AppendChild(comment);
```

이 외에도 다른 노드 형식의 개체를 위한 Create 메서드를 제공하고 있습니다.

이렇게 생성한 노드 개체는 XmlDocument 개체의 노드를 삽입하는 메소드를 이용하여 원하는 위치에 노드를 추가할 수 있습니다.

```
public XmlNode InsertBefore ( XmlNode new_node, XmlNode old_node);
```

```
public XmlNode InsertAfter ( XmlNode new_node, XmlNode old_node);
```

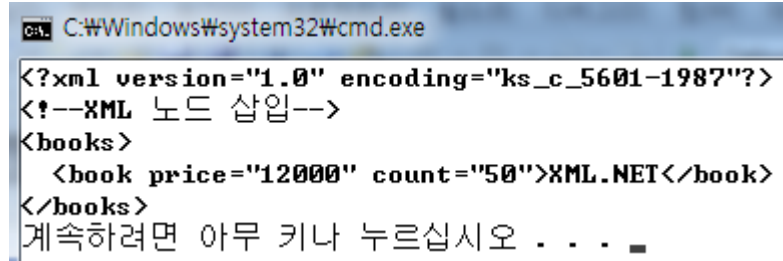
```
public XmlNode AppendChild ( XmlNode new_node);
```

```
public XmlNode PrependChild ( XmlNode new_node);
```

```
public XmlNode Append ( XmlNode new_node);
```

```
static void Main(string[] args)
{
    XmlDocument doc = new XmlDocument();
    XmlDeclaration xmldecl;
    xmldecl = doc.CreateXmlDeclaration("1.0", null, null);
    doc.InsertBefore(xmldecl, doc.DocumentElement);
    XmlComment comment;
    comment = doc.CreateComment("XML 노드 삽입 ");
    doc.AppendChild(comment);
    XmlElement root = doc.CreateElement("books");
    doc.AppendChild(root);
    XmlElement elem = doc.CreateElement("book");
    elem.InnerText = "XML.NET";
    doc.DocumentElement.AppendChild(elem);
    elem.SetAttribute("price", "12000");
    XmlAttribute attr = doc.CreateAttribute("count");
    attr.Value = "50";
    elem.Attributes.Append(attr);
    doc.Save(Console.Out);
    Console.WriteLine();
}
```

[소스 5.2] 노드 삽입 예제 코드



```
C:\Windows\system32\cmd.exe
<?xml version="1.0" encoding="ks_c_5601-1987"?>
<!--XML 노드 삽입-->
<books>
  <book price="12000" count="50">XML.NET</book>
</books>
계속하려면 아무 키나 누르십시오 . . .
```

[그림 5.1] [소스 5.2] 실행 화면

특정 노드의 특정 자식 노드를 제거할 때는 XmlNode의 RemoveChild 메서드를 사용합니다.

```
public XmlNode RemoveChild ( XmlNode node);
```

특정 노드의 모든 자식과 특성을 제거할 때는 XmlNode의 RemoveAll 메서드를 사용합니다.

```
public void RemoveAll ( );
```

특성 컬렉션에서 특정 특성을 제거할 때는 특성 컬렉션의 Remove 메서드를 사용합니다.

```
public XmlAttribute Remove ( );
```

특성 컬렉션에서 모든 특성을 제거할 때는 특성 컬렉션의 RemoveAll 메서드를 사용합니다.

```
public void RemoveAll ( );
```

특성 컬렉션에서 특정 인덱스의 특성을 제거할 때는 특성 컬렉션의 RemoveAt 메서드를 사용합니다.

```
public XmlAttribute RemoveAt ( int i);
```

특정 요소의 특성 컬렉션을 제거할 때는 XmlElement 개체의 RemoveAllAttributes를 사용합니다.

```
public void RemoveAllAttributes ( );
```

특정 요소의 특성을 이름으로 제거할 때는 XmlElement 개체의 RemoveAttribute 메서드를 사용합니다.

```
public void RemoveAttribute ( string name);
```

```
public void RemoveAttribute ( string name, string ns);
```

특정 요소의 특성을 인덱스로 제거할 때는 XmlElement 개체의 RemoveAttributeAt 메서드를 사용합니다.

```
public XmlNode RemoveAttributeAt ( int i);
```

검색 조건과 일치하는 첫 번째 노드를 선택할 때 XmlNode의 SelectSingleNode 메서드를 사용합니다.

```
public XmlNode SelectSingleNode ( string xpath);
```

```
public XmlNode SelectSingleNode ( string xpath, XmlNamespaceManager nsmg);
```

검색 조건과 일치하는 노드의 목록을 선택할 때 XmlNode의 SelectNodes 메서드를 사용합니다.

```
public XmlNodeList SelectNodes ( string xpath);
```

```
public XmlNodeList SelectNodes ( string xpath, XmlNamespaceManager nsmg);
```

```

class Tracer
{
    XmlDocument doc;

    public void MakeDocument()
    {
        doc = new XmlDocument();
        XmlDeclaration xmldecl;
        xmldecl = doc.CreateXmlDeclaration("1.0", null, null);
        doc.InsertBefore(xmldecl, doc.DocumentElement);

        XmlElement root = doc.CreateElement("books");
        doc.AppendChild(root);
        XmlElement elem = doc.CreateElement("book");
        elem.InnerText = "XML.NET";
        doc.DocumentElement.AppendChild(elem);
        elem.SetAttribute("price", "12000");
        XmlAttribute attr = doc.CreateAttribute("count");
        attr.Value = "50";
        elem.Attributes.Append(attr);
        XmlElement elem2 = doc.CreateElement("book");
        elem2.InnerText = "ADO.NET";
        doc.DocumentElement.AppendChild(elem2);
    }

    public void Trace()
    {
        TestXmlNode_RemoveChild();
        TestXmlNode_RemoveAll();
        TestAttributes_Remove();
        TestAttributes_RemoveAll();
        TestAttributes_RemoveAt();
        TestElement_RemoveAllAttributes();
        TestElement_RemoveAttribute();
        TestElement_RemoveAttributeAt();
    }
}

```

```

private void TestElement_RemoveAttributeAt()
{
    Console.WriteLine("---Start TestElement_RemoveAttributeAt---");
    MakeDocument();
    XmlNode pnode = doc.SelectSingleNode("books");
    XmlNode cnode = pnode.ChildNodes[0];
    XmlElement xelem = cnode as XmlElement;
    if (xelem != null) { xelem.RemoveAttributeAt(0); }
    doc.Save(Console.Out);
    Console.WriteLine();
    Console.WriteLine("---End TestElement_RemoveAttributeAt---");
}

private void TestElement_RemoveAttribute()
{
    Console.WriteLine("---Start TestElement_RemoveAttribute---");
    MakeDocument();
    XmlNode pnode = doc.SelectSingleNode("books");
    XmlNode cnode = pnode.ChildNodes[0];
    XmlElement xelem = cnode as XmlElement;
    if (xelem != null) { xelem.RemoveAttribute("price"); }
    doc.Save(Console.Out);
    Console.WriteLine();
    Console.WriteLine("---End TestElement_RemoveAttribute---");
}

private void TestElement_RemoveAllAttributes()
{
    Console.WriteLine("---Start TestElement_RemoveAllAttributes---");
    MakeDocument();
    XmlNode pnode = doc.SelectSingleNode("books");
    XmlNode cnode = pnode.ChildNodes[0];
    XmlElement xelem = cnode as XmlElement;
    if (xelem != null) { xelem.RemoveAllAttributes(); }
    doc.Save(Console.Out);
    Console.WriteLine();
    Console.WriteLine("---End TestElement_RemoveAllAttributes---");
}

```

```

private void TestAttributes_RemoveAt()
{
    Console.WriteLine("---Start TestAttributes_RemoveAt---");
    MakeDocument();
    XmlNode pnode = doc.SelectSingleNode("books");
    XmlNode cnode = pnode.ChildNodes[0];
    XmlAttributeCollection col = cnode.Attributes;
    col.RemoveAt(0);
    doc.Save(Console.Out);
    Console.WriteLine();
    Console.WriteLine("---End TestAttributes_RemoveAt---");
}

private void TestAttributes_RemoveAll()
{
    Console.WriteLine("---Start TestAttributes_RemoveAll---");
    MakeDocument();
    XmlNode pnode = doc.SelectSingleNode("books");
    XmlNode cnode = pnode.ChildNodes[0];
    XmlAttributeCollection col = cnode.Attributes;
    col.RemoveAll();
    doc.Save(Console.Out);
    Console.WriteLine();
    Console.WriteLine("---End TestAttributes_RemoveAll---");
}

private void TestAttributes_Remove()
{
    Console.WriteLine("---Start TestAttributes_Remove---");
    MakeDocument();
    XmlNode pnode = doc.SelectSingleNode("books");
    XmlNode cnode = pnode.ChildNodes[0];
    XmlAttributeCollection col = cnode.Attributes;
    XmlAttribute attr = col["price"];
    col.Remove(attr);
    doc.Save(Console.Out);
    Console.WriteLine();
    Console.WriteLine("---End TestAttributes_Remove---");
}

```

```

private void TestXmlNode_RemoveAll()
{
    Console.WriteLine("---Start TestXmlNode_RemoveAll---");
    MakeDocument();
    XmlNode pnode = doc.SelectSingleNode("books");
    pnode.RemoveAll();
    doc.Save(Console.Out);
    Console.WriteLine();
    Console.WriteLine("---End TestXmlNode_RemoveAll---");
}

private void TestXmlNode_RemoveChild()
{
    Console.WriteLine("---Start TestXmlNode_RemoveChild---");
    MakeDocument();
    XmlNode pnode = doc.SelectSingleNode("books");
    XmlNode cnode = pnode.ChildNodes[0];
    pnode.RemoveChild(cnode);
    doc.Save(Console.Out);
    Console.WriteLine();
    Console.WriteLine("---End TestXmlNode_RemoveChild---");
}

}

class Program
{
    static void Main(string[] args)
    {
        Tracer tracer = new Tracer();
        tracer.Trace();
    }
}

```

[소스 5.3] 노드 제거 예제 코드

## 5. 4 DOM 모델로 OPEN API 활용

OPEN API는 표준 웹 프로토콜을 이용하여 프로그램 개발에 사용할 수 있는 공개 API 입니다. 일반적으로 현실 세계에 있는 실제 데이터를 수집한 단체에서 개발자들이 개발하는 응용에서 사용할 수 있게 제공하고 있습니다. 이러한 OPEN API에서는 웹 쿼리로 검색 질의를 보내면 XML 문서 형태로 결과를 보내는 것이 일반적입니다.

여기에서는 Naver 검색 API를 이용하여 도서를 검색하는 간단한 콘솔 응용을 제작하는 과정을 소개할게요.

OPEN API를 사용하기 위해서는 질의를 어떻게 보내고 결과가 어떠한 형태로 되는지 먼저 알아야 합니다. 여기서 다룰 Naver 검색 API는 사용하기 위해서는 Naver 개발자 센터 홈페이지에서 오픈 API를 사용하기 위한 키를 발급받아야 합니다.

NAVER DeveloperCenter

오픈소스 | 오픈API | nFORG

튜토리얼 >

키 이용등록/수정 >

- 이용약관

키 이용등록/수정

컨텐츠 API >

데이터 API >

지도 API >

검색 API >

기능 API >

FAQ GO

제휴신청 GO

### 키 이용등록/수정

이 곳은, 오픈 API 를 이용해 새로운 사이트 및 프로그램을 개발하시기 위한 키(Key)를 발급받고 확인하

키 발급에는 일반키 발급 및 지도키 발급으로 2가지가 있습니다.

일반키는 지도 API를 제외한 모든 API 서비스를 이용하실 수 있습니다.  
지도 서비스 개발을 위해서는 반드시 지도 API용 키를 별도로 발급 받으시기 바랍니다.

일반키 발급 지도키 발급

+ 일반키는 지도 API를 제외한 모든 API 서비스를 이용하실 수 있습니다.  
지도 서비스 개발을 위해서는 반드시 지도 API용 키를 별도로 발급 받으시기 바랍니다.

이메일 XXXX @ XXXX.COM

연락처 XXX - XXX - XXX

사용용도

API를 이용하여 어떤 서비스를 구현하시고자 하는지 적어주세요.  
고객님들의 의견은 더 좋은 네이버 오픈API 를 만들어 가는 데 큰 도움이 됩니다.

웹 based에서 UCC를 제작하고 편집하고 다양한 콘텐츠를 포함할 수 있으며 UCC시청 중 원하는 콘텐츠를 쉽게 갈무리 할 수 있고 해당 콘텐츠에 관련 정보를 쉽게 검색할 수 있고 관련 홈페이지와 쉽게 연동하는 사용자 컨트롤

확인

현재 사용중인 키 : 67dd96cd71e5fbc65675baff118728010 삭제하기

오늘 키 사용/제한쿼리 : 0/25000

[그림 5.2] 키 발급



키를 발급 받았으면 검색 API에 사용할 질의를 확인해 봅시다. 네이버 개발자 센터에서는 OPEN API 서비스 종류에 따라 검색 질의를 확인할 수 있는 가이드라인을 홈 페이지로 제공하고 있습니다.

**오pen API**

**검색 API**

지도 API

JavaScript

StaticMap

Android

iOS

네이버 로그인

단축 URL API

OAuth

카페 API

미투데이 API

Syndication API

**검색 API > 책**

제목뿐만 아니라 저자, 출판사, 카테고리별 검색 등 다양한 옵션이 제공되는 책 API를 이용해 나만의 도서관을 만들어 보세요.

**1. 요청 URL (request url)**

`http://openapi.naver.com/search`

**2. 요청 변수 (request parameter)**

**2.1 기본검색**

요청 변수	값	설명
key	string (필수)	이용 등록을 통해 받은 key 스트링을 입력합니다.
target	string (필수) : book	서비스를 위해서는 무조건 지정해야 합니다.
query	string (필수)	검색을 원하는 질의, UTF-8 인코딩입니다.
display	integer : 기본값 10, 최대 100	검색결과 출력건수를 지정합니다. 최대 100 까지 가능합니다.
start	integer : 기본값 1, 최대 1000	검색의 시작위치를 지정할 수 있습니다. 최대 1000 까지 가능합니다.

[그림 5.3] 검색 질의 확인

검색 질의는 요청 URL과 요청 변수의 조합으로 작성합니다. 요청 URL과 요청 변수 사이에는 ? 문자로 구분을 하고 요청 변수와 요청 변수 사이는 &로 구분하고 요청 변수와 요청 변수의 값은 =로 구분하여 질의를 나타냅니다.

만약 XML.NET이라는 책을 검색하고자 한다면 "http://openapi.naver.com/search?key=[발급받은 키]&target=book&query=XML.NET" 형태로 질의하면 됩니다.

검색 질의를 작성하는 방법을 알았으면 질의 결과가 어떤 형태로 전달받는지 알아야 합니다. 이에 관한 사항도 네이버 개발 센터에서 확인할 수 있습니다.

### 3. 출력 결과 필드 (response field)

필드	값	설명
rss	-	디버그를 쉽게 하고 RSS 리더기만으로 이용할 수 있게 하기 위해 만든 RSS 포맷의 컨테이너이며 그 외의 특별한 의미는 없습니다.
channel	-	검색 결과를 포함하는 컨테이너입니다. 이 안에 있는 title, link, description 등의 항목은 참고용으로 무시해도 무방합니다.
lastBuildDate	datetime	검색 결과를 생성한 시간입니다.
total	integer	검색 결과 문서의 총 개수를 의미합니다.
start	integer	검색 결과 문서 중, 문서의 시작점을 의미합니다.
display	integer	검색된 검색 결과의 개수입니다.
item	-	개별 검색 결과이며, title, link, description을 포함합니다.
title	string	검색 결과 문서의 제목을 나타냅니다. 제목에서 검색어와 일치하는 부분은 태그로 감싸져 있습니다.
link	string	검색 결과 문서의 하이퍼텍스트 link를 나타냅니다.
image	string	썸네일 이미지의 URL입니다. 이미지가 있는 경우만 나타납니다.
author	string	저자정보입니다.
price	integer	정가 정보입니다. 절판도서 등으로 가격이 없으면 나타나지 않습니다.
discount	integer	할인 가격정보입니다. 절판도서 등으로 가격이 없으면 나타나지 않습니다.
publisher	string	출판사 정보입니다.
pubdate	date	출간일 정보입니다.
isbn	integer	ISBN 넘버입니다.
description	string	검색 결과 문서의 내용을 요약한 패시지 정보입니다. 문서 전체의 내용은 link를 따라가면 읽을 수 있습니다. 패시지에서 검색어와 일치하는 부분은 태그로 감싸져 있습니다.

[그림 5.4] 검색 질의 결과 가이드

응용의 기본 알고리즘을 살펴봅시다.

도서 검색을 위해 먼저 검색할 도서명을 입력받습니다. 그리고 도서 검색 개체를 생성할게요. 도서 검색 개체는 검색할 도서명을 입력인자로 받아 검색 요청을 하고 검색한 결과를 출력하기로 할게요.

```
class Program
{
    static void Main(string[] args)
    {
        string book_name = string.Empty;
        Console.WriteLine("검색할 도서명을 입력하세요.");
        book_name = Console.ReadLine();
        BookSearcher bs = new BookSearcher();
        bs.Find(book_name);
        bs.View();
    }
}
```

도서 검색을 위한 클래스로 BookSearcher 클래스와 검색한 도서를 위한 Book 클래스를 정의할게요.

도서 검색을 위한 BookSearcher 클래스는 생성자와 검색 요청하는 Find 메서드, 결과를 요청하는 View 메서드를 제공합니다.

```
class BookSearcher
{
    internal BookSearcher();
    internal void Find(string book_name);
    public void View();
}
```

생성자에서는 검색 결과를 보관할 컬렉션 개체를 생성하기로 할게요.

```
internal BookSearcher()
{
    ar = new ArrayList();
}
```

도서를 검색하는 Find 메서드에서는 입력 받은 도서명으로 검색 질의를 만듭니다.

```
internal void Find(string book_name)
{
    this.book_name = book_name;
    string query = string.Empty;
    query = string.Format("http://openapi.naver.com/search?key={0}&query={1}&target=book&start=1&display=10", "1e6a5c19f8af1b4b8f85240f2f77e398 ", book_name);
```

도서 검색 질의를 만들었으면 XmlDocument 개체를 생성한 후에 질의에 의해 Xml 문서를 로드합니다.

```
XmlDocument doc = new XmlDocument();
doc.Load(query);
```

도서 검색 문서에 루트 노드인 "rss" 노드를 선택하고 자식 노드인 "channel" 노드를 선택합니다.

```
XmlNode node = doc.SelectSingleNode("rss");
XmlNode n = node.SelectSingleNode("channel");
```

그리고 반복해서 "item" 노드를 선택하여 해당 노드를 입력 인자로 Book 개체를 생성합니다. 생성한 Book 개체는 검색 결과를 보관하는 컬렉션에 추가합니다. 개체를 생성하는 것은 Book 클래스의 정정 메서드 MakeBook을 정의하여 생성하기로 할게요.

```
Book book = null;
foreach (XmlNode el in n.SelectNodes("item"))
{
    book = Book.MakeBook(el);
    ar.Add(book);
}
```

검색 결과를 출력하는 View 메서드에서는 질의에 사용한 도서명을 출력한 후에 검색 결과를 보관한 컬렉션의 책의 정보를 출력하기로 할게요.

```
Console.WriteLine("질의문:{0}", book_name);
foreach (Book book in ar)
{
    Console.WriteLine(book.Title);
    Console.WriteLine("✎저자:{0}", book.Author);
    Console.WriteLine("✎출판사:{0}", book.Publisher);
    Console.WriteLine("✎가격:{0}", book.Price);
    Console.WriteLine("✎설명:{0}", book.Description);
}
```

검색 결과인 책은 Book 클래스로 정의할게요. Book 클래스에는 검색 결과를 설정하고 얻어오는 멤버 속성과 생성자를 제공합니다. 각 속성의 설정을 위한 set 블록은 Book 클래스 내에서 접근 가능하게 private으로 설정하고 얻어오는 get 블록은 프로젝트 내에서 접근할 수 있게 internal로 지정할게요.

```
internal string Title
{
    get;
    private set;
}
internal string Author
{
    get;
    private set;
}
internal int Price
{
    get;
    private set;
}
internal string Publisher
{
    get;
    private set;
}
internal string Description
{
    get;
    private set;
}
```

생성자 메서드도 Book 클래스 내에서 접근 가능하게 private으로 설정할게요. 실제 Book 개체를 생성하는 것은 정적 메서드인 MakeBook을 이용하게 할 것입니다.

```
internal Book(string title, string author, int price, string publisher, string description)
{
    Title = title;
    Author = author;
    Price = price;
    Publisher = publisher;
    Description = description;
}
```

XmlNode 개체를 입력인자로 전달받아 Book 개체를 생성하여 반환하는 MakeBook 메서드의 알고리즘은 다음과 같습니다. 입력 인자로 전달받은 XmlNode 개체는 "item" 요소에 관한 XmlNode 개체입니다. 질의 결과는 item 자식 요소로 "title", "price", "publisher", "description" 등이 있습니다. 여기에서는 각 요소의 내부 텍스트를 얻어와서 "<"와 ">" 사이에 있는 문자열을 필터링하는 ConvertString 메서드를 이용하여 결과를 얻어 올게요. 그리고 난 후에 얻은 결과를 입력 인자로 Book 개체를 생성하기로 합니다.

```
static internal Book MakeBook(XmlNode xn)
{
    string title = string.Empty;
    string author = string.Empty;
    int price = 0;
    string publisher = string.Empty;
    string description = string.Empty;

    XmlNode title_node = xn.SelectSingleNode("title");
    title = ConvertString(title_node.InnerText);
    XmlNode price_node = xn.SelectSingleNode("price");
    price = int.Parse(price_node.InnerText);
    XmlNode publisher_node = xn.SelectSingleNode("publisher");
    publisher = ConvertString(publisher_node.InnerText);
    XmlNode description_node = xn.SelectSingleNode("description");
    description = ConvertString(description_node.InnerText);

    return new Book(title, author, price, publisher, description);
}

private static string ConvertString(string str)
{
    int sindex = 0;
    int eindex = 0;
    while (true)
    {
        sindex = str.IndexOf('<');
        if (sindex == -1)
        {
            break;
        }
        eindex = str.IndexOf('>');
        str = str.Remove(sindex, eindex - sindex + 1);
    }
    return str;
}
```

다음은 도서 검색 응용 프로그램 코드입니다.

```
using System;
using System.Xml;
using System.Collections;

namespace 예제_5_4_도서_검색_응용
{
    class Book
    {
        internal string Title
        {
            get;
            private set;
        }
        internal string Author
        {
            get;
            private set;
        }
        internal int Price
        {
            get;
            private set;
        }
        internal string Publisher
        {
            get;
            private set;
        }
        internal string Description
        {
            get;
            private set;
        }
    }
}
```

```

static internal Book MakeBook(XmlNode xn)
{
    string title = string.Empty;
    string author = string.Empty;
    int price = 0;
    string publisher = string.Empty;
    string description = string.Empty;

    XmlNode title_node = xn.SelectSingleNode("title");
    title = ConvertString(title_node.InnerText);

    XmlNode price_node = xn.SelectSingleNode("price");
    price = int.Parse(price_node.InnerText);

    XmlNode publisher_node = xn.SelectSingleNode("publisher");
    publisher = ConvertString(publisher_node.InnerText);

    XmlNode description_node = xn.SelectSingleNode("description");
    description = ConvertString(description_node.InnerText);

    return new Book(title, author, price, publisher, description);
}

internal Book(string title, string author, int price, string publisher, string description)
{
    Title = title;
    Author = author;
    Price = price;
    Publisher = publisher;
    Description = description;
}

```



```

private static string ConvertString(string str)
{
    int sindex = 0;
    int eindex = 0;
    while (true)
    {
        sindex = str.IndexOf('<');
        if (sindex == -1)
        {
            break;
        }
        eindex = str.IndexOf('>');
        str = str.Remove(sindex, eindex - sindex+1);
    }
    return str;
}

public override string ToString()
{
    return Title;
}
}

```

```

class BookSearcher
{
    ArrayList ar = new ArrayList();
    string book_name;

    internal BookSearcher()
    {
        ar = new ArrayList();
    }
}

```

```

internal void Find(string book_name)
{
    this.book_name = book_name;
    string query = string.Empty;
    query = string.Format("http://openapi.naver.com/search?key={0}&query={1}&target=book&start=1&display=10", "1e6a5c19f8af1b4b8f85240f2f77e398 ", book_name);

    XmlDocument doc = new XmlDocument();
    doc.Load(query);

    XmlNode node = doc.SelectSingleNode("rss");
    XmlNode n = node.SelectSingleNode("channel");

    Book book = null;
    foreach (XmlNode el in n.SelectNodes("item"))
    {
        book = Book.MakeBook(el);
        ar.Add(book);
    }
}

public void View()
{
    Console.WriteLine("질의문:{0}", book_name);

    foreach (Book book in ar)
    {
        Console.WriteLine(book.Title);
        Console.WriteLine("著자:{0}", book.Author);
        Console.WriteLine("출판사:{0}", book.Publisher);
        Console.WriteLine("가격:{0}", book.Price);
        Console.WriteLine("설명:{0}", book.Description);
    }
}
}

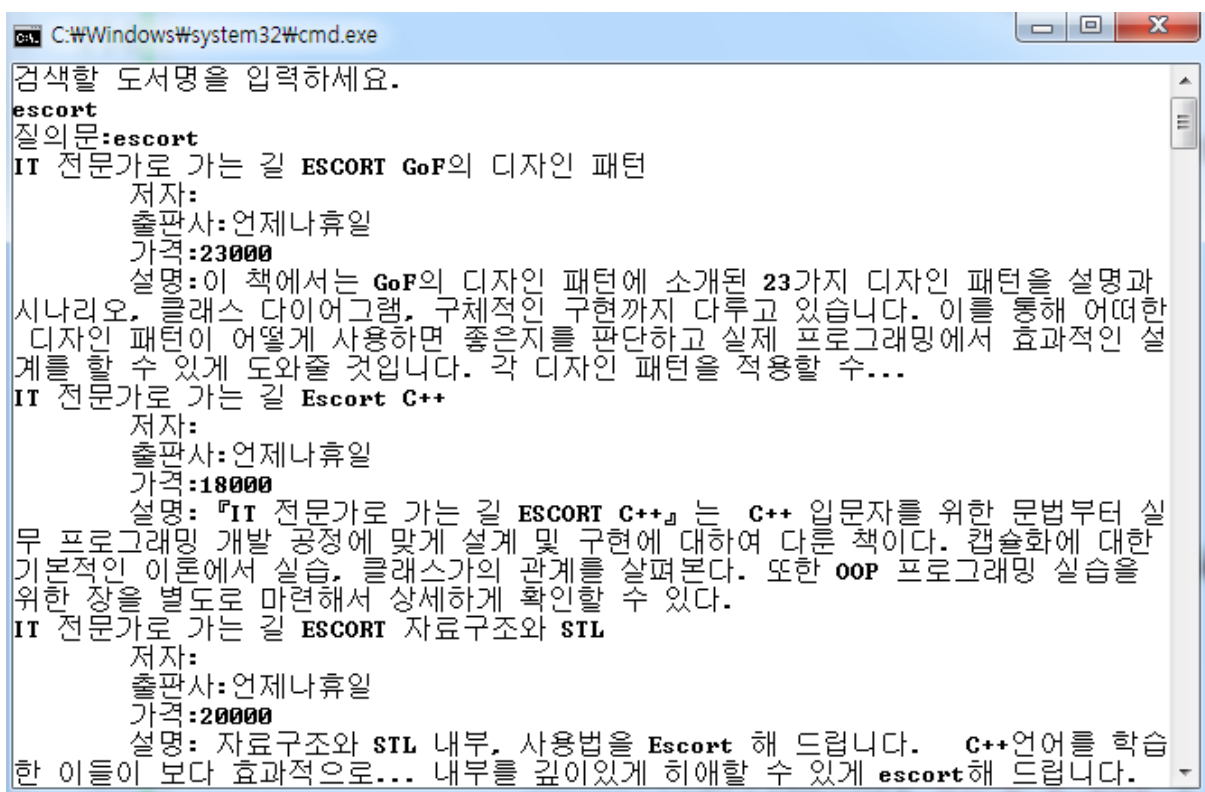
```

```

class Program
{
    static void Main(string[] args)
    {
        string book_name = string.Empty;
        Console.WriteLine("검색할 도서명을 입력하세요.");
        book_name = Console.ReadLine();
        BookSearcher bs = new BookSearcher();
        bs.Find(book_name);
        bs.View();
    }
}

```

[소스 5.4] OPEN API를 이용한 도서 검색 예제 코드



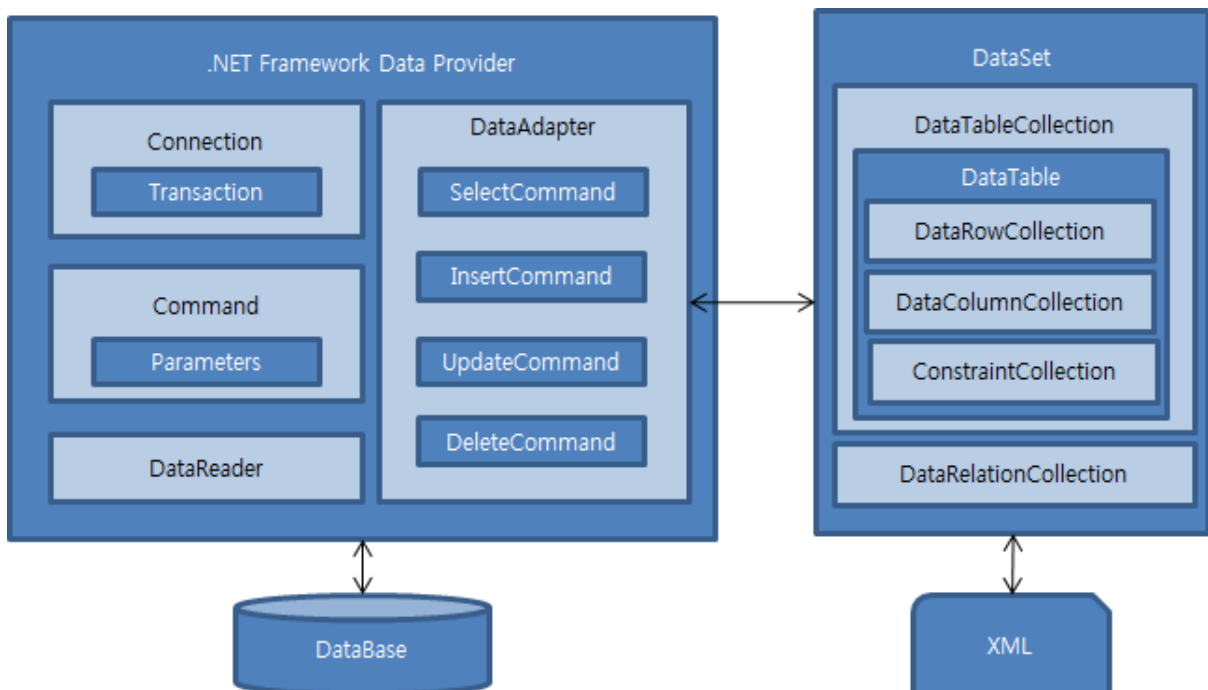
[그림 5.5] [소스 5.4] 실행 결과 (escort로 검색했을 때)

## 6. ADO.NET 구조

### 6. 1 ADO.NET 개요

ADO.NET은 데이터 소스의 종류에 관계없이 일관성 있는 액세스를 제공하는 기술을 말합니다. ADO.NET 데이터 소스에는 SQL Server나 XML 형태로 된 문서 뿐만 아니라 OLE DB 및 ODBC로 접근할 수 있는 다양한 데이터 소스가 있습니다. 이러한 데이터 소스에 접근하고자 할 때 ADO.NET 기술을 사용하면 쉽고 강력하게 데이터 소스에 연결하여 필요한 데이터를 검색하거나 추가 등의 원하는 작업을 할 수 있습니다.

### 6. 2 ADO.NET 구성 요소



[그림 6.1] ADO.NET 구조

ADO.NET 기술은 .NET Framework 데이터 공급자와 DataSet으로 구성하고 있습니다.

.NET Framework 데이터 공급자는 데이터 소스와 연결 및 원하는 작업 처리를 담당하며 DataSet은 데이터 소스와 관계없이 프로그램의 논리적 데이터를 구성하여 원하는 작업 처리를 담당합니다.

.NET Framework 데이터 공급자에는 데이터 소스에 연결을 제공하는 Connection과 데이터 소스에 원하는 질의를 할 수 있는 Command, 데이터 소스를 정방향으로 빠르게 읽기 작업을 할 수 있는 DataReader와 데이터 소스와 DataSet 사이의 연결을 제공하는 DataAdapter가 있습니다.

DataSet은 데이터 소스와 관계없이 관리할 데이터의 구조를 테이블 형태로 디자인 하고 관계를 정의하여 프로그램 내에서 데이터 작업을 쉽게 해 주는 역할을 수행합니다. 그리고 DataSet은 구조와 데이터를 XML 문서 형태로 기록하고 로딩하여 사용할 수 있는 강력한 컬렉션입니다.

## 6. 3 SQL 개요

MS SQL 2008 서버는 Microsoft 사의 DBMS(Database Management System)입니다. DBMS는 데이터 베이스를 관리하는 시스템으로 사용자의 요구에 따라 Database를 조작하고 제어하는 기능을 제공하는 소프트웨어입니다.

Database는 사용자의 요구에 즉각적으로 응답하는 실시간 접근성을 제공하고 삽입, 삭제, 변경 등의 작업으로 유효한 데이터의 변화를 갖게 됩니다. 또한 여러 사용자가 동시에 원하는 데이터에 접근 가능하며 사용자가 직접 data의 물리적 주소에 직접 접근하는 것을 막고 간접적으로 원하는 데이터를 참조할 수 있게 해 줍니다. 이러한 특징을 갖는 Database는 "여러 사용자가 동시에 사용할 수 있게 통합 관리하는 데이터의 집합"이라고 정의할 수 있습니다.

DBMS를 사용하여 데이터 베이스를 관리하면 표준화된 같은 방법으로 데이터 관리가 가능하고 동시에 사용이 가능할 뿐만 아니라 사용자가 정한 규칙을 벗어나는 데이터가 생기지 않게 무결성을 제공합니다. 이 외에도 주기적이거나 원하는 조건에 따라 자동으로 데이터를 백업할 수 있고 보안 등의 장점이 생깁니다.

### 6.3.1 SQL(Structured Query Language)

사용자는 DBMS의 데이터 베이스를 정의하고 조작, 제어하기 위한 언어를 사용하는데 대표적인 것이 SQL입니다. SQL은 테이블을 만들고 삭제하는 등의 작업을 하는 데이터 정의어(DDL)와 데이터를 추가, 삭제 등의 작업을 하는 데이터 조작어(DML)와 특정 사용자에게 권한을 부여하거나 제거하는 데이터 제어어(DCL) 등으로 구분할 수 있습니다.

#### ▷ 데이터 정의어(DDL, Data Definition Language)

CREATE : 테이블 생성

ALTER: 테이블 구조 변경

RENAME: 테이블 이름 변경

TRUNCATE: 테이블 내의 데이터를 삭제

DROP: 테이블 삭제

#### ▷ 데이터 조작어(DML, Data Manipulation Language)

INSERT : 새로운 데이터를 추가

DELETE : 기존의 데이터를 삭제

UPDATE : 기존의 데이터를 변경

SELECT : 데이터를 조회

\*SELECT 문은 데이터 질의어(DQL, Data Query Language)로도 부름

#### ▷ 데이터 제어어(DCL, Data Control Language)

GRANT: 사용자에게 권한 부여

REVOKE: 사용자에게 부여한 권한 제거

COMMIT : 트랜잭션에서 수행한 작업을 확정

ROLLBACK : 트랜잭션을 시작하기 이전 상태로 복원

SAVEPOINT: 트랜잭션의 작업 도중에 특정 지점을 저장(COMMIT이나 ROLLBACK의 기준점으로 사용)

\*COMMIT, ROLLBACK, SAVEPOINT 문은 트랜잭션 제어어(TCL, Transaction Control Language)로도 부름

### 6.3.2 데이터 베이스 스키마(Schema)

데이터 베이스 스키마는 데이터 베이스의 성질을 형식적으로 기술한 것으로 데이터 베이스의 논리적 정의입니다.

데이터 베이스 스키마는 사용자의 관점에 따라 볼 수 있는 데이터 베이스 구조인 외부 스키마와 사용자와 관계없이 실제 데이터 베이스의 구조인 개념 스키마, 저장 장치에 저장하는 구조를 얘기하는 물리 스키마로 구분할 수 있습니다. DBMS 관리자 입장에서는 개념 스키마와 물리 스키마에 관심을 갖으며 DBMS를 사용하는 개발자 입장에서는 개념 스키마와 외부 스키마에 관심을 갖을 것입니다.

컴퓨터 프로그래밍에서 DBMS를 이용하여 데이터 베이스를 구축하는 여러 가지 이유 중에는 개발자가 정의한 데이터 베이스 스키마의 제약 조건에 위배하는 작업을 DBMS에서 방지해 주는 것은 매우 중요하고 개발 비용을 줄이는 요인입니다. 유일성과 무결성을 보장하는 컬럼을 지정하거나 주요 키와 외래 키의 관계를 설정하는 것은 데이터 신뢰성을 높이는 일반적인 제약 조건입니다.

#### ▷ 개체 무결성 (Entity Integrity)

테이블의 구조를 정의할 때 어떠한 컬럼들로 구성할 것인지를 결정합니다. 이 때 중복된 값을 갖지 못하게 컬럼의 속성을 정의하면 같은 값을 갖는 데이터를 추가하는 것을 방지할 수 있습니다. 특히 주요 키(Primary Key)로 지정한 컬럼은 값이 유일하고 널(NULL)을 허용하지 않음을 보장하게 되어 유일성과 무결성을 보장합니다.

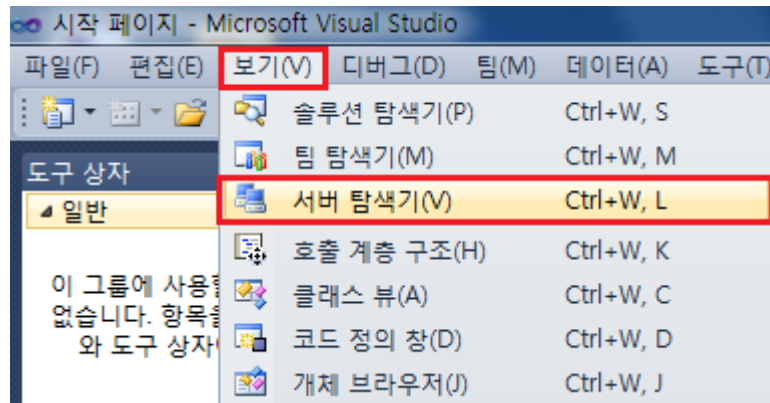
#### ▷ 참조 무결성 (Referntial Integrity)

데이터 베이스에 서로 다른 테이블을 정의할 때 다른 테이블의 주요 키를 자신의 테이블의 외래 키(Foreign Key)로 설정하여 관계를 정의할 수 있습니다. 이처럼 관계를 정의하면 다른 테이블의 데이터의 주요 키 값이 없는 데이터를 추가하는 것을 방지할 수 있습니다.

이와 같은 무결성 제약을 DBMS가 해 줌으로써 데이터 신뢰성과 무결성을 유지하기 위한 개발자의 개발 비용을 줄일 수 있습니다.

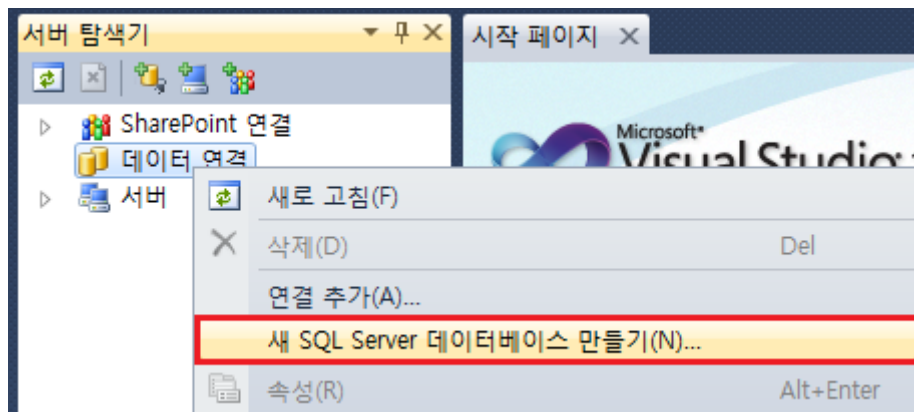
## 6. 4 서버 탐색기를 이용하여 MS SQL Server 에 DB 구축하기

여기에서는 Microsoft Visual Studio 2010에서 서버 탐색기를 이용하여 DB를 구축하는 과정을 간략하게 소개할게요. 먼저 Microsoft Visual Studio 2010 메뉴에서 보기 => 서버 탐색기를 선택합니다.



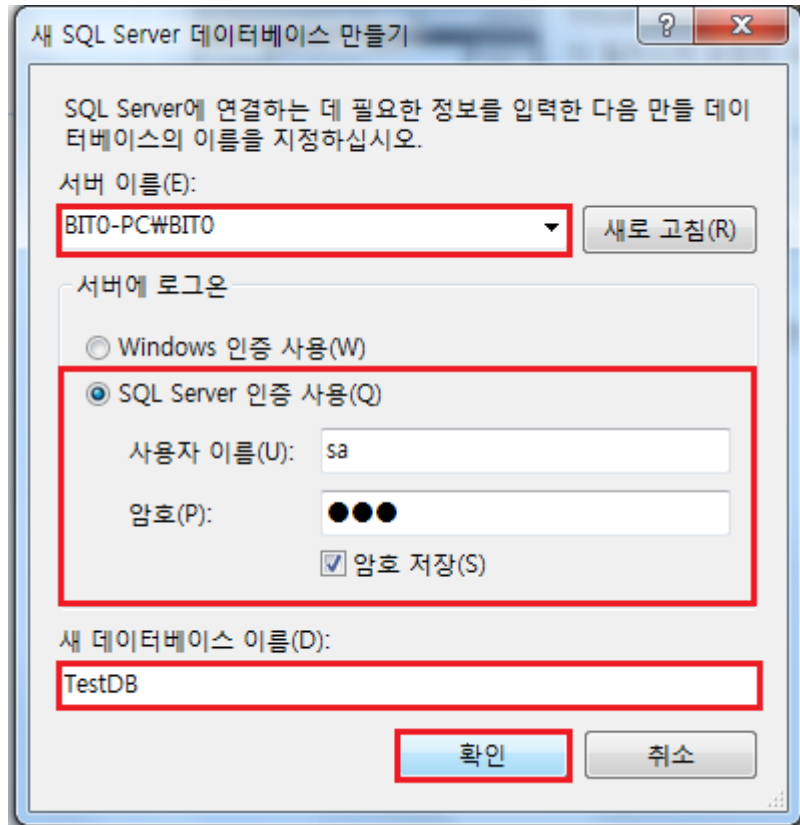
[그림 6.2] 서버 탐색기 선택

서버 탐색기를 이용하면 이미 만들어져 있는 데이터 베이스에 연결하거나 새로운 데이터 베이스를 만들 수 있습니다. 여기에서는 새로운 데이터 베이스를 만들어서 사용해 봅시다. 여러분께서는 서버 탐색기 창에 데이터 연결에 마우스 커서를 이동한 후에 오른쪽 마우스 버튼을 클릭하세요. 클릭하면 나오는 컨텍스트 메뉴에서 새 SQL Server 데이터 베이스 만들기를 선택합니다.



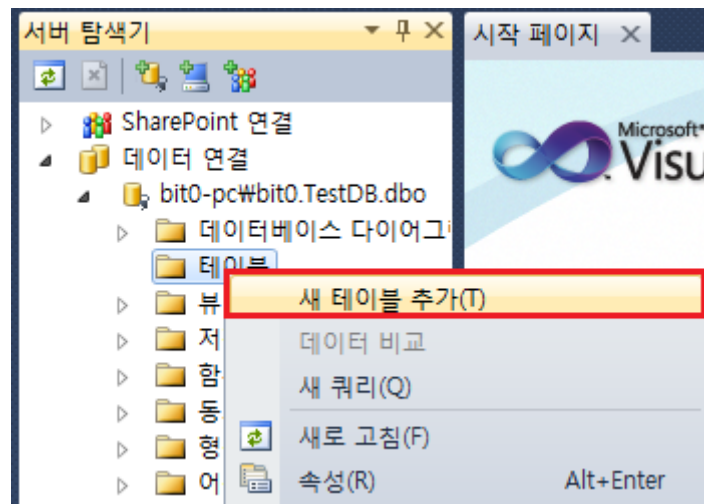
[그림 6.3] 새 SQL Server 데이터 베이스 만들기

새 SQL Server 데이터 베이스 만들기 창이 뜨면 설치되어 있는 MS SQL 서버를 선택하시고 서버를 사용하기 위해 로그인을 지정하세요. Windows 인증 사용을 선택하거나 SQL Server 인증 사용을 선택할 수 있습니다. 여기에서는 SQL Server 인증 사용을 선택할게요. 여러분은 SQL Server 설치 시에 설정하였거나 이 후에 추가한 사용자 계정과 암호를 입력하세요. 그리고 새로 만들 데이터 베이스 이름을 입력한 후에 확인 버튼을 누르면 새로운 데이터 베이스를 생성할 수 있습니다.



[그림 6.4] 새 SQL Server 데이터 베이스 만들기

이번에는 새로 만든 데이터 베이스에 새로운 테이블을 추가해 봅시다. 서버 탐색기의 테이블 항목에서 마우스 오른쪽 버튼을 클릭할 시에 나오는 컨텍스트 메뉴에서 새 테이블을 추가합니다.



[그림 6.5] 새 테이블 추가

테이블 추가를 선택하였으면 먼저 테이블을 디자인합니다. 테이블을 디자인한다는 것은 열(Column)을 추가하고 열의 속성 등을 지정하는 것을 말합니다.



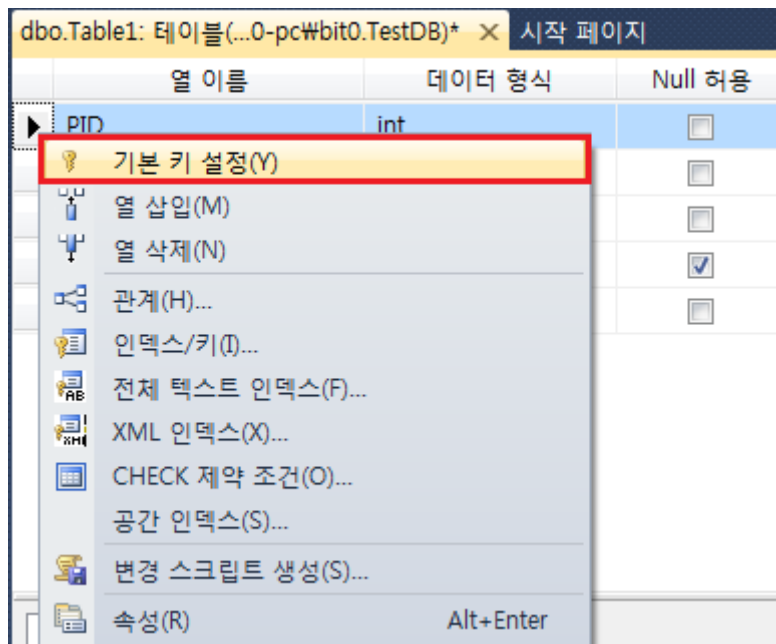
여기서는 상품 테이블을 디자인해 볼게요. 상품 테이블에는 상품의 일련 번호인 PID와 상품 이름(PNAME), 가격(Price), 설명(Description)에 해당하는 열을 추가할게요.

dbo.Table1: 테이블(...0-pc\bit0.TestDB)\* × 시작 페이지

열 이름	데이터 형식	Null 허용
PID	int	<input type="checkbox"/>
PNAME	varchar(50)	<input type="checkbox"/>
Price	int	<input type="checkbox"/>
Description	varchar(MAX)	<input checked="" type="checkbox"/>

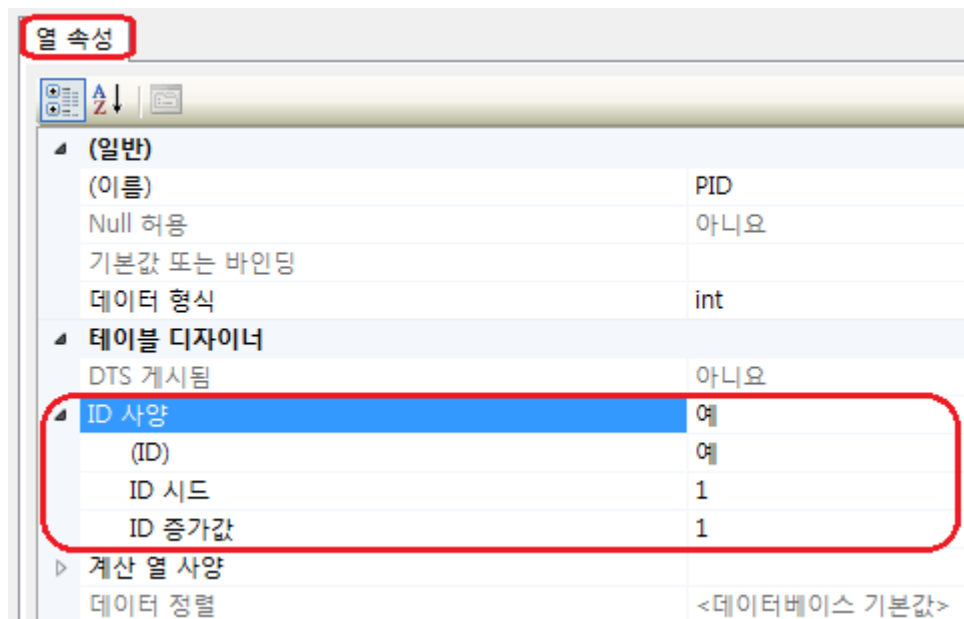
[그림 6.6] 상품 테이블 디자인

상품 아이디는 기본 키(Primary Key)로 설정합니다. 다음 그림처럼 컨텍스트 메뉴에서 기본 키 설정을 선택하세요.



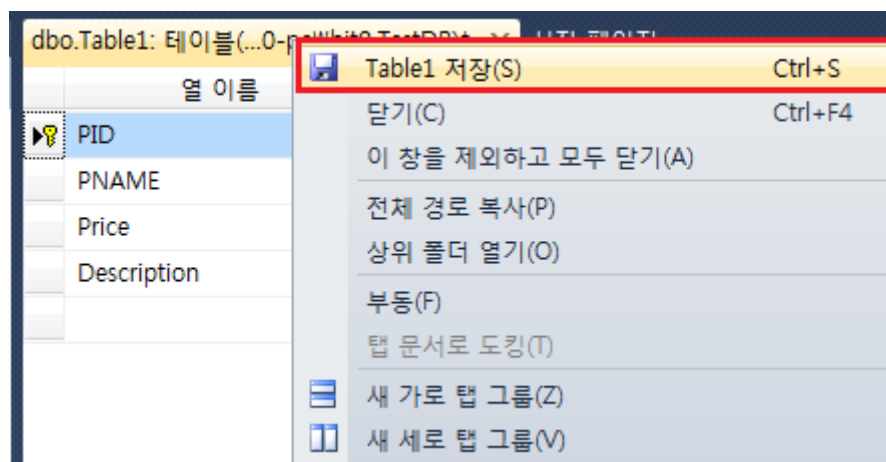
[그림 6.7] 기본 키 설정

그리고 속성 창을 이용하여 열의 세부 속성을 지정하세요.



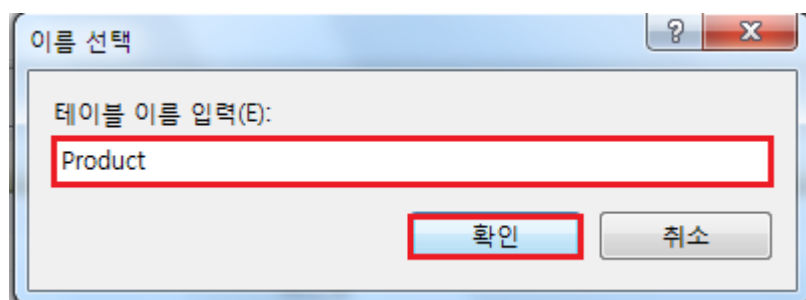
[그림 6.8] 열 속성 지정

테이블을 디자인한 후에 컨텍스트 메뉴를 통해 테이블을 저장합니다.



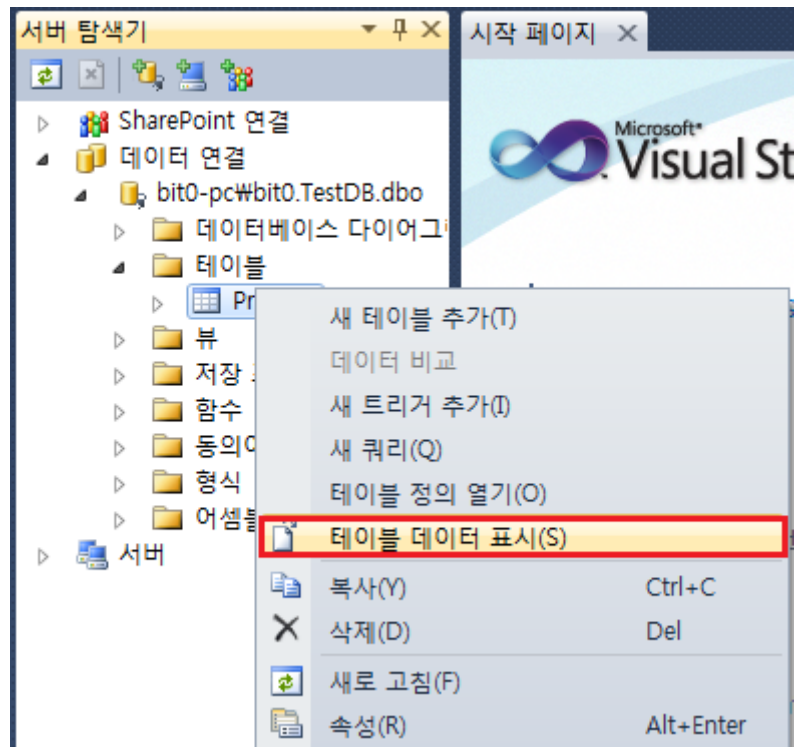
[그림 6.9] 테이블 저장

테이블 저장을 선택한 후에 테이블 이름을 결정하면 테이블이 만들어집니다.



[그림 6.10] 테이블 이름 선택

이제 새로 만들어진 Product 테이블에 상품을 추가할게요. 테이블을 선택한 후에 컨텍스트 메뉴를 띄운 후에 테이블 데이터 표시를 선택하세요. 참고로 테이블 정의 열기를 선택하면 테이블을 디자인할 수 있습니다.



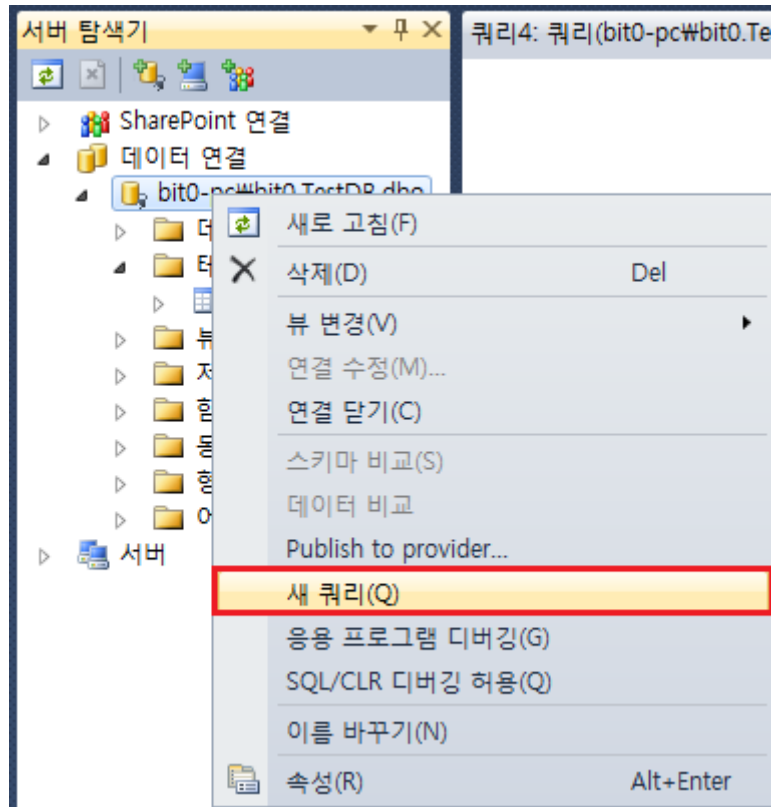
[그림 6.11] 테이블 데이터 표시

테이블 데이터 표시를 선택하면 쉽게 데이터를 추가할 수 있는 탭이 생깁니다. 여러분께서는 몇 개의 상품 데이터를 추가해 보세요.

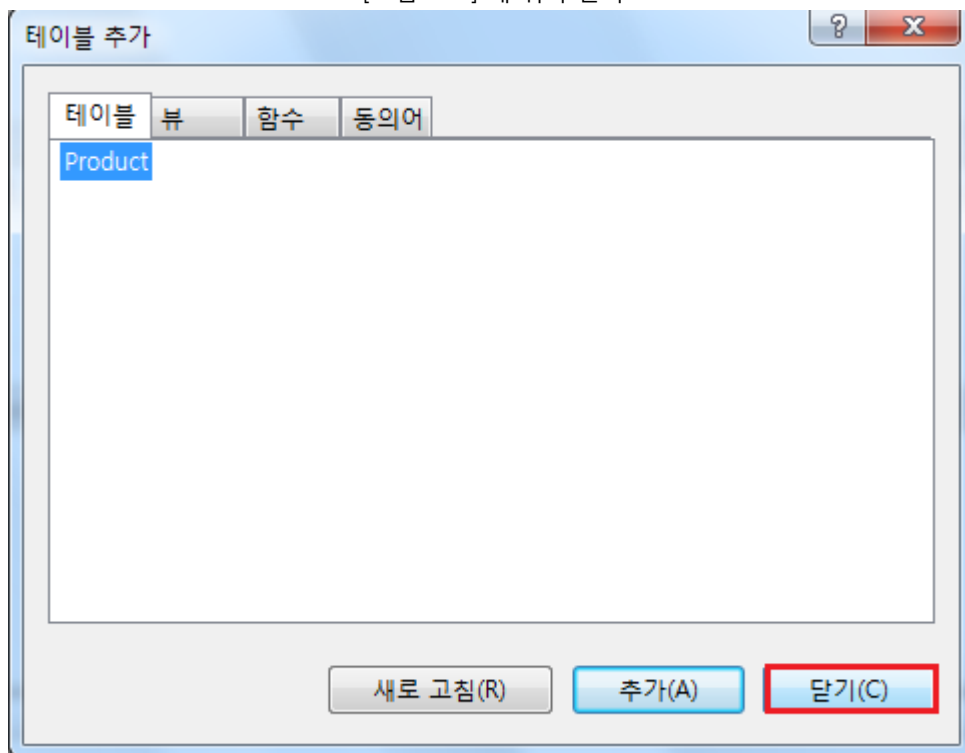
Product: 쿼리(bit0-pc#bit0.TestDB) 시작 페이지				
	PID	PNAME	Price	Description
	1	Escort ADO.NET	25000	MS SQL에서 A...
▶*	NULL	NULL	NULL	NULL

[그림 6.12] 상품 데이터 추가

이번에는 쿼리문을 이용하여 테이블을 추가하는 것을 해 봅시다. 여기서는 고객 테이블을 추가하는 것을 보여드릴게요. 데이터 베이스의 컨텍스트 메뉴에서 새 쿼리를 선택하세요. 그리고 테이블 추가 창에서 닫기 버튼을 누르세요.



[그림 6.13] 새 쿼리 선택

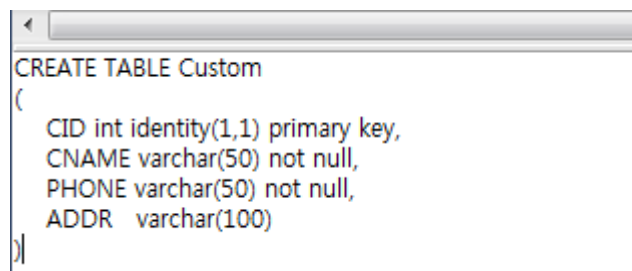


[그림 6.14] 테이블 추가 창에서 닫기 버튼 선택

그리고 테이블을 생성하는 쿼리문을 작성합니다. 테이블을 생성하는 쿼리문은 다음과 같습니다.

```
CREATE TABLE [테이블 명]
(
    [컬럼 이름] [컬럼 형식 및 속성 지정],
    [컬럼 이름] [컬럼 형식 및 속성 지정],
    ...
)
```

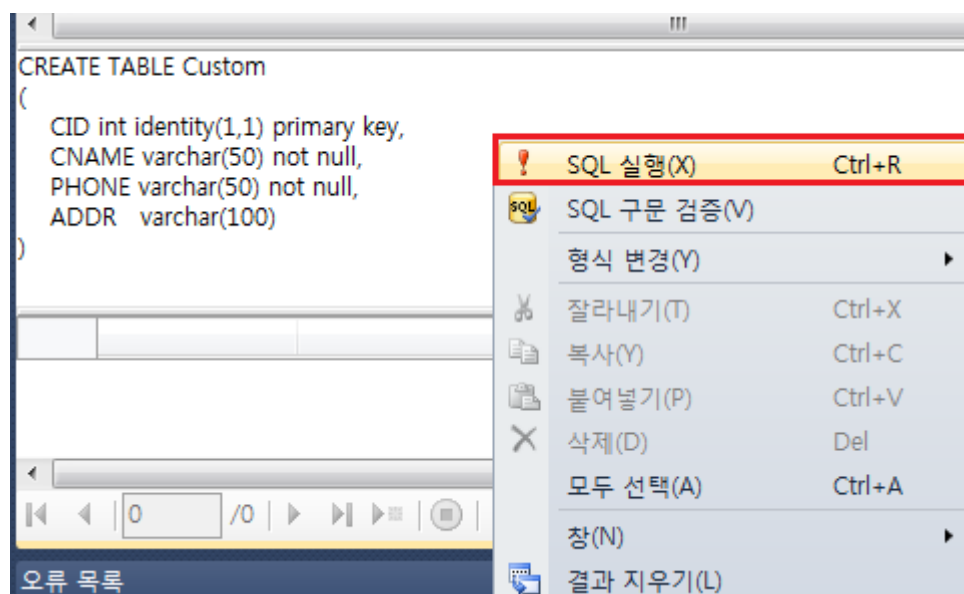
이 책에서는 간략하게 SQL 쿼리문을 다루기는 하지만 깊이 있게 다루지는 않습니다.



```
CREATE TABLE Custom
(
    CID int identity(1,1) primary key,
    CNAME varchar(50) not null,
    PHONE varchar(50) not null,
    ADDR varchar(100)
)
```

[그림 6.15] 테이블 생성 쿼리문 작성

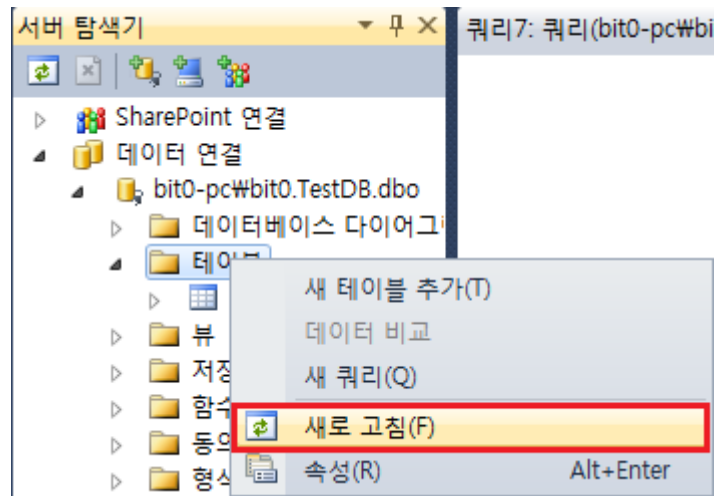
쿼리문을 작성하였으면 컨텍스트 메뉴에서 SQL 실행을 선택하세요.



[그림 6.16] SQL 실행

만약, 쿼리 정의 다음 창이 뜨셔도 계속 버튼을 누르면 쿼리를 실행합니다. 물론 쿼리문을 잘못 작성하면 동작하지 않습니다.

쿼리를 실행하더라도 서버 탐색기에 추가한 테이블이 바로 보이지 않습니다. 테이블의 컨텍스트 메뉴에서 새로 고침을 선택하셔야 새로 추가한 테이블을 볼 수 있습니다.



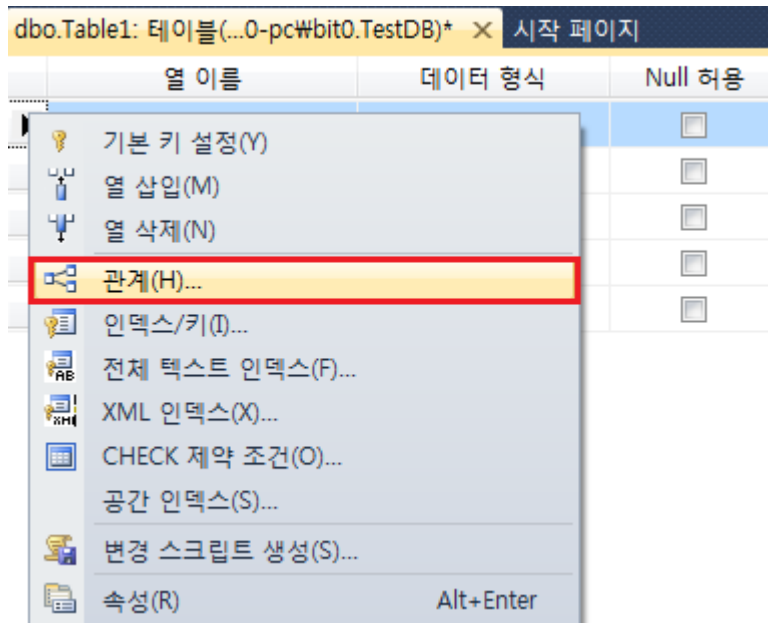
[그림 6.17] 테이블의 컨텍스트 메뉴에서 새로 고침 선택

고객 테이블과 상품 테이블을 추가하였으면 판매 테이블을 추가해 봅시다. 여기서는 처음에 추가한 상품 테이블을 만드는 방법으로 판매 테이블을 추가할게요. 추가할 판매 테이블은 CID, PID와 판매 개수(COUNT), 판매 일시(SaleDate)로 구성할게요.

dbo.Table1: 테이블(...0-pc\bit0.TestDB)* 시작 페이지			
	열 이름	데이터 형식	Null 허용
	CID	int	<input type="checkbox"/>
	PID	int	<input type="checkbox"/>
	COUNT	int	<input type="checkbox"/>
▶	SaleDate	datetime	<input type="checkbox"/>
			<input type="checkbox"/>

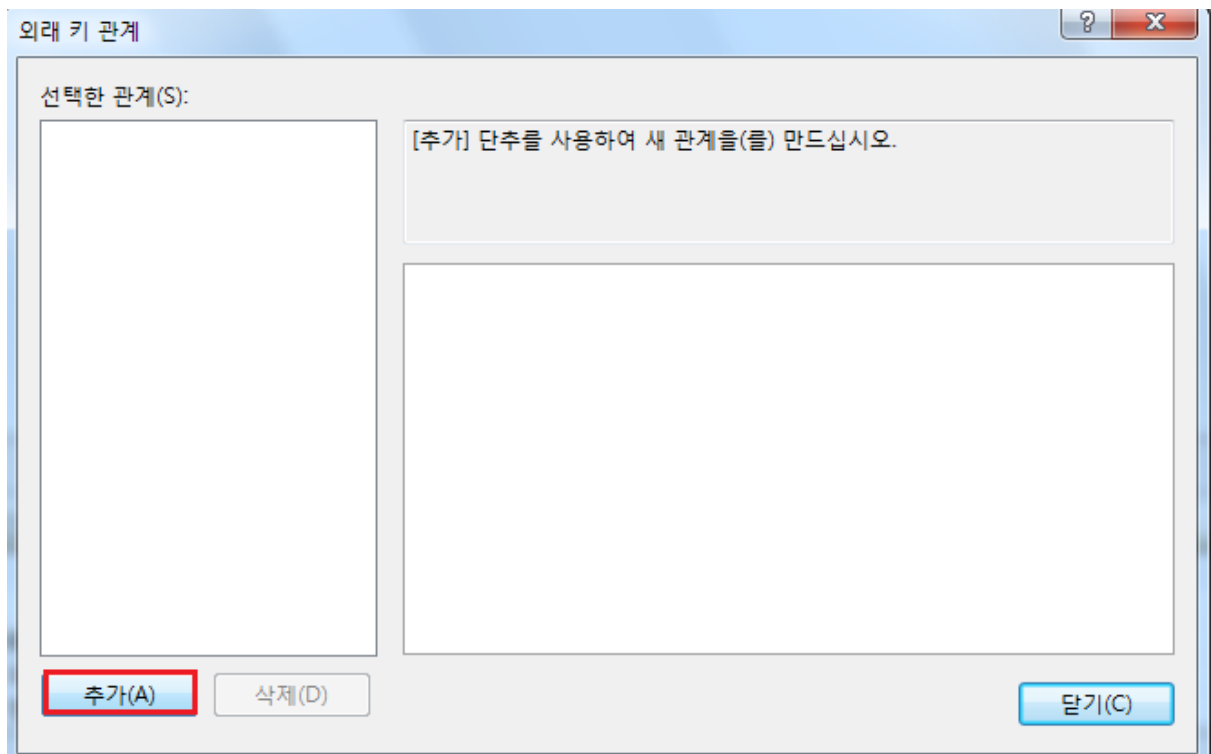
[그림 6.18] 판매 테이블 디자인

고객 테이블의 CID와 판매 테이블의 CID 사이의 관계와 상품 테이블의 PID와 판매 테이블의 PID 사이의 관계를 설정합니다. 관계를 설정하면 고객 테이블에 존재하지 않는 CID나 상품 테이블에 존재하지 않는 PID를 값으로 하는 데이터를 판매 테이블에 추가할 수 없게 됩니다. 관계를 추가하기 위해 컨텍스트 메뉴에서 관계를 선택하세요.



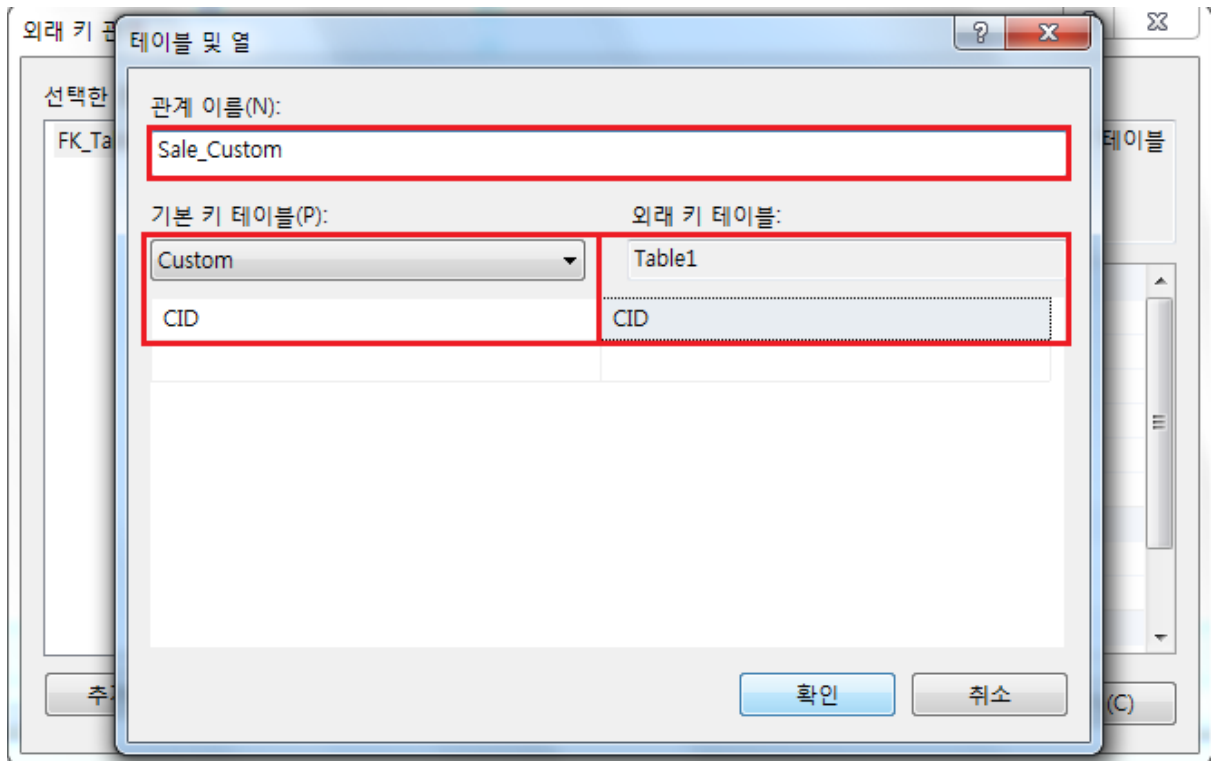
[그림 6.19] 컨텍스트 메뉴에서 관계 선택

그리고 구체적인 관계를 추가하기 위해 외래 키 관계 창에서 추가를 선택하세요.

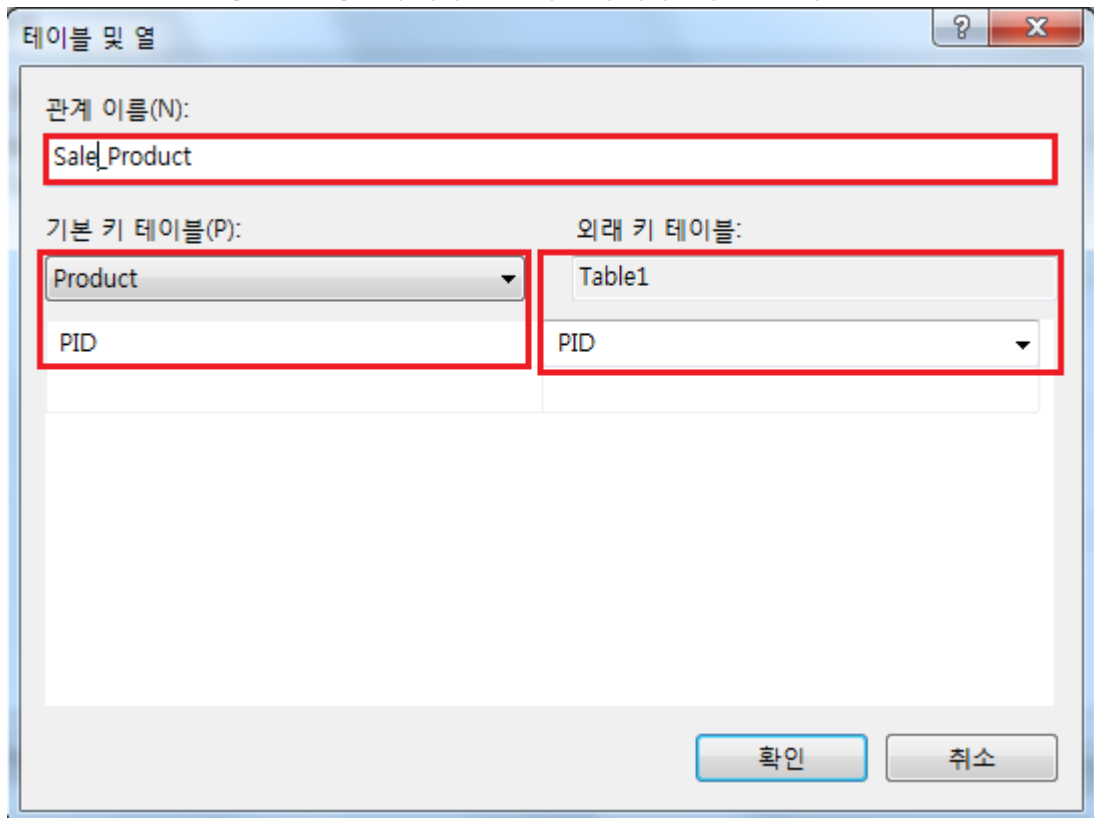


[그림 6.19] 외래 키 관계 창에서 추가 선택

테이블 및 열 사양 항목을 선택하면 기본 키 테이블의 어떤 항목과 외래 키 테이블의 어느 항목을 관계로 지정할 것인지 선택할 수 있습니다. 같은 방법으로 상품 테이블의 PID와 판매 테이블의 PID 관계도 추가하세요.



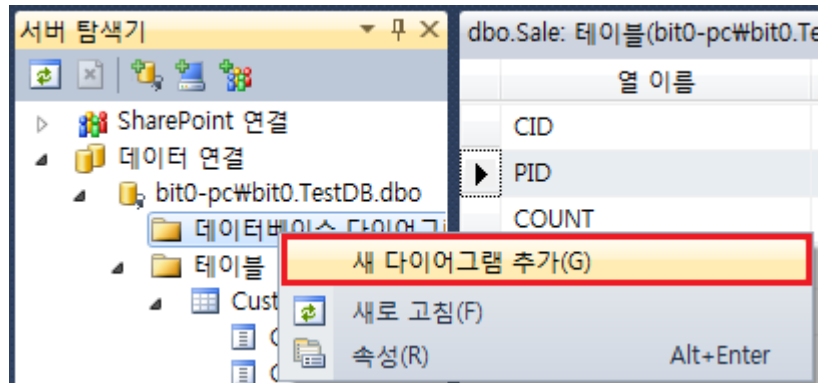
[그림 6.20] 고객 테이블 CID와 판매 테이블의 CID 관계 설정



[그림 6.21] 상품 테이블의 PID와 판매 테이블의 PID 관계 설정

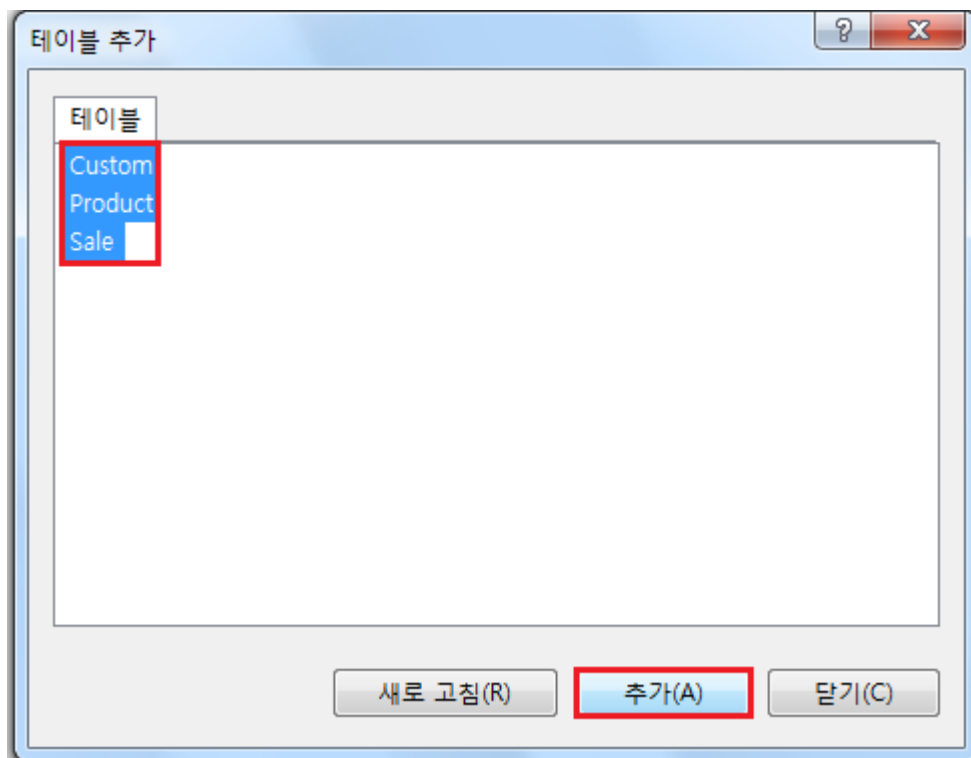


Microsoft Visual Studio 2010의 서버 탐색기에서는 데이터 베이스에 추가한 테이블과 관계를 시각적으로 보기 쉽게 다이어그램을 제공합니다. 이를 위해 서버 탐색기의 컨텍스트 메뉴에서 새 다이어그램 추가를 선택하세요.



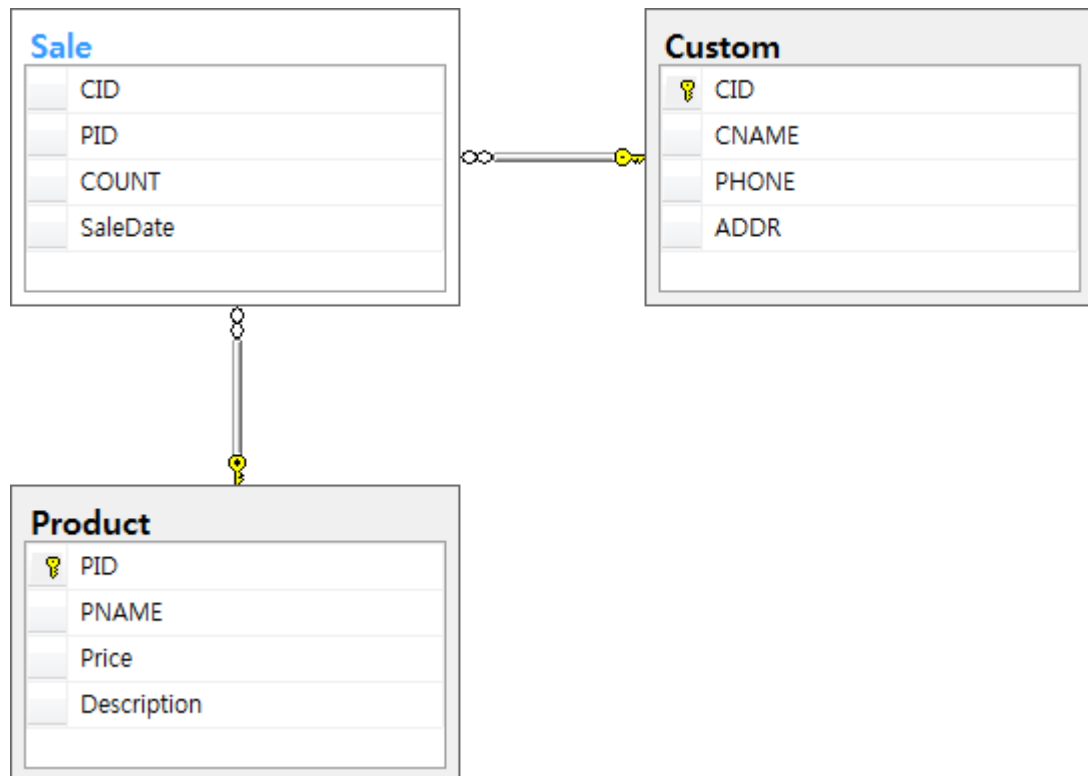
[그림 6.22] 새 다이어그램 추가 선택

그리고 테이블 추가창에서 다이어그램에 도식할 테이블을 선택한 후에 추가 버튼을 클릭하세요.



[그림 6.23] 다이어그램에 도식할 테이블 추가

이렇게 하면 선택한 테이블의 디자인과 관계를 다이어그램으로 확인할 수 있습니다.



[그림 6.24] 다이어그램

## 6. 5 SQL 쿼리

### 6.5.1 CREATE TABLE

CREATE 쿼리문은 새로운 엔터티(Entity)를 정의할 때 사용합니다. 새 데이터 베이스를 정의하거나 사용자 계정을 정의 및 테이블 등을 정의할 때 사용합니다. 여기에서는 CREATE TABLE로 시작하는 새 테이블을 정의하는 쿼리문을 살펴봅시다.

포맷:

CREATE TABLE

```
[ database_name . [ schema_name ] . | schema_name . ] table_name
[ AS FileTable ]
( { <column_definition> | <computed_column_definition>
  | <column_set_definition> | [ <table_constraint> ] [ ,...n ] } )
[ ON { partition_scheme_name ( partition_column_name ) | filegroup | "default" } ]
[ { TEXTIMAGE_ON { filegroup | "default" } } ]
[ FILESTREAM_ON { partition_scheme_name | filegroup | "default" } ]
[ WITH ( <table_option> [ ,...n ] ) ]
[ ; ]
```

예:

CREATE TABLE Custom

```
(
  CID int identity(1,1) primary key,
  CNAME varchar(50) not null,
  PHONE varchar(50) not null,
  ADDR varchar(100)
)
```

새 테이블을 정의할 때는 필수적으로 테이블 이름을 지정하고 테이블에 속할 열(Column)을 정의해야 합니다. 그리고 선택적으로 테이블이 속할 데이터 베이스 이름과 스키마 이름 등을 지정할 수 있습니다. 또한 파일을 보관하고 관리할 목적으로 테이블을 정의할 때는 FileTable을 만들면 강력한 DBMS의 기능을 제공받을 수 있습니다.

테이블의 열(Column)을 정의할 때는 열 이름과 형식이 오며 열의 다양한 속성과 제약 사항을 선택적으로 지정할 수 있습니다.

포맷:

```
<column_definition> ::=  
column_name <data_type>  
    [ FILESTREAM ] [ COLLATE collation_name ] [ SPARSE ][ NULL | NOT NULL ]  
    [ [ CONSTRAINT constraint_name ] DEFAULT constant_expression ]  
    | [ IDENTITY [ ( seed ,increment ) ] [ NOT FOR REPLICATION ] ]  
    [ ROWGUIDCOL ]
```

예:

```
CID int identity(1,1) primary key,  
CNAME varchar(50) not null,  
PHONE varchar(50) not null,  
ADDR varchar(100)
```

열 정의 예에서는 CID, CNAME, PHONE, ADDR 이름의 열을 정의한 것입니다. int, varchar(50), varchar(100)은 형식을 나타내는 것입니다. 그리고 identity는 테이블의 특정 열의 값을 데이터 베이스 엔진에서 자동으로 값을 지정하기 위한 것이며 (1,1)은 테이블에 처음 추가하는 데이터의 값을 1로 시작하고 1씩 증가하라고 지정한 것입니다. 일반적으로 identity로 지정한 열은 PRIMARY KEY 제약 조건을 지정합니다. 보다 자세한 내용을 원하시면 쿼리문을 자세히 다루는 레퍼런스나 MSDN을 참고하시기 바랍니다.

## 6.5.2 ALTER TABLE

ALTER문은 기존의 엔터티 정의를 수정할 때 사용합니다. 여기서는 테이블 정의를 수정하는 ALTER TABLE을 살펴볼게요. ALTER TABLE 문을 이용하면 테이블의 열과 제약 조건 등을 추가, 삭제, 변경 등을 할 수 있습니다. 이 책에서는 간단하게 기존 테이블에 새로운 열을 추가하고 삭제 및 변경하는 쿼리문을 살펴볼게요.

▷ 열 변경

포맷:

```
ALTER TABLE [ database_name . [ schema_name ] . | schema_name . ] table_name  
{  
    ALTER COLUMN column_name  
    {  
        [ type_schema_name. ] type_name [ ( { precision [ , scale ] | max | xml_schema_collection } ) ]  
        [ COLLATE collation_name ] [ NULL | NOT NULL ] [ SPARSE ] |  
        {ADD | DROP } { ROWGUIDCOL | PERSISTED | NOT FOR REPLICATION | SPARSE }  
    }  
}[ ; ]
```

예:

```
ALTER TABLE Custom ALTER COLUMN ADDR varchar(200)
```

위 예는 Custom 테이블의 ADDR 열의 형식을 변경하는 쿼리문입니다.

▷ 열 추가

포맷:

```
ALTER TABLE [ database_name . [ schema_name ] . | schema_name . ] table_name
{
    ADD
    {
        <column_definition>|<computed_column_definition>|<table_constraint>|<column_set_definition>
    }
}[ ; ]
```

예:

```
ALTER TABLE Custom ADD Description varchar(100)
```

위 예는 Custom 테이블에 Description 열을 추가하는 쿼리문입니다.

▷ 열 삭제

포맷:

```
ALTER TABLE [ database_name . [ schema_name ] . | schema_name . ] table_name
{
    DROP
    {
        [ CONSTRAINT ]
        {
            constraint_name
            [ WITH ( <drop_clustered_constraint_option> [ ,...n ] ) ]
        } [ ,...n ]
        | COLUMN
        {
            column_name
        } [ ,...n ]
    } [ ,...n ]
}[ ; ]
```

예:

```
ALTER TABLE Custom DROP COLUMN Description
```

위 예는 Custom 테이블의 Description 열을 삭제하는 쿼리문입니다.

### 6.5.3 SP\_RENAME

MS SQL에서는 테이블의 이름을 변경할 때 SP\_RENAME을 사용합니다. 데이터 베이스를 다루는 곳에서 데이터 정의로 RENAME문을 사용한다고 설명하는데 MS SQL에서는 SP\_RENAME을 사용합니다.

포맷:

```
sp_rename [ @objname = ] 'object_name' , [ @newname = ] 'new_name' [ , [ @objtype = ] 'object_type' ]
```

예:

```
SP_RENAME Custom, Custom2
```

위 예는 Custom 테이블 이름을 Custom2로 바꾸는 쿼리문입니다.

### 6.5.4 TRUNCATE TABLE

TRUNCATE TABLE문은 테이블의 모든 행을 삭제할 때 사용하는 쿼리문입니다. DELETE 문을 이용하여 모든 행을 삭제하는 것과 비슷합니다. 차이가 있다면 TRUNCATE TABLE문은 ID 사양의 열의 값을 초기화한다는 것입니다. 이 외에도 TRUNCATE TABLE문은 DELETE문을 이용하는 것보다 속도도 빠르고 작업한 트랜잭션 로그의 양도 작습니다.

포맷:

```
TRUNCATE TABLE
```

```
[ { database_name . [ schema_name ] . | schema_name . } ] table_name
```

```
[ ; ]
```

예:

```
TRUNCATE TABLE SALE
```

위 예는 SALE 테이블의 모든 행을 삭제하는 쿼리문입니다. 쿼리문을 수행하면 모든 행을 삭제하고 ID 사양의 열의 값을 초기화합니다.

### 6.5.5 DROP TABLE

DROP문을 사용하면 기존 엔티티를 제거할 수 있습니다. 만약 기존 테이블을 삭제하려면 DROP TABLE문을 사용하세요.

포맷:

```
DROP TABLE [ database_name . [ schema_name ] . | schema_name . ]
```

```
table_name [ ,...n ] [ ; ]
```

예:

```
DROP TABLE SALE
```

위 예는 SALE 테이블을 삭제하는 쿼리문입니다.

### 6.5.6 INSERT

테이블에 새로운 행을 추가할 때는 INSERT문을 사용합니다.

포맷:

[ WITH <common\_table\_expression> [ ,...n ] ]

INSERT

```
{
    [ TOP ( expression ) [ PERCENT ] ] [ INTO ]
    { <object> | rowset_function_limited [ WITH ( <Table_Hint_Limited> [ ...n ] ) ] }
    {
        [ ( column_list ) ] [ <OUTPUT Clause> ]
        {
            VALUES ( { DEFAULT | NULL | expression } [ ,...n ] ) [ ,...n ]
            | derived_table | execute_statement | <dml_table_source> | DEFAULT VALUES
        }
    }
}
```

[;]

예:

INSERT INTO Custom

(CNAME, PHONE, ADDR)

VALUES ('강감찬', '010-0000-0000', '서울시 종로구 종로1동 1번지')

### 6.5.7 DELETE

테이블의 행을 삭제할 때는 DELETE문을 사용합니다.

포맷:

[ WITH <common\_table\_expression> [ ,...n ] ]

DELETE

```
[ TOP ( expression ) [ PERCENT ] ] [ FROM ]
{
    { table_alias | <object> | rowset_function_limited [ WITH ( table_hint_limited [ ...n ] ) ] }
    | @table_variable
}
[ <OUTPUT Clause> ] [ FROM table_source [ ,...n ] ]
[ WHERE
    {
        <search_condition> | { [ CURRENT OF { [ GLOBAL ] cursor_name } | cursor_variable_name ] }
    }
]
[ OPTION ( <Query Hint> [ ,...n ] ) ]
[;]
```

예:

DELETE FROM Custom WHERE (CNAME = '강감찬')

## 6.5.8 UPDATE

UPDATE문은 테이블의 행의 내용을 변경할 때 사용하는 쿼리문입니다.

포맷:

[ WITH <common\_table\_expression> [...n] ]

UPDATE

[ TOP ( expression ) [ PERCENT ] ]

{ { table\_alias | <object> | rowset\_function\_limited

[ WITH ( <Table\_Hint\_Limited> [ ...n ] ) ]

}

| @table\_variable

}

SET

{ column\_name = { expression | DEFAULT | NULL }

| { udt\_column\_name.{ { property\_name = expression

| field\_name = expression }

| method\_name ( argument [ ,...n ] )

}

}

| column\_name { .WRITE ( expression , @Offset , @Length ) }

| @variable = expression

| @variable = column = expression

| column\_name { += | -= | \*= | /= | %= | &= | ^= | |= } expression

| @variable { += | -= | \*= | /= | %= | &= | ^= | |= } expression

| @variable = column { += | -= | \*= | /= | %= | &= | ^= | |= } expression

} [ ,...n ]

[ <OUTPUT Clause> ]

[ FROM{ <table\_source> } [ ,...n ] ]

[ WHERE { <search\_condition>

| { [ CURRENT OF

{ { [ GLOBAL ] cursor\_name }

| cursor\_variable\_name }

]

}

}

]

[ OPTION ( <query\_hint> [ ,...n ] ) ]

[ ; ]

예:

UPDATE Custom SET CNAME = '홍길동' WHERE (CID = 3)

위 예는 Custom 테이블에서 CID가 3인 데이터의 CNAME 값을 홍길동으로 변경하는 쿼리문입니다.



### 6.5.9 SELECT

SELECT문은 데이터 베이스에서 원하는 데이터를 검색할 때 사용하는 쿼리문입니다.

포맷:

```
[ WITH <common_table_expression> ]  
SELECT select_list [ INTO new_table ]  
[ FROM table_source ] [ WHERE search_condition ] [ GROUP BY group_by_expression ]  
[ HAVING search_condition ] [ ORDER BY order_expression [ ASC | DESC ] ]
```

예:

```
SELECT CNAME, PHONE FROM Custom WHERE (CID = 3)
```

위 예는 Custom 테이블에서 CID가 3인 행의 CNAME과 PHONE 열의 값을 질의하는 쿼리문입니다.

## 6. 6 제공 함수

### 6.6.1 수치 연산 함수

ABS(수식) : 절대값

ACOS(실수 표현, -1에서 1사이) : 아크 코사인(코사인의 역함수)

ASIN(실수 표현, -1에서 1사이) : 아크 사인(사인의 역함수)

ATAN(실수 표현) : 아크 사인(사인의 역함수)

ATN2(실수 표현, 실수 표현): 원점에서 입력 인자의 점(y,x)까지의 선이 X축과의 각도

CEILING(수식): 올림

COS(실수 표현): 코사인

COT(실수 표현): 코탄젠트

DEGREES(수식): 라디안 각도를 도 단위로 변환

EXP(실수 표현): 지수

FLOOR(수식): 내림

LOG(실수[, 밑수]) :로그 , 밑수가 없으면 자연 로그

LOG10(실수): 상용 로그

PI() : PI의 상수값

POWER(실수,y): 거듭 제곱

RADIANS(수식): 도 단위를 라디안 각도로 변환

RAND([seed]):0에서 1사이의 난수

ROUND(수식,자리[,함수]):반올림

SIGN(수식): 부호(양수:1, 음수:-1, 0:0)

SIN(실수 표현): 사인

SQRT(실수 표현): 제곱근

SQUARE(실수 표현): 제곱

TAN(실수 표현): 탄젠트

### 6.6.1 날짜 및 시간 데이터 함수

다음은 MS SQL 서버에서 제공하는 날짜 및 시간 데이터에 관한 형식입니다.

형식이름	표현	정확도
time	hh:mm:ss[.nnnnnnnn]	100나노초
date	YYYY-MM-DD	1일
smalldatetime	YYYY-MM-DD hh:mm:ss	1초
datetime	YYYY-MM-DD hh:mm:ss[.nnn]	0.00333초
datetime2	YYYY-MM-DD hh:mm:ss[.nnnnnnnn]	100나노초
datetimeoffset	YYYY-MM-DD hh:mm:ss[.nnnnnnnn][+ -]hh:mm	100나노초

[표 6.1] 날짜 및 시간 관련 형식

## 6. 7 서버 탐색기를 이용하여 SQL 쿼리 사용하기

이번에는 서버 탐색기를 이용하여 SQL 쿼리를 사용하는 방법을 알아보시다.

### 6.7.1 데이터 추가

먼저 테이블에 데이터를 추가하는 SQL 쿼리문을 사용해 봅시다.

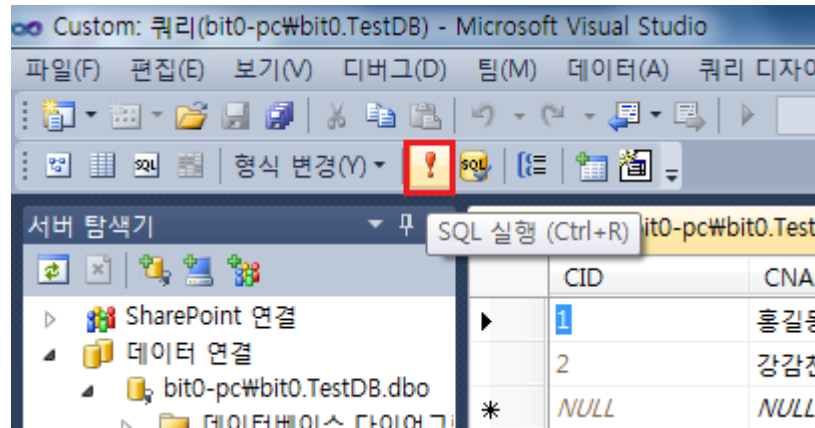
테이블에 데이터를 추가할 때는 INSERT 쿼리문을 사용합니다.

```
INSERT INTO [테이블 이름]
      ([컬럼명], [컬럼명],...)
VALUES ([데이터],[데이터],...)
```

테이블의 컨텍스트 메뉴를 이용하여 새 쿼리를 추가합니다. 여기서는 고객 테이블에 고객 데이터를 추가하는 예를 보여드릴게요.

```
INSERT INTO Custom
      (CNAME, PHONE, ADDR)
VALUES ('홍길동', '010-1111-1111', '서울특별시 종로구 1가 1번지')
```

쿼리를 작성하였으면 컨텍스트 메뉴를 이용하여 쿼리를 실행하세요. 쿼리를 실행하면 ‘최종 쿼리로 인해 1행이 영향을 받았습니다.’라는 메시지 창이 뜹니다. 고객 테이블의 데이터를 확인하면 추가한 데이터를 확인할 수 있을 것입니다. 만약 테이블 데이터 표시 창에 추가한 항목을 확인할 수 없다면 툴바의 SQL 실행 버튼을 눌러보세요.



[그림 6.14] 툴바의 SQL 실행 버튼

참고로 INSERT 쿼리문에서 모든 열의 정보를 명시하여 추가할 때는 테이블 이름 뒤에 컬럼 항목을 나열할 필요가 없습니다.

```
INSERT INTO [테이블 이름] VALUES ([데이터],[데이터],...)
```

### 6.7.2 데이터 삭제

테이블에 항목을 삭제할 때는 DELETE 쿼리문을 사용합니다.

```
DELETE FROM [테이블 이름]
```

```
WHERE [조건]
```

다음은 고객 이름이 홍길동인 항목을 삭제하는 쿼리문입니다.

```
DELETE FROM Custom
```

```
WHERE (CNAME = '홍길동')
```

다음은 고객 테이블에서 CID가 4보다 큰 항목을 삭제하는 쿼리문입니다.

```
DELETE FROM Custom
```

```
WHERE (CID > 4)
```

### 6.7.3 데이터 변경

테이블에 원하는 항목을 변경할 때는 UPDATE 쿼리문을 사용합니다.

```
UPDATE [테이블 이름]
```

```
SET [항목 명] = [항목 값], [항목 명] = 항목 값],...
```

```
WHERE [조건]
```

다음은 고객 테이블에서 CID가 3인 데이터의 CNAME과 PHONE을 변경하는 쿼리문입니다.

```
UPDATE Custom
```

```
SET CNAME = '광개토대왕', PHONE='010-1111-2000'
```

```
WHERE (CID = 3)
```

UPDATE 쿼리문도 영향받은 행의 개수를 표시해 줍니다.

#### 6.7.4 데이터 검색

원하는 조건의 데이터를 검색할 때는 SELECT 쿼리문을 사용합니다.

```
SELECT [항목 명], [항목 명],...  
FROM   [테이블 이름]  
WHERE  [조건]
```

모든 항목을 원할 때는 항목 리스트 대신 \*을 명시하고 조건이 필요없을 때는 WHERE 이하 구문은 생략할 수도 있습니다.

다음은 고객 테이블에서 CID가 2보다 크거나 같고 4보다 작은 데이터의 CNAME 항목과 PHONE 항목을 검색하는 쿼리문입니다.

```
SELECT CNAME, PHONE  
FROM   Custom  
WHERE  (CID >= 2) AND (CID < 4)
```

SELECT 쿼리문은 검색 결과 데이터를 표시합니다.

이번에는 여러 개의 테이블 사이에 관계가 있을 때 검색하는 쿼리를 작성해 봅시다. 이를 위해 먼저 판매 테이블에 데이터를 추가하기 위한 쿼리를 작성하여 실행하세요.

```
INSERT INTO Sale  
          (CID, PID, COUNT, SaleDate)  
VALUES (3, 1, 10, { fn NOW() })
```

여러 테이블 사이에 관계가 있을 때 검색할 때는 JOIN 쿼리문을 포함하여 사용하는 경우가 많이 있습니다. 여기에서는 여러가지 JOIN문 중에서 두 개의 테이블의 특정 항목이 같을 때 사용하는 INNER JOIN을 사용하는 예를 보여드릴게요. 다음은 고객 테이블의 CID와 판매 테이블이 같은 데이터와 판매 테이블의 PID와 상품 테이블의 PID가 같은 데이터를 JOIN하여 가격, 개수, 상품 이름, 고객 이름을 검색하는 쿼리문입니다.

```
SELECT Product.Price, Sale.COUNT, Product.PNAME, Custom.CNAME  
FROM   Custom INNER JOIN  
          Sale ON Custom.CID = Sale.CID INNER JOIN  
          Product ON Sale.PID = Product.PID
```

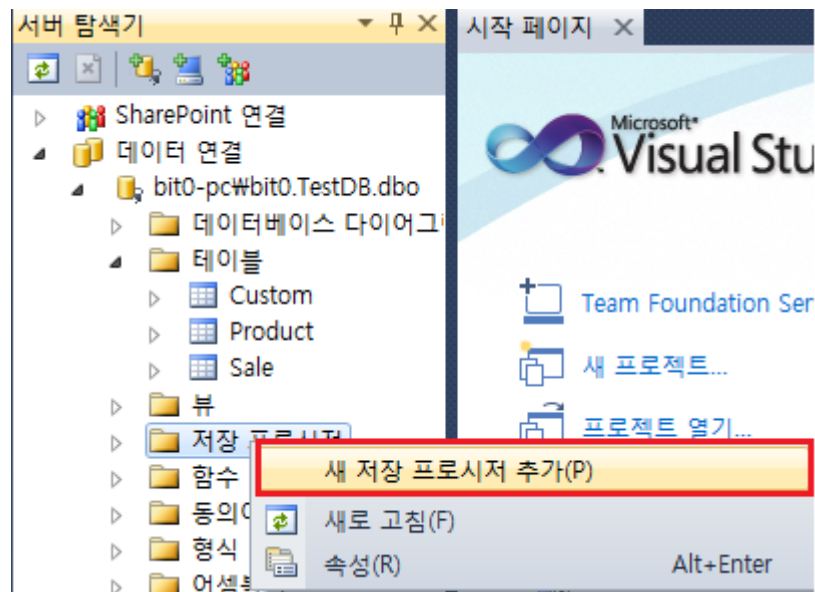
SQL 쿼리문은 이러한 기본 쿼리문들을 이용하여 보다 복잡한 쿼리문을 작성할 수 있는데 이 책에서는 이들에 대해 별도로 다루지 않습니다.

## 6. 8 서버 탐색기를 이용하여 SQL 저장 프로시저 사용하기

MS SQL 2008에서는 입력 인자를 포함하여 자주 사용하는 쿼리문을 저장하였다가 사용할 수 있게 저장 프로시저를 제공하고 있습니다.

### 6.8.1 상품 추가 저장 프로시저

저장 프로시저를 추가하려면 먼저 서버 탐색기의 저장 프로시저의 컨텍스트 메뉴에서 새 저장 프로시저 추가를 선택하세요.



[그림 6.15] 새 저장 프로시저 추가 선택

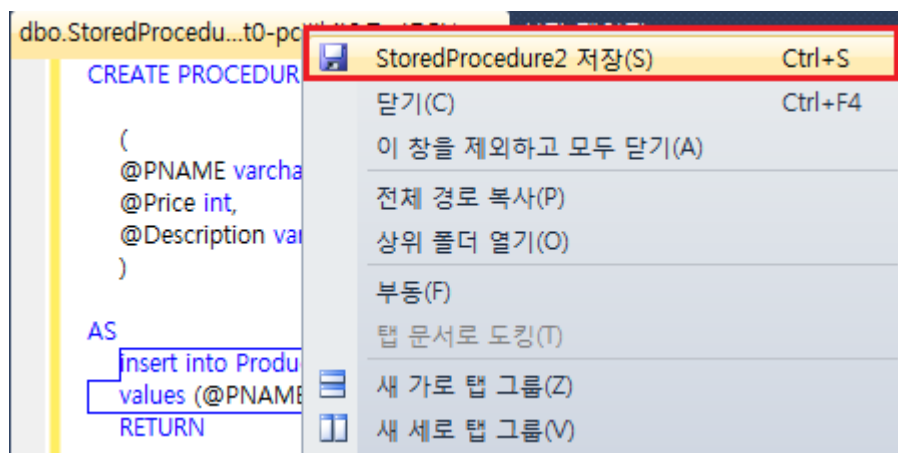
저장 프로시저는 저장 프로시저 이름과 매개 변수와 내부 변수, 수행 쿼리문 등을 포함할 수 있습니다.

```
CREATE PROCEDURE dbo.[저장 프로시저 이름]
(
    @[인자명] [인자형식],
    @[인자명] [인자형식],
    ...
)
AS
declare @[변수명] [변수 형식]
    쿼리문
RETURN
```

상품을 추가하려면 매개 변수로 상품 이름, 가격, 상세 정보가 필요합니다. 이를 저장 프로시저의 매개 변수로 명시한 후에 insert 쿼리문을 이용하여 상품 추가 저장 프로시저를 작성합니다.

```
CREATE PROCEDURE dbo.AddProduct
(
    @PNAME varchar(50),
    @Price int,
    @Description varchar(MAX)
)
AS
insert into Product (PNAME, Price, Description)
values (@PNAME, @Price, @Description)
RETURN
```

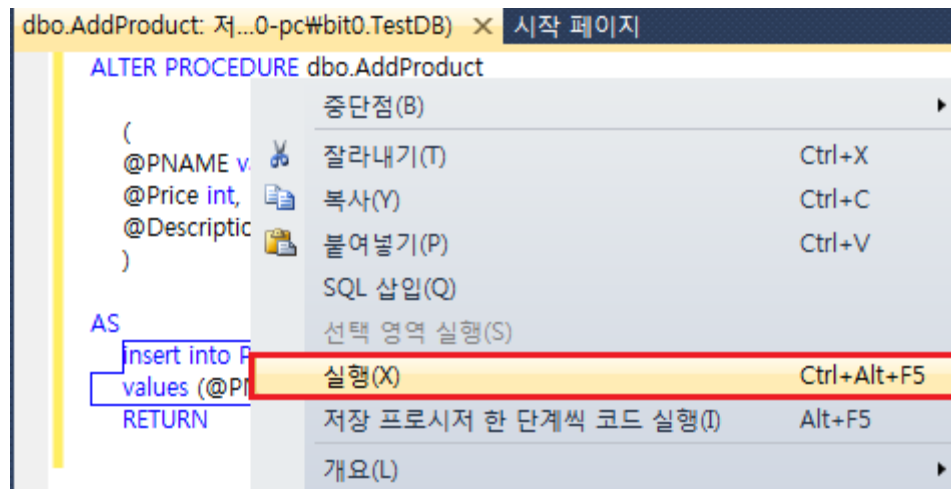
저장 프로시저를 작성하였으면 컨텍스트 메뉴를 이용하여 저장하세요. 만약, 저장 프로시저의 구문에 문제가 없으면 저장되고 CREATE 부분이 ALTER로 바뀝니다.



[그림 6.16] 저장 프로시저 저장

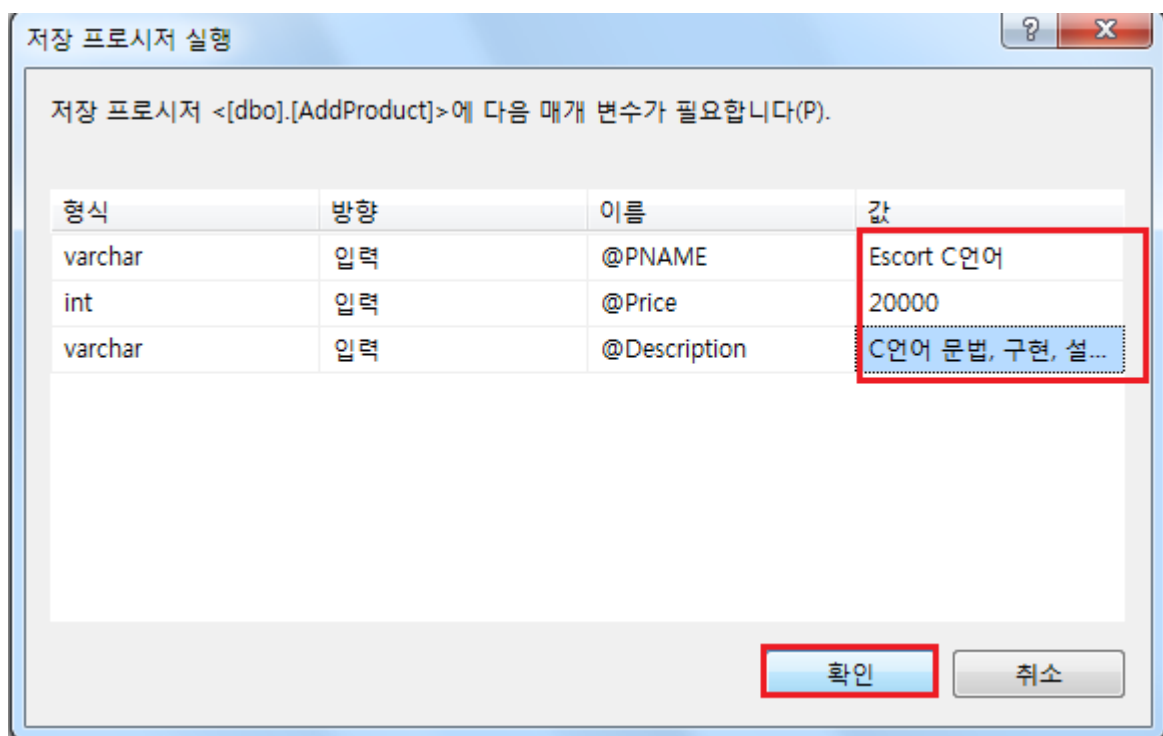
```
ALTER PROCEDURE dbo.AddProduct
(
    @PNAME varchar(50),
    @Price int,
    @Description varchar(MAX)
)
AS
insert into Product (PNAME, Price, Description)
values (@PNAME, @Price, @Description)
RETURN
```

이처럼 작성한 저장 프로시저는 컨텍스트 메뉴를 이용해 실행할 수 있습니다.



[그림 6.17] 저장 프로시저 실행

만약 저장 프로시저에 매개 변수가 있다면 저장 프로시저 실행 창에 인자를 설정하세요.



[그림 6.18] 매개 변수 설정

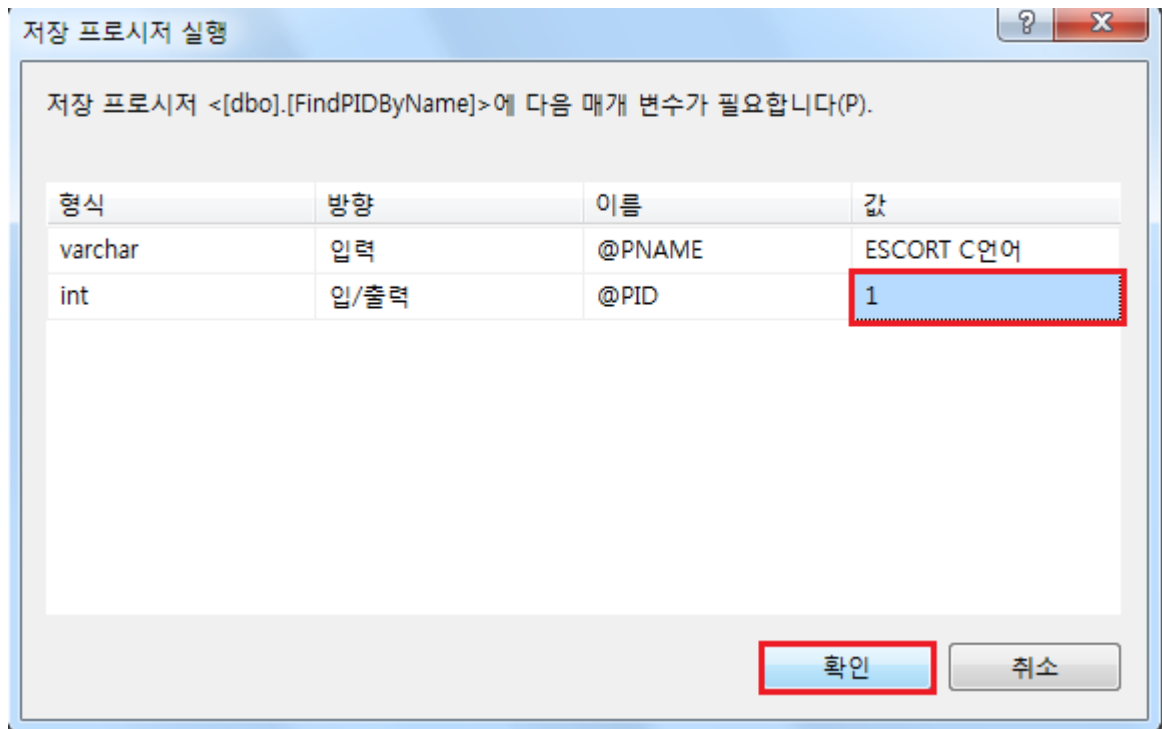
상품 저장 프로시저는 INSERT 쿼리문을 실행하므로 출력 창에 영향받은 행의 개수가 나타납니다. 여러분은 실행 후에 상품 테이블을 새로 고침하여 추가한 결과를 확인해 보세요.

## 6.8.2 상품 이름으로 PID 검색 저장 프로시저

이번에는 상품 이름으로 PID를 검색하는 저장 프로시저를 작성해 봅시다. 여기에서는 상품 이름을 인자로 받아서 검색한 후에 PID를 알려주어야 합니다. PID처럼 수행 결과를 알려 주려면 OUTPUT 유형의 매개 변수로 지정하세요. 참고로 여기에서는 검색한 상품이 없을 때 PID 값을 -1로 지정하는 것으로 설정하였습니다.

```
ALTER PROCEDURE dbo.FindPIDByName
(
    @PNAME varchar(50),
    @PID int OUTPUT
)
AS
    SET @PID = -1
    Select @PID = PID from Product
    where PNAME = @PNAME
    RETURN
```

그리고 OUTPUT 유형의 매개 변수가 있는 저장 프로시저를 실행할 때는 저장 프로시저 실행 창에서 임의의 값을 설정하여 실행해야 출력 창에 OUTPUT 유형의 매개 변수의 변화가 나타납니다.



[그림 6.19] OUTPUT 유형의 매개 변수의 값도 설정



### 6.8.3 고객 이름으로 CID 검색 저장 프로시저

```
ALTER PROCEDURE dbo.FindCIDByName
(
    @CNAME varchar(50),
    @CID int OUTPUT
)
AS
    SET @CID = -1
    Select @CID = CID from Custom
    where CNAME = @CNAME
    RETURN
```

### 6.8.4 상품 이름으로 판매 개수 확인 저장 프로시저

이번에는 상품 이름으로 판매 개수를 확인하는 저장 프로시저를 작성해 봅시다.

특정 상품의 판매 개수는 판매 테이블에 데이터를 저장하였습니다. 그런데 판매 테이블에는 상품 이름이 없기 때문에 먼저 상품 이름으로 PID를 얻어와야 판매 테이블에서 판매 개수를 확인할 수 있습니다. 앞에서 상품 이름으로 PID를 검색하는 저장 프로시저를 작성하였는데 여기에서는 이를 이용하는 저장 프로시저를 만들어서 사용할 것입니다.

저장 프로시저에서 다른 저장 프로시저를 호출할 때는 EXEC 구문을 이용합니다.

**EXEC** [저장 프로시저 이름] [인자 리스트]

EXEC 구문에 인자 리스트를 열거할 때는 출력 유형의 매개 변수는 OUTPUT 키워드를 명시해야 합니다. 그리고 저장 프로시저 내부에서 변수를 선언할 때는 DECLARE 구문을 이용하여 변수명과 형식을 명시합니다.

```
ALTER PROCEDURE dbo.GetCountByPname
(
    @PNAME varchar(50),
    @COUNT int OUTPUT
)
AS
    DECLARE @PID int
    EXEC FindPIDByName @PNAME, @PID output
    SELECT @COUNT =SUM(COUNT) FROM SALE where PID = @PID
    RETURN
```

### 6.8.5 상품 이름으로 상품 제거 저장 프로시저

이번에는 상품 이름으로 상품을 제거하는 저장 프로시저를 만들어 봅시다. 상품 테이블의 PID는 판매 테이블의 PID와 관계가 있기 때문에 상품 테이블에 상품 데이터를 제거하기 위해서는 판매 테이블에서 해당 상품의 PID인 판매 데이터를 먼저 제거해야 합니다. 이에 PID를 인자로 받아 판매 데이터를 제거하는 저장 프로시저를 먼저 작성합니다.

```
ALTER PROCEDURE dbo.RemoveSaleByPID
(
    @PID int
)
AS
    delete from Sale where PID = @PID
RETURN
```

상품 제거 저장 프로시저는 입력 인자로 상품 이름을 받는데 판매 테이블에서 판매 데이터를 제거하기 위해서는 PID를 알아야 하므로 앞에서 만든 상품 이름으로 PID를 구하는 FindPIDByName 저장 프로시저를 사용해야겠죠.

```
ALTER PROCEDURE dbo.RemoveProduct
(
    @PNAME varchar(50),
    @Result int OUTPUT
)
AS
declare @PID int
    Exec FindPIDByName      @PNAME,@PID OUTPUT
    if @PID = -1
    begin
        set @Result = 0
    end
    else
    begin
        Exec RemoveSaleByPID @PID
        delete from Product where PID = @PID
        set @Result = 1
    end
RETURN
```

### 6.8.6 고객 이름으로 고객 정보 제거 저장 프로시저

고객 이름으로 고객 정보를 제거하는 저장 프로시저도 고객 테이블의 고객 정보를 제거하기 전에 판매 테이블에 있는 해당 고객의 CID인 판매 정보를 제거해야 합니다.

```
ALTER PROCEDURE dbo.RemoveSaleByCID
(
    @CID int
)
AS
    delete from Sale where CID = @CID
RETURN
```

```
ALTER PROCEDURE dbo.RemoveCustom
(
    @CNAME varchar(50),
    @Result int OUTPUT
)
AS
declare @CID int
    Exec FindCIDByName      @CNAME,@CID OUTPUT
    if @CID = -1
    begin
        set @Result = 0
    end
    else
    begin
        Exec RemoveSaleByCID @CID
        delete from Custom where CID = @CID
        set @Result = 1
    end
RETURN
```

### 6.8.7 판매 데이터 추가 저장 프로시저

이번에는 판매 데이터를 추가하는 저장 프로시저를 작성해 봅시다. 판매 데이터는 PID, CID, COUNT 정보가 있어야 합니다. 그리고 이미 판매 테이블에 PID와 CID가 같은 판매 데이터가 있다면 COUNT를 추가하면 되겠죠. 따라서 먼저 판매 테이블에 PID와 CID가 같은 판매 데이터가 있는지 확인이 필요합니다.

```
Exists (Select * from SALE where PID = @PID and CID = @CID )
```

그리고 이미 PID와 CID가 같은 판매 데이터가 있다면 COUNT 정보를 얻어서 입력 인자로 받은 COUNT 값과 더한 값으로 변경해야 합니다. 여기에서는 현재 판매 데이터의 COUNT 값을 NOWCOUNT 변수에 얻어 올게요.

```
DECLARE @NOWCOUNT int
```

```
SELECT @NOWCOUNT = COUNT from SALE where PID = @PID and CID=@CID
```

그리고 입력 인자로 전달받은 COUNT와 NOWCOUNT의 값을 더하여 기존 판매 데이터의 COUNT 값을 변경해야겠죠.

```
SET @COUNT = @COUNT + @NOWCOUNT
```

```
UPDATE Sale SET COUNT = @COUNT where PID = @PID and CID=@CID
```

물론, 판매 테이블에 PID와 CID가 같은 판매 데이터가 없다면 판매 테이블에 추가하면 되겠죠. 그런데 판매 테이블에 판매 데이터를 추가하려면 판매 일기도 필요한데 이는 MS SQL 내장 함수인 now를 이용하기로 할게요.

```
insert into sale values(@CID, @PID, @Count, {fn now()})
```

```
ALTER PROCEDURE dbo.AddSale
(
    @PID int,
    @CID int,
    @COUNT int OUTPUT
)
AS
IF Exists (Select * from SALE where PID = @PID and CID = @CID )
Begin
    DECLARE @NOWCOUNT int
    SELECT @NOWCOUNT = COUNT from SALE where PID = @PID and CID=@CID
    SET @COUNT = @COUNT + @NOWCOUNT
    UPDATE Sale SET COUNT = @COUNT where PID = @PID and CID=@CID
End
```

```

else

    Begin

    insert into sale values(@CID, @PID, @Count, {fn now()})

    End

RETURN

```

이번에는 상품 이름과 고객 이름, 판매 개수를 인자로 하는 판매 추가 저장 프로시저를 작성해 봅시다. 이미 앞에서 상품 아이디와 고객 아이디로 판매를 추가하는 저장 프로시저는 AddSale 저장 프로시저로 만들었죠. 여기에서는 상품 이름으로 상품 아이디를 얻어오고 고객 이름으로 고객 아이디를 얻어와서 AddSale 저장 프로시저를 사용합니다.

```

ALTER PROCEDURE dbo.AddSale2

(
    @PNAME varchar(50),
    @CNAME varchar(50),
    @COUNT int OUTPUT
)

AS

declare @PID int
declare @CID int
Exec FindCIDByName @CNAME,@CID output
IF @CID = -1
    BEGIN
        set @COUNT = -1
        RETURN
    END
Exec FindPIDByName @PNAME,@PID output
IF @PID = -1
    BEGIN
        set @COUNT = -1
        RETURN
    END
Exec AddSale @PID, @CID, @COUNT output
RETURN

```

## 7. SqlConnection

이 책에서는 데이터 소스를 SQL Server 환경에서 사용하는 예로 설명할 것입니다. 하지만 ADO.NET 기술은 데이터 소스의 종류에 상관없이 일관된 방법으로 사용할 수 있게 하고 있어서 다른 데이터 소스를 사용하기 위해 새롭게 학습할 필요는 없습니다. 그리고 이 책에서는 .NET Framework 4를 기준으로 서술하고 있으니 참고하시기 바랍니다.

SqlConnection 클래스는 SQL Server 데이터 소스와의 연결을 제공하는 클래스입니다.

▷ 클래스 상속 계층

System.Object

System.MarshalByRefObject

System.ComponentModel.Component

System.Data.Common.DbConnection

System.Data.SqlClient.SqlConnection

▷ 네임 스페이스 : System.Data.SqlCleit

▷ 어셈블리: System.Data.dll

SqlConnection 개체는 SQL 데이터 소스와의 작업을 수행하기 위해 필요한 연결을 여는 작업을 수행합니다. 연결을 열기 위해 연결 대상인 데이터 소스와 계정 정보 등을 지정해야 합니다.

```
static void Main(string[] args)
{
    string constr = @"Data Source=[서버 이름];Initial Catalog=[DB 명]; User ID=[ID];Password=[PW]";
    SqlConnection scon = new SqlConnection(constr);
    scon.Open(); //연결 열기
    //작업 수행
    Console.WriteLine(scon.WorkstationId);
    Console.WriteLine(scon.ServerVersion);
    Console.WriteLine(scon.PacketSize);
    Console.WriteLine(scon.ConnectionTimeout);
    Console.WriteLine(scon.Database);
    Console.WriteLine(scon.DataSource);
    Console.WriteLine(scon.State);
    scon.Close(); //연결 닫기
}
```

[소스 7.1] SqlConnection 소개

## 7. 1 SqlConnection 생성자

SqlConnection 클래스에는 두 가지 생성자를 제공합니다.

SqlConnection();

SqlConnection(string conStr);

▷ SqlConnection();

연결할 데이터 소스를 지정하지 않은 상태의 SqlConnection 개체를 생성합니다. 연결을 열려면 연결 문자열을 ConnectionString 속성에 지정하여야 합니다.

속성	초기값	설명
ConnectionString	string.Empty	연결 문자열
ConnectionTimeout	15(분)	연결 대기시간
Database	string.Empty	데이터 베이스 명
DataSource	string.Empty	SQL 서버 인스턴스 명

만약, 연결 문자열을 지정하지 않고 열기를 시도하면 InvalidOperationException이 발생하며 예외 메시지 속성 값은 "ConnectionString 속성이 초기화되지 않았습니다." 입니다.

```
static void Main(string[] args)
{
    try
    {
        string constr = @"Data Source=[서버 이름];Initial Catalog=[DB 명]; User ID=[ID];Password=[PW]";
        SqlConnection scon = new SqlConnection();
        scon.ConnectionString = constr;
        scon.Open(); //연결 열기
        //작업 수행
        scon.Close(); //연결 닫기
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
```

[소스 7.2] SqlConnection 생성자

▷ SqlConnection(string conStr);

연결 문자열을 인자로 전달하여 SqlConnection 개체를 생성합니다. 연결 문자열에는 데이터 소스의 이름과 데이터 베이스 이름, 계정 정보 등을 지정할 수 있습니다.

연결 문자열은 연결에 필요한 여러 종류의 속성을 약속한 키워드와 대입 연산자와 값을 세미콜론으로 구분하여 지정할 수 있습니다.

```
string constr = @"Data Source=My Com\SqlServer;Initial Catalog=TestDB; User ID=abc;Password=123";
```

다음은 연결 문자열에 사용할 수 있는 주요한 키워드입니다. 이 외에도 세부 설정을 위한 다양한 키워드를 제공하고 있는데 자세한 사항은 MSDN을 참고하세요.

키워드	초기값	설명
Addr, Address, Data Source, Server, Network Address	없음	데이터 소스(SQL 서버 인스턴스 명)
Initial Catalog, Database	없음	데이터 베이스 명
User ID, UID	없음	SQL 로그인 계정
Password, PWD	없음	SQL 로그인 계정의 비밀번호
Connect Timeout, Connection Timeout, Timeout	15(초)	연결 대기 시간(초)
Packet Size	8192(바이트)	SQL 서버와 통신 네트워크 패킷사이즈 512~32768

만약 연결 문자열에 사용할 수 없는 키워드가 있다면 ArgumentException이 발생합니다. 그리고 연결에 필요한 값이 잘못 지정하였을 때에는 Open 메서드를 호출할 때 예외가 발생합니다.



## 7. 2 SqlConnection 속성

SqlConnection 개체는 연결에 관한 여러 가지 속성과 연결 상태에 관한 속성을 제공합니다. 다음은 주요한 속성입니다.

속성	가져오기/설정하기	설명
ConnectionString	가져오기/설정하기	연결 문자열
Connection Timeout	가져오기/설정하기	연결 대기 시간
Database	가져오기	데이터 베이스 명
DataSource	가져오기	데이터 소스(SQL 서버 인스턴스 명)
PacketSize	가져오기	패킷 사이즈
ServerVersion	가져오기	SQL 서버 버전
State	가져오기	상태
WorkstationId	가져오기	데이터 베이스 클라이언트 식별 ID (지정하지 않았을 때는 컴퓨터 이름)

대부분의 속성은 연결 문자열로 지정한 값입니다. SqlConnection 속성 중에 State 속성은 연결 상태를 나타내는 속성으로 다음의 값 중에 하나를 갖습니다.

상태	설명
Closed	연결이 닫힌 상태
Open	연결이 열린 상태
Connecting	연결 열기를 시도하는 상태
Executing	명령을 실행하고 있는 상태
Fetching	데이터를 검색하고 있는 상태
Broken	연결이 열린 이후에 끊어진 상태(연결을 닫은 후에 연결하여 사용해야 함)

## 7. 3 SqlConnection 메서드

SqlConnection 개체는 데이터 소스와 연결을 열거나 닫기 위한 메서드를 제공합니다. 그리고 연결이 열린 상태에서 여러 개의 작업을 하나의 논리 작업으로 수행하기 위한 트랜잭션 개체를 생성하는 메서드와 데이터 소스를 변경하는 메서드 등을 제공합니다. 다음은 SqlConnection 클래스에서 제공하는 주요 메서드입니다.

메서드	설명
BeginTransaction	트랜잭션을 시작(트랜잭션 개체를 생성하여 반환함)
ChangeDatabase	열려있는 SqlConnection 개체의 데이터 베이스를 변경
Close	연결 열기
Open	연결 닫기

트랜잭션은 여러 개의 작업을 하나의 논리 작업으로 묶어 작업 중간에 초기 상태로 복귀(Rollback)하거나 현재까지의 작업을 완료(Commit)시키는 것을 말합니다. 이를 사용하는 것은 데이터 소스에 연결을 열고 원하는 작업을 수행하는 SqlCommand 개체를 사용할 때 여러 작업을 논리적으로 묶기 위한 것이므로 여기에서 사용 예를 보이지 않고 SqlCommand 설명 후에 얘기할게요.

ChangeDatabase 메서드는 SqlConnection 개체로 연결을 연 후에 데이터 베이스를 변경할 때 사용합니다.

```
static void Main(string[] args)
{
    try
    {
        string constr = @"Data Source=[서버 이름];Initial Catalog=[DB 명]; User ID=[ID];Password=[PW]";
        SqlConnection scon = new SqlConnection();
        scon.ConnectionString = constr;
        scon.Open(); //연결 열기
        Console.WriteLine("데이터 베이스:{0}", scon.Database);
        scon.ChangeDatabase("master");
        Console.WriteLine("데이터 베이스:{0}", scon.Database);
        scon.Close(); //연결 닫기
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
```

[소스 7.3] SqlConnection 클래스의 ChangeDatabase 메서드

## 8. SqlCommand

SqlConnection 개체로 데이터 소스에 연결한 후에 필요한 구체적인 작업을 할 때 SqlCommand 개체를 사용합니다.

SqlCommand 개체로 수행할 수 있는 작업은 쿼리문이나 저장 프로시저를 이용합니다. 그리고 인자를 사용하기 위해 SqlParameter 개체를 보관하는 컬렉션이 있으며 수행할 작업 종류에 따라 다양한 명령 방법을 제공하고 있습니다.

SqlCommand 개체를 생성한 후에 작업 내용에 해당하는 쿼리 문자열 혹은 저장 프로시저를 CommandText 속성에 설정하거나 생성할 때 입력 인자로 전달할 수 있습니다. 그리고 SqlCommand 개체를 이용하여 구체적인 작업을 수행하기 전에 연결이 열린 SqlConnection 개체를 지정해야 하는데 이 또한 생성자에 인자로 전달하거나 Connection 속성에 설정할 수 있습니다.

```
private static void ExSqlCommand(string constr, string comstr)
{
    SqlConnection scon = new SqlConnection(constr);
    SqlCommand scom = new SqlCommand();
    scom.CommandText = comstr;
    scom.Connection = scon;
    scon.Open();
    scom.ExecuteNonQuery();
    scon.Close();
}
```

▷ 클래스 상속 계층

System.Object

System.MarshalByRefObject

System.ComponentModel.Component

System.Data.Common.DbCommand

System.Data.SqlClient.SqlCommand

▷ 네임 스페이스 : System.Data.SqlCleint

▷ 어셈블리: System.Data.dll

## 8. 1 SqlCommand 생성자

SqlCommand 개체는 SqlConnection 개체의 CreateCommand 메서드를 이용하거나 SqlCommand 생성자 호출로 생성할 수 있습니다. 다음은 두 가지 방법을 사용한 예입니다.(예외처리 구문 생략하였음)

```
class Program
{
    static void Main(string[] args)
    {
        string constr = @"Data Source=[서버 이름];Initial Catalog=[DB 명]; User ID=[ID];Password=[PW]";
        string comtxt = "insert into Product (PNAME, Price, Description) values ('ADO.NET',20000,'생략')";
        ExCreateCommand(constr,comtxt);
        ExSqlCommandCreator(constr, comtxt);
    }
    private static void ExSqlCommandCreator(string constr,string comtxt)
    {
        SqlConnection scon = new SqlConnection(constr);
        SqlCommand scom = scon.CreateCommand();
        scom.CommandText = comtxt;
        ExecCommand(scon,scom);
    }
    private static void ExCreateCommand(string constr, string comtxt)
    {
        SqlConnection scon = new SqlConnection(constr);
        SqlCommand scom = new SqlCommand(comtxt, scon);
        ExecCommand(scon, scom);
    }
    private static void ExecCommand(SqlConnection scon,SqlCommand scom)
    {
        scon.Open();
        if (scom.ExecuteNonQuery() == 1)
        {
            Console.WriteLine("추가 성공");
        }
        scon.Close();
    }
}
```

SqlCommand 클래스에는 네 가지 생성자를 제공합니다.

```
SqlCommand();  
SqlCommand(string comText);  
SqlCommand(string comText, SqlConnection scon);  
SqlCommand(string comText, SqlConnection scon, SqlTransaction stran);
```

▷ SqlCommand();

입력인자를 전달하지 않고 SqlCommand 개체를 생성하였을 때는 별도의 구문으로 SqlConnection 개체와 실제 수행할 명령을 설정한 후에 명령을 실행시킬 수 있습니다.

속성	초기값	설명
CommandText	string.Empty	수행할 명령
CommandType	CommandType.Text	명령 종류
Connection	null	데이터 소스와 연결 SqlConnection 개체

▷ SqlCommand(string comText);

SqlCommand 개체를 생성할 때 실제 수행할 명령을 인자로 전달하여 생성할 수 있으며 이 때 명령 종류를 의미하는 CommandType 속성은 CommandType.Text입니다. CommandType.Text는 SQL 쿼리문을 의미하며 생성할 때 전달한 명령이 저장 프로시저 이름이라면 CommandType.StoredProcedure로 설정해야 합니다.

SqlCommand 개체를 이용하여 작업을 수행할 때 작업 형식으로 SQL 명령어 혹은 저장 프로시저 이름을 사용할 수 있습니다. OLE DB일 때는 TableDirect도 사용할 수 있지만 SqlCommand를 이용할 때는 사용할 수 없습니다. SqlCommand 개체의 디폴트 작업 형식은 SQL 명령어로 되어 있는데 만약 저장 프로시저를 사용하려면 CommandType 속성을 CommandType 열거형의 StoredProcedure로 설정하세요.

```
scom.CommandType = CommandType.StoredProcedure;
```

SqlCommand에 매개 변수를 설정하여 작업을 수행할 때는 SqlParameter 개체를 사용하고 작업의 종류에 따라 실행할 수 있는 다양한 메서드를 제공하고 있는데 이들은 SqlParameter와 SqlCommand의 실행 메서드들에서 다룰게요.

▷ SqlCommand(string comText, SqlConnection scon);

SqlCommand 개체를 이용하여 실제 명령을 수행하기 위해선 반드시 SqlConnection 개체가 설정되어 있어야 합니다. SqlCommand 클래스의 생성자 중에는 SqlConnection 개체를 입력 인자로 받는 생성자를 제공하고 있습니다.

▷ SqlCommand(string comText, SqlConnection scon, SqlTransaction stran);

DB 관련 명령을 수행할 때는 여러 개의 명령을 하나의 논리 작업으로 묶어서 작업 중에 문제가 있을 때 작업 전 상태로 복원할 수 있습니다. 이와 같이 처리하려면 여러 개의 명령을 하나의 논리 작업으로 묶는 개체가 필요한데 SqlTransaction으로 제공하고 있습니다.

## 8. 2 SqlCommand 속성

SqlCommand 개체는 명령에 관한 여러 가지 속성을 제공합니다. 다음은 주요한 속성입니다.

속성	가져오기/설정하기	설명
CommandText	가져오기/설정하기	실행할 SQL문이나 테이블 이름, 저장 프로시저 이름
CommandTimeout	가져오기/설정하기	명령 실행할 때 대기시간(초,기본값 30초)
CommandType	가져오기/설정하기	명령 종류
DbConnection	가져오기/설정하기	명령에 사용할 DbConnection 개체
Parameters	가져오기	SqlParameterCollection
Transaction	가져오기/설정하기	명령을 실행할 때 SqlTransaction 개체

CommandText 속성은 실제 수행할 명령에 관한 속성입니다. CommandText 속성 값은 CommandType 속성이 CommandType.Text일 때는 명령에 수행할 SQL문을 설정합니다. 그리고 CommandText.StoredProcedure일 때는 저장 프로시저 이름으로 설정합니다.

만약 명령에 매개 변수를 사용하고자 한다면 Parameters 속성에 SqlParameter 개체를 만들어 추가하세요.

```
SqlCommand command = new SqlCommand();  
command.CommandText = "SELECT * FROM Books ";  
command.CommandTimeout = 15;  
command.CommandType = CommandType.Text;
```


## 8. 3 SqlCommand Execute 메서드

SqlCommand 개체는 명령을 실행할 때 사용하는 개체입니다. SqlCommand 클래스는 다양한 형태로 실행할 수 있게 다양한 Execute메서드를 제공하고 있습니다.

### 8.3.1 ExecuteNonQuery 메서드

ExecuteNonQuery 메서드는 명령을 수행하고 영향을 받은 행의 수를 반환하는 메서드입니다. 행 추가나 변경, 삭제 등의 명령을 수행할 때는 명령으로 영향받은 행의 수만 알면 되기 때문에 ExecuteNonQuery 메서드를 사용합니다.

다음처럼 Books 테이블이 있을 때 SqlCommand 개체의 ExecuteNonQuery 메서드를 이용하여 도서를 추가하는 부분을 작성해 봅시다.

Books			
	열 이름	데이터 형식	Null 허용
	Title	varchar(50)	<input type="checkbox"/>
	Price	int	<input type="checkbox"/>
	Author	varchar(50)	<input type="checkbox"/>
	ISBN	varchar(50)	<input type="checkbox"/>
			<input type="checkbox"/>

[그림 8.1] Books 테이블 다이어그램

먼저 연결에 필요한 문자열이 필요합니다. 접근할 서버 이름과 DB 이름, 접근할 계정 ID와 비밀번호로 구성된 연결 문자열을 설정합니다.

```
string constr = @"Data Source=[서버 이름];Initial Catalog=[DB 명]; User ID=[ID];Password=[PW]";
```

명령에 필요한 SQL 문을 작성합니다. 여기에서는 하드 코딩하기로 할게요.

```
string comtext = "insert into Books values ('ADO.NET', 12000, '홍길동', '2984756325')";
```

SqlConnection 개체를 생성하고 SqlCommand 개체를 생성합니다.

```
SqlConnection scon = new SqlConnection(constr);
```

```
SqlCommand command = new SqlCommand(comtext, scon);
```

이제 연결을 열고 명령을 수행하면 됩니다. 물론 명령 작업이 끝났으면 연결을 닫아주세요. 결과를 알고자 한다면 ExecuteNonQuery 메서드 호출에서 결과를 반환받아 처리하면 됩니다.

```
scon.Open();  
command.ExecuteNonQuery();  
scon.Close();
```

```

static void Main(string[] args)
{
    string comtext = "insert into Books values ('ADO.NET', 12000, '홍길동', '2984756325')";
    string constr = @"Data Source=[서버 이름];Initial Catalog=[DB 명]; User ID=[ID];Password=[PW]";
    SqlConnection scon = new SqlConnection(constr);
    SqlCommand command = new SqlCommand(comtext, scon);
    scon.Open();
    if (command.ExecuteNonQuery() == 1)
    {
        Console.WriteLine("추가 성공");
    }
    else
    {
        Console.WriteLine("추가 실패");
    }
    scon.Close();
}

```

[소스 8.1] ExecuteNonQuery 메서드를 이용한 행 추가

### 8.3.2 ExecuteScalar 메서드

명령을 실행하고 수행한 결과 집합에서 1행 1열을 반환하는 ExecuteScalar 메서드를 제공하고 있습니다.



### 8.3.3 ExecuteReader 메서드

명령 수행하고 수행한 결과를 확인할 때 사용할 SqlDataReader 개체를 빌드하여 반환하는 ExecuteReader 메서드를 제공합니다.

Select 문처럼 명령 수행 결과가 집합일 때 사용하는 메서드입니다. 메서드를 호출 후에 결과를 확인하기 위해서 SqlDataReader 개체를 사용하며 사용이 끝나면 반드시 SqlDataReader 개체의 Close 메서드를 호출하여야 다른 명령을 수행할 수 있습니다.

```
static void Main(string[] args)
{
    string comtext = "Select * From Books";
    string constr = @"Data Source=[서버 이름];Initial Catalog=[DB 명]; User ID=[ID];Password=[PW]";
    SqlConnection scon = new SqlConnection(constr);
    SqlCommand command = new SqlCommand(comtext, scon);
    scon.Open();
    SqlDataReader reader = command.ExecuteReader();
    while (reader.Read())
    {
        Console.WriteLine("도서명: {0}", reader["Title"]);
        Console.WriteLine("ISBN: {0}", reader["ISBN"]);
        Console.WriteLine("저자: {0}", reader["Author"]);
        Console.WriteLine("가격: {0}", reader["Price"]);
    }
    reader.Close();
    scon.Close();
}
```

[소스 8.2] ExecuteReader를 이용한 검색

이 외에도 SqlCommand 개체에는 비동기 명령을 실행하기 위한 메서드와 비동기 명령을 종료하는 메서드들도 제공하고 있습니다.

## 9. SqlParameter

앞에서 SqlCommand 개체를 이용하여 명령을 수행하는 예를 보여주었는데 정적인 명령문의 형태만 보여주었습니다. 만약 명령을 수행할 때 매개 변수를 설정하여 작업을 수행하고자 한다면 SqlParameter 개체를 이용합니다.

SqlParameter 클래스는 명령 실행에 사용할 매개 변수를 설정할 때 사용하는 클래스입니다.

▷ 클래스 상속 계층

System.Object

System.MarshalByRefObject

System.Data.Common.DbParameter

System.Data.SqlClient.SqlParameter

### 9.1 SqlParameter 생성자

SqlParameter 클래스에는 6가지 생성자를 제공합니다.

```
SqlParameter ( );  
SqlParameter ( string par_name, SqlDbType dbtype );  
SqlParameter (string par_name, Object value );  
SqlParameter (string par_name, SqlDbType dbtype, int size );  
SqlParameter (string par_name, SqlDbType dbtype, int size, string column_name );  
SqlParameter ( string par_name, SqlDbType dbtype, int size, ParameterDirection direction, bool nullable,  
                byte precision, byte scale, string column_name, DataRowVersion version, Object value );  
SqlParameter ( string par_name, SqlDbType dbtype, int size, ParameterDirection direction, byte precision,  
                byte scale, string column_name, DataRowVersion version, Object value,  
                string xmlschema_collection_db, string xmlschema_collection_owning_schema,  
                string xmlschema_collection_name );
```

SqlParameter 개체는 매개 변수 이름, 값, 형식, 길이, 방향, 전체 자릿수, 소수 자릿수, 소수 열의 이름, 버전, 소스 열 매핑 여부, XML 개체의 스키마 컬렉션이 있는 DB 이름, XML 개체의 스키마 컬렉션이 소유하는 관계형 스키마, 스키마 컬렉션 이름 등을 입력 인자로 전달하여 개체를 생성할 수 있습니다.

SqlParameter 개체를 생성할 때 명확하게 결정한 사항은 전달하여 생성하고 나머지 사항은 디폴트 값이나 속성을 이용하여 결정할 수 있습니다.

SqlParameter 개체는 독립적으로 어떠한 작업을 수행하는 것이 아니라 SqlCommand 개체의 Execute 명령 수행에 필요한 매개 변수를 결정하는 것이기 때문에 Execute 명령 수행 전에 필요한 값을 설정하면 됩니다.

## 9. 2 SqlParameter 속성

SqlParameter 클래스에는 명령에 사용할 매개 변수에 관한 다양한 속성을 제공하고 있습니다. 다음은 주요 속성입니다.

속성	가져오기/설정하기	설명
Direction	가져오기/설정하기	매개 변수의 용도를 설정 (입력 전용, 출력 전용, 양방향, 반환 값)
IsNullable	가져오기/설정하기	null 허용 여부
ParameterName	가져오기/설정하기	매개 변수 이름
Precision	가져오기/설정하기	Value 속성이 숫자일 때 최대 자릿 수
Scale	가져오기/설정하기	Value를 확인하는 소수 자릿 수
Size	가져오기/설정하기	데이터의 최대 크기(바이트 단위)
SoureceColumn	가져오기/설정하기	DataSet에 매핑하여 Value를 로드하거나 반환하기 위해 사용한 소스 열의 이름
SqlDbType	가져오기/설정하기	SqlDbType
SqlValue	가져오기/설정하기	SQL 형식의 매개 변수의 값
Value	가져오기/설정하기	매개 변수의 값

[표 9.1] SqlParameter 속성

## 9. 3 SqlParameter 사용 예

이번에는 SqlParameter 를 사용하는 구체적인 예를 살펴봅시다.

SqlCommand 개체를 사용하는 예제 코드에서는 매개 변수를 사용하지 않아 정적인 쿼리문을 사용하는 예를 보여드렸습니다.

이번에는 매개 변수를 이용하는 예를 들어보기로 합시다.

추가할 책의 정보를 입력 인자로 받아 책을 추가하는 메서드를 만들어 사용하기로 합시다.

```
static void Main(string[] args)
{
    AddBook("XML.NET", 15000, "홍길동", "9224764583");
    AddBook("CSharp", 18000, "홍길동", "9228964583");
}
private static void AddBook(string title, int price, string author, string isbn)
{
    //to be defined
}
```

책을 추가하는 SQL문을 매개 변수를 이용하여 표현합니다.

```
string comtext = "insert into Books values (@Title, @Price,@Author,@ISBN)";
```

입력 인자로 받은 것을 값으로 하는 SqlParameter 개체를 생성하여 SqlCommand 개체에 추가하는 로직이 필요합니다.

다음은 매개 변수 이름과 값을 생성자에 전달하여 SqlParameter 개체를 생성한 예입니다.

```
SqlParameter param_title = new SqlParameter("@Title", title);
command.Parameters.Add(param_title);
```

다음은 SqlParameter 개체를 생성한 후에 매개 변수 이름, 타입, 값을 설정하는 예입니다.

```
SqlParameter param_price = new SqlParameter();
param_price.ParameterName = "@Price";
param_price.SqlDbType = System.Data.SqlDbType.Int;
param_price.Value = price;
command.Parameters.Add(param_price);
```

```

static void Main(string[] args)
{
    AddBook("XML.NET", 15000, "홍길동", "9224764583");
    AddBook("CSharp", 18000, "홍길동", "9228964583");
}

private static void AddBook(string title, int price, string author, string isbn)
{
    string comtext = "insert into Books values (@Title, @Price,@Author,@ISBN)";
    string constr = @"Data Source=[서버 이름];Initial Catalog=[DB 명]; User ID=[ID];Password=[PW]";
    SqlConnection scon = new SqlConnection(constr);
    SqlCommand command = new SqlCommand(comtext, scon);

    SqlParameter param_title = new SqlParameter("@Title", title);
    command.Parameters.Add(param_title);
    SqlParameter param_price = new SqlParameter();
    param_price.ParameterName = "@Price";
    param_price.SqlDbType = System.Data.SqlDbType.Int;
    param_price.Value = price;
    command.Parameters.Add(param_price);
    SqlParameter param_author = new SqlParameter("@Author", author);
    command.Parameters.Add(param_author);
    SqlParameter param_isbn = new SqlParameter("@ISBN", isbn);
    command.Parameters.Add(param_isbn);

    scon.Open();
    if (command.ExecuteNonQuery() == 1)
    {
        Console.WriteLine("{0} 추가 성공", title);
    }
    else
    {
        Console.WriteLine("{0} 추가 실패", title);
    }
    scon.Close();
}

```

[소스 9.1] SqlParameter를 이용한 도서 추가 예제 코드

이번에는 저장 프로시저를 이용하는 예를 들어보기로 할게요.

먼저 도서를 추가하는 저장 프로시저를 만듭니다.

```
ALTER PROCEDURE dbo.AddBook
(
    @ISBN varchar(50),
    @Title varchar(50),
    @Author varchar(50),
    @Price int,
    @Result int OUTPUT
)
AS
begin
    set @Result = 0
    select @Result = count(*) from Books where (ISBN=@ISBN)
    if @Result = 0
    begin
        insert Books values(@Title,@Price,@Author,@ISBN)
        set @Result = 1
    end
end
RETURN
```

[소스 9.2] AddBook 저장 프로시저

저장 프로시저를 실행하려면 SqlCommand의 CommandType 속성을 StoredProcedure로 설정해야 합니다.  
command.CommandType = System.Data.CommandType.StoredProcedure;

AddBook 저장 프로시저를 보면 @Result 인자는 인자 유형이 OUTPUT으로 되어 있기 때문에 파라미터의 방향을 Output으로 설정해야 합니다.

```
SqlParameter param_result = new SqlParameter();
param_result.Direction = System.Data.ParameterDirection.Output;
param_result.ParameterName = "@Result";
param_result.SqlDbType = System.Data.SqlDbType.Int;
command.Parameters.Add(param_result);
```

```

static void Main(string[] args)
{
    AddBook(".NET 플랫폼", 15000, "홍길동", "9224764583");
    AddBook("CSharp", 18000, "홍길동", "9228964583");
}

private static void AddBook(string title, int price, string author, string isbn)
{
    string comtext = "AddBook";
    string constr = @"Data Source=[서버 이름];Initial Catalog=[DB 명]; User ID=[ID];Password=[PW]";

    SqlConnection scon = new SqlConnection(constr);
    SqlCommand command = new SqlCommand(comtext, scon);
    command.CommandType = System.Data.CommandType.StoredProcedure;

    SqlParameter param_title = new SqlParameter("@Title", title);
    command.Parameters.Add(param_title);

    SqlParameter param_price = new SqlParameter();
    param_price.ParameterName = "@Price";
    param_price.SqlDbType = System.Data.SqlDbType.Int;
    param_price.Value = price;
    command.Parameters.Add(param_price);

    SqlParameter param_author = new SqlParameter("@Author", author);
    command.Parameters.Add(param_author);

    SqlParameter param_isbn = new SqlParameter("@ISBN", isbn);
    command.Parameters.Add(param_isbn);

    SqlParameter param_result = new SqlParameter();
    param_result.Direction = System.Data.ParameterDirection.Output;
    param_result.ParameterName = "@Result";
    param_result.SqlDbType = System.Data.SqlDbType.Int;
    command.Parameters.Add(param_result);
}

```

```
scon.Open();
command.ExecuteNonQuery();
int result = (int)param_result.Value;
if (result == 1)
{
    Console.WriteLine("{0} 도서추가 성공", title);
}
else
{
    Console.WriteLine("{0} 도서추가 실패", title);
}
scon.Close();
}
```

[소스 9.3] 저장 프로시저를 이용한 도서 추가 예제 코드



## 10. DataTable

ADO.NET에서는 논리적인 데이터 집합을 디자인하고 데이터를 관리할 수 있는 DataTable 클래스를 제공하고 있습니다. DataTable은 DataSet을 구성하는 주요 개체로 프로그램 메모리 상의 한 개의 테이블입니다.

여기에서는 DataTable 개체를 생성하여 테이블을 설계하고 해당 개체를 이용하여 데이터를 관리하는 방법을 살펴볼게요.

▷ 클래스 상속 계층

System.Object

System.ComponentModel.MarshalByValueComponent

System.Data.DataTable

### 10. 1 DataTable 개체 생성과 테이블 구조 설계

DataTable은 개체를 생성한 후에 테이블의 구조를 설계한 후에 사용합니다. 테이블의 구조를 설계한다는 것은 열을 추가하는 것과 기타 테이블 속성을 설정하는 것을 말합니다.

DataTable 클래스에서는 public 액세스 지정으로 설정한 세 가지 생성자를 제공합니다.

```
public DataTable ();  
public DataTable ( string table_name);  
public DataTable ( string table_name, string namespace);
```

DataTable을 생성한 이후에 구조 설계할 때는 DataColumn 개체를 생성하여 DataTable 개체에 추가하는 형태로 진행합니다.

#### 10.1.1 DataColumn

DataColumn 클래스는 DataTable의 구조를 정의할 때 사용하는 열의 스키마입니다.

DataColumn 클래스에서 제공하는 생성자는 다음과 같습니다.

```
public DataColumn ();  
public DataColumn ( string col_name);  
public DataColumn ( string col_name, Type data_type);  
public DataColumn ( string col_name, Type data_type, string expr); //열 생성 사용식  
public DataColumn ( string col_name, Type data_type, string expr, MappingType type);
```

DataRow 클래스에 제공하는 주요 속성은 다음과 같습니다.

속성	가져오기/설정하기	설명
AllowDBNull	가져오기/설정하기	null 허용 여부
AutoIncrement	가져오기/설정하기	열 값이 자동으로 증가 여부
AutoIncrementSeed	가져오기/설정하기	AutoIncrement 속성이 true 일 때 열의 시작 값 (기본값은 0)
AutoIncrementStep	가져오기/설정하기	AutoIncrement 속성이 true 일 때 열 값의 증가하는 폭
ColumnName	가져오기/설정하기	열의 이름
DataType	가져오기/설정하기	열의 데이터 형식
DefaultValue	가져오기/설정하기	기본 값
Unique	가져오기/설정하기	각 행에 있는 값이 고유해야 하는지 여부

[표 10.1] DataRow 속성

### 10.1.2 예제

이제 DataTable 개체를 생성하고 테이블 구조를 설계하는 예제 코드를 작성해 봅시다.

먼저 DataTable 개체를 생성합니다.

```
DataTable dt = new DataTable("Books");
```

DataRow 개체를 생성하고 속성을 설정한 후에 DataTable 개체의 Columns 컬렉션 속성에 추가합니다.

```
DataRow dc_title = new DataRow();
dc_title.ColumnName = "Title";
dc_title.DataType = typeof(string);
dc_title.AllowDBNull = false;
dt.Columns.Add(dc_title);
```

만약 DataTable에 주요 키를 설정하려면 DataTable 개체의 PrimaryKey 속성에 주요 키로 사용할 컬럼 배열로 설정합니다.

```
DataRow[] pkeys = new DataRow[1];
pkeys[0] = dc_isbn;
dt.PrimaryKey = pkeys;
```

```

static void Main(string[] args)
{
    DataTable dt = new DataTable("Books");
    DesingTable(dt);
    ViewAll(dt);
}

private static void DesingTable(DataTable dt)
{
    DataColumn dc_title = new DataColumn();
    dc_title.ColumnName = "Title";
    dc_title.DataType = typeof(string);
    dc_title.AllowDBNull = false;
    dt.Columns.Add(dc_title);

    DataColumn dc_isbn = new DataColumn("ISBN", typeof(string));
    dc_isbn.Unique = true;
    dc_isbn.AllowDBNull = false;
    dt.Columns.Add(dc_isbn);

    DataColumn dc_author = new DataColumn();
    dc_author.ColumnName = "Author";
    dc_author.DataType = typeof(string);
    dc_author.AllowDBNull = false;
    dt.Columns.Add(dc_author);

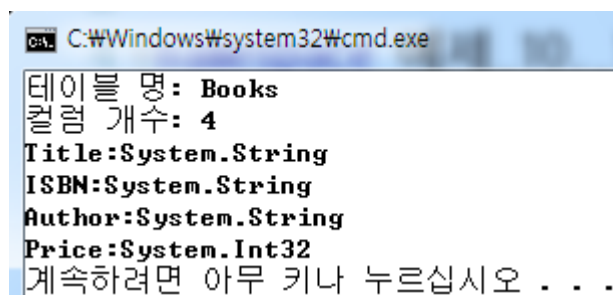
    DataColumn dc_price = new DataColumn();
    dc_price.ColumnName = "Price";
    dc_price.DataType = typeof(int);
    dc_price.AllowDBNull = false;
    dt.Columns.Add(dc_price);

    DataColumn[] pkeys = new DataColumn[1];
    pkeys[0] = dc_isbn;
    dt.PrimaryKey = pkeys;
}

```

```
private static void ViewAll(DataTable dt)
{
    Console.WriteLine("테이블 명: {0}", dt.TableName);
    Console.WriteLine("컬럼 개수: {0}", dt.Columns.Count);
    foreach (DataColumn dc in dt.Columns)
    {
        Console.WriteLine("{0}:{1}", dc.ColumnName, dc.DataType);
    }
}
```

[소스 10.1] 테이블 생성 및 구조 설계 예제 코드



```
C:\Windows\system32\cmd.exe
테이블 명: Books
컬럼 개수: 4
Title: System.String
ISBN: System.String
Author: System.String
Price: System.Int32
계속하려면 아무 키나 누르십시오 . . .
```

[그림 10.1] [소스 10.1] 실행 결과

## 10. 2 DataTable 개체에 행 추가

이번에는 DataTable 개체에 행을 추가하는 방법을 알아보시다.

DataTable 클래스에는 Rows 컬렉션 속성을 제공하고 있으며 추가한 Row 개체를 보관하는 컬렉션입니다. 따라서 DataTable 개체에 행을 추가할 때는 Row 클래스를 이용합니다.

그런데, Row 개체는 DataTable 개체에 설계한 구조에 맞아야 하기 때문에 직접 생성할 수 없고 DataTable 개체의 NewRow 메서드를 이용하여 설계한 구조에 맞는 Row 개체를 생성합니다.

```
DataRow dr = dt.NewRow();
```

그리고 DataRow 개체에 얼마다 원하는 값을 지정합니다.

```
dr["ISBN"] = isbn;
dr["Title"] = title;
dr["Author"] = author;
dr["Price"] = price;
```

주의할 점은 DataRow 개체를 DataTable 개체의 Rows 컬렉션 속성에 추가해야 합니다. 이 때 테이블 스키마에 위배가 발생하면 예외를 발생합니다.

```
dt.Rows.Add(dr);
```

```

static void Main(string[] args)
{
    DataTable dt = new DataTable("Books");
    DesignTable(dt);
    AddBooks(dt, "111-11-1111-111-1", "ADO.NET", "홍길동", 12000);
    AddBooks(dt, "111-11-1111-111-2", "XML.NET", "홍길동", 15000);
    ViewAll(dt);
}

private static void AddBooks(DataTable dt, string isbn, string title, string author, int price)
{
    try
    {
        DataRow dr = dt.NewRow();
        dr["ISBN"] = isbn;
        dr["Title"] = title;
        dr["Author"] = author;
        dr["Price"] = price;
        dt.Rows.Add(dr);
    }
    catch (Exception e)
    {
        Console.WriteLine("{0} 추가 실패", title);
        Console.WriteLine("이유:{0}", e.Message);
    }
}

private static void DesignTable(DataTable dt)
{
    DataColumn dc_title = new DataColumn();
    dc_title.ColumnName = "Title";
    dc_title.DataType = typeof(string);
    dc_title.AllowDBNull = false;
    dt.Columns.Add(dc_title);

    DataColumn dc_isbn = new DataColumn("ISBN", typeof(string));
    dc_isbn.Unique = true;
    dc_isbn.AllowDBNull = false;
    dt.Columns.Add(dc_isbn);
}

```

```

        DataColumn dc_author = new DataColumn();
        dc_author.ColumnName = "Author";
        dc_author.DataType = typeof(string);
        dc_author.AllowDBNull = false;
        dt.Columns.Add(dc_author);

        DataColumn dc_price = new DataColumn();
        dc_price.ColumnName = "Price";
        dc_price.DataType = typeof(int);
        dc_price.AllowDBNull = false;
        dt.Columns.Add(dc_price);

        DataColumn[] pkeys = new DataColumn[1];
        pkeys[0] = dc_isbn;
        dt.PrimaryKey = pkeys;
    }

    private static void ViewAll(DataTable dt)
    {
        int row_count = dt.Rows.Count;
        Console.WriteLine("테이블 명: {0}", dt.TableName);
        Console.WriteLine("행 개수:{0}", row_count);

        DataRow dr = null;
        for(int i = 0; i < row_count; i++)
        {
            dr = dt.Rows[i];

            foreach (DataColumn dc in dt.Columns)
            {
                Console.WriteLine("{0}:{1}", dc.ColumnName, dr[dc]);
            }

            Console.WriteLine("-----");
        }
    }
}

```

[소스 10.2] 행 추가 예제 코드

## 10. 3 DataTable과 XML

DataTable 클래스에서는 메모리 상의 테이블입니다. ADO.NET 에서는 논리적 DB를 XML 형태로 저장하고 로딩할 수 있게 기술을 지원하고 있습니다.

DataTable 클래스에도 WriteXml, WriteXmlSchema, LoadXml, LoadXmlSchema 메서드를 제공하여 메모리 상의 테이블 구조와 내용을 XML 형태로 저장하고 로딩할 수 있습니다.

```
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, bool hierarchy); //hierarchy: 테이블의 모든 하위 항목 내용 포함 여부
public void WriteXml(Stream stream, XmlWriteMode mode);
public void WriteXml(string filename, bool hierarchy);
public void WriteXml(string filename, XmlWriteMode mode);
public void WriteXml(TextWriter writer, bool hierarchy);
public void WriteXml(TextWriter writer, XmlWriteMode mode);
public void WriteXml(XmlWriter writer, bool hierarchy);
public void WriteXml(XmlWriter writer, XmlWriteMode mode);
public void WriteXml(Stream stream, XmlWriteMode mode, bool hierarchy);
public void WriteXml(string filename, XmlWriteMode mode, bool hierarchy);
public void WriteXml(TextWriter writer, XmlWriteMode mode, bool hierarchy);
public void WriteXml(XmlWriter writer, XmlWriteMode mode, bool hierarchy);
```

```
public void WriteXmlSchema(Stream stream);
public void WriteXmlSchema(String filename);
public void WriteXmlSchema(TextWriter writer);
public void WriteXmlSchema(XmlWriter writer);
public void WriteXmlSchema(Stream stream, bool hierarchy);
public void WriteXmlSchema(String filename, bool hierarchy);
public void WriteXmlSchema(TextWriter writer, bool hierarchy);
public void WriteXmlSchema(XmlWriter writer, bool hierarchy);
```

```
public XmlReadMode ReadXml(Stream stream);
public XmlReadMode ReadXml(string filename);
public XmlReadMode ReadXml(TextWriter writer);
public XmlReadMode ReadXml(XmlWriter writer);
```

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

DataTable 개체의 구조를 파일에 저장할 때는 WriteXmlSchema 메서드를 사용합니다.  
dt.WriteXmlSchema(schema\_fname, true);

DataTable 개체에 보관한 데이터를 저장할 때는 WriteXml 메서드를 사용합니다.  
dt.WriteXml(fname, true);

DataTable 개체의 구조를 파일에서 로딩할 때는 ReadXmlSchema 메서드를 사용합니다.  
dt.ReadXmlSchema(schema\_fname);

파일에 있는 데이터를 DataTable 개체로 로딩할 때 ReadXml 메서드를 사용합니다.  
dt.ReadXml(fname);

```
using System;
using System.Data;
using System.IO;

namespace 예제|_10._3_도서_관리_프로그램
{
    class BookManager
    {
        DataTable dt = new DataTable("Books");
        readonly string schema_fname = "books.xsd";
        readonly string fname = "book.xml";
        static BookManager bm = null;

        internal static void Run()
        {
            bm = new BookManager();
            bm.Start();
            bm.Save();
        }

        private void Save()
        {
            dt.WriteXml(fname, true);
        }
    }
}
```



```

private void Start()
{
    ConsoleKey key;

    while ((key = SelectMenu()) != ConsoleKey.Escape)
    {
        switch (key)
        {
            case ConsoleKey.F1: AddBook(); break;
            case ConsoleKey.F2: RemoveBook(); break;
            case ConsoleKey.F3: FindBook(); break;
            case ConsoleKey.F4: ListBook(); break;
            default: Console.WriteLine("잘못 선택하였습니다."); break;
        }

        Console.WriteLine("아무 키나 누르세요.");
        Console.ReadKey();
    }
}

private void ListBook()
{
    int row_count = dt.Rows.Count;
    Console.WriteLine("보유 도서수:{0}", row_count);

    foreach(DataRow dr in dt.Rows)
    {
        ViewBook(dr);
    }
}

```

```

private void FindBook()
{
    string isbn = string.Empty;
    Console.WriteLine("검색할 도서의 ISBN을 입력하세요.");
    isbn = Console.ReadLine();
    DataRow dr = dt.Rows.Find(isbn);
    if (dr == null)
    {
        Console.WriteLine("존재하는 ISBN이 아닙니다.");
        return;
    }
    ViewBook(dr);
}

private void ViewBook(DataRow dr)
{
    foreach (DataColumn dc in dt.Columns)
    {
        Console.WriteLine("{0}:{1}", dc.ColumnName, dr[dc]);
    }
    Console.WriteLine("-----");
}

private void RemoveBook()
{
    string isbn = string.Empty;
    Console.WriteLine("삭제할 도서의 ISBN을 입력하세요.");
    isbn = Console.ReadLine();
    DataRow dr = dt.Rows.Find(isbn);
    if (dr == null)
    {
        Console.WriteLine("존재하는 ISBN이 아닙니다.");
        return;
    }
    dt.Rows.Remove(dr);
    Console.WriteLine("삭제하였습니다.");
}

```

```

private void AddBook()
{
    string isbn = string.Empty;
    string title = string.Empty;
    string author = string.Empty;
    int price = 0;

    Console.WriteLine("추가할 도서의 ISBN을 입력하세요.");
    isbn = Console.ReadLine();

    if (Exist(isbn))
    {
        Console.WriteLine("이미 존재하는 ISBN입니다.");
        return;
    }
    Console.WriteLine("도서명을 입력하세요.");
    title = Console.ReadLine();

    Console.WriteLine("저자명을 입력하세요.");
    author = Console.ReadLine();

    Console.WriteLine("가격을 입력하세요.");
    if (int.TryParse(Console.ReadLine(), out price) == false)
    {
        Console.WriteLine("잘못 입력하였습니다.");
        return;
    }

    AddBook(isbn, title, author, price);
}

```

```

private void AddBook(string isbn, string title, string author, int price)
{
    try
    {
        DataRow dr = dt.NewRow();
        dr["ISBN"] = isbn;
        dr["Title"] = title;
        dr["Author"] = author;
        dr["Price"] = price;
        dt.Rows.Add(dr);
        Console.WriteLine("{0} 추가 성공", title);
    }
    catch (Exception e)
    {
        Console.WriteLine("{0} 추가 실패", title);
        Console.WriteLine("이유:{0}", e.Message);
    }
}

private bool Exist(string isbn)
{
    return dt.Rows.Find(isbn) != null;
}

private ConsoleKey SelectMenu()
{
    Console.Clear();
    Console.WriteLine("도서 관리 프로그램 [ESC]: 종료");
    Console.WriteLine("F1:도서 추가 F2:도서 삭제 F3:도서 검색 F4:전체보기");
    return Console.ReadKey().Key;
}

```

```

private BookManager()
{
    if (File.Exists(schema_fname))
    {
        dt.ReadXmlSchema(schema_fname);

        if (File.Exists(fname))
        {
            dt.ReadXml(fname);
        }
    }
    else
    {
        TableDesign();
        dt.WriteXmlSchema(schema_fname, true);
    }
}

private void TableDesign()
{
    DataColumn dc_title = new DataColumn();
    dc_title.ColumnName = "Title";
    dc_title.DataType = typeof(string);
    dc_title.AllowDBNull = false;
    dt.Columns.Add(dc_title);

    DataColumn dc_isbn = new DataColumn("ISBN", typeof(string));
    dc_isbn.Unique = true;
    dc_isbn.AllowDBNull = false;
    dt.Columns.Add(dc_isbn);

    DataColumn dc_author = new DataColumn();
    dc_author.ColumnName = "Author";
    dc_author.DataType = typeof(string);
    dc_author.AllowDBNull = false;
    dt.Columns.Add(dc_author);
}

```

```

        DataColumn dc_price = new DataColumn();
        dc_price.ColumnName = "Price";
        dc_price.DataType = typeof(int);
        dc_price.AllowDBNull = false;
        dt.Columns.Add(dc_price);

        DataColumn[] pkeys = new DataColumn[1];
        pkeys[0] = dc_isbn;
        dt.PrimaryKey = pkeys;
    }
}

class Program
{
    static void Main(string[] args)
    {
        BookManager.Run();
    }
}
}

```

[소스 10.3] DataTable과 XML 파일을 이용한 도서 관리 응용 예제 코드

## 11. DataSet

DataSet은 ADO.NET 핵심 구성 요소로 프로그램 내의 논리적 DB로 데이터 집합과 관계로 구성합니다.

DataTable은 메모리 상의 하나의 테이블을 표현하는 개체이고 DataSet은 메모리 상의 DB를 표현하는 개체입니다. 따라서 DataSet은 여러 개의 테이블과 관계들의 집합체라고 볼 수 있습니다.

DataSet 개체를 구성할 DataTable 개체들은 Tables 컬렉션에 보관하고 관계는 Relations 컬렉션에 보관합니다. DataSet 개체도 DataTable 처럼 스키마와 데이터를 저장하거나 읽어와서 구성하는 메서드를 제공합니다.

DataSet 클래스에는 DataSet 개체를 생성할 때 사용하는 생성자를 제공합니다.

```
public DataSet ( );  
public DataSet ( string name);
```

DataSet 개체를 생성한 후에는 DB를 구성할 테이블과 관계를 정의할 수 있습니다. 이러한 작업을 DB 설계라고 말합니다.

DB를 설계할 때는 DB에 포함할 테이블을 정의하여 DataSet에 추가하고 서로 다른 테이블에 있는 열 간의 관계를 정의하여 DataSet 개체에 추가합니다.

```
DataTable dt_publishers = new DataTable("Publishers");  
PublishersTableDesign(dt_publishers);  
ds.Tables.Add(dt_publishers);  
DataTable dt_books = new DataTable("Books");  
BooksTableDesign(dt_books);  
ds.Tables.Add(dt_books);  
  
DataColumn dc_publishers_cid = dt_publishers.Columns["CID"];  
DataColumn dc_books_cid = dt_books.Columns["CID"];  
  
DataRelation dr = new DataRelation("도서출판", dc_publishers_cid, dc_books_cid);  
ds.Relations.Add(dr);
```

다음은 출판사, 도서 테이블을 갖는 DataSet을 디자인하고 간단하게 데이터를 추가하는 예제입니다.

```
using System;
using System.Data;

namespace 예제_11._1_DB_설계_및_데이터추가
{
    class Program
    {
        static void Main(string[] args)
        {
            DataSet ds = new DataSet("Library");
            DataSetDesign(ds);

            AddPublisher(ds, "언제나휴일");
            AddPublisher(ds, "언제나평일");

            AddBooks(ds, "언제나휴일", "111-11-1111-111-1", "ADO.NET", "홍길동", 12000);
            AddBooks(ds, "언제나평일", "111-12-1111-111-1", "XML.NET", "강감찬", 15000);
            AddBooks(ds, "언제나휴일", "111-11-1111-111-2", "Enterprise Service", "홍길동", 20000);
            AddBooks(ds, "언제나평일", "111-12-1111-111-2", "C#", "이기백", 15000);

            View(ds);
        }

        private static void View(DataSet ds)
        {
            Console.WriteLine(ds.DataSetName);

            ViewTables(ds.Tables);
            ViewRelations(ds.Relations);
        }
    }
}
```



```

private static void ViewRelations(DataRelationCollection drc)
{
    Console.WriteLine("관계 수:{0}", drc.Count);
    foreach (DataRelation dr in drc)
    {
        Console.WriteLine("관계명:{0}", dr.RelationName);
        DataColumn dc_parent = dr.ParentColumns[0];
        DataColumn dc_child = dr.ChildColumns[0];
        Console.WriteLine("부모:{0}-{1}", dc_parent.Table.TableName, dc_parent.ColumnName);
        Console.WriteLine("자식:{0}-{1}", dc_child.Table.TableName, dc_child.ColumnName);
    }
}

private static void ViewTables(DataTableCollection dtc)
{
    Console.WriteLine("테이블 수:{0}", dtc.Count);
    foreach (DataTable dt in dtc)
    {
        ViewTables(dt);
    }
}

private static void ViewTables(DataTable dt)
{
    int row_count = dt.Rows.Count;
    Console.WriteLine("테이블 명: {0}", dt.TableName);
    Console.WriteLine("행 개수:{0}", row_count);
    DataRow dr = null;

    for (int i = 0; i < row_count; i++)
    {
        dr = dt.Rows[i];
        foreach (DataColumn dc in dt.Columns)
        {
            Console.WriteLine("{0}:{1}", dc.ColumnName, dr[dc]);
        }
    }
}

```

```

static void AddBooks(DataSet ds, string cname, string isbn, string title, string author, int price)
{
    try
    {
        DataTable dt = ds.Tables["Books"];
        int cid = GetCID(ds, cname);
        if (cid == -1)
        {
            throw new ApplicationException("존재하지 않는 출판사");
        }
        DataRow dr = dt.NewRow();
        dr["ISBN"] = isbn;
        dr["Title"] = title;
        dr["Author"] = author;
        dr["Price"] = price;
        dr["CID"] = cid;
        dt.Rows.Add(dr);
    }
    catch (Exception e)
    {
        Console.WriteLine("{0} 추가 실패", title);
        Console.WriteLine("이유:{0}", e.Message);
    }
}

private static int GetCID(DataSet ds, string cname)
{
    DataTable dt = ds.Tables["Publishers"];
    foreach (DataRow dr in dt.Rows)
    {
        if (cname == dr["CompanyName"])
        {
            return (int)dr["CID"];
        }
    }
    return -1;
}

```

```

private static void AddPublisher(DataSet ds, string cname)
{
    try
    {
        DataTable dt = ds.Tables["Publishers"];

        DataRow dr = dt.NewRow();
        dr["CompanyName"] = cname;
        dt.Rows.Add(dr);
    }
    catch (Exception e)
    {
        Console.WriteLine("{0} 출판사 추가 실패", cname);
        Console.WriteLine("이유:{0}", e.Message);
    }
}

private static void DataSetDesign(DataSet ds)
{
    DataTable dt_publishers = new DataTable("Publishers");
    PublishersTableDesign(dt_publishers);
    ds.Tables.Add(dt_publishers);

    DataTable dt_books = new DataTable("Books");
    BooksTableDesign(dt_books);
    ds.Tables.Add(dt_books);

    DataColumn dc_publishers_cid = dt_publishers.Columns["CID"];
    DataColumn dc_books_cid = dt_books.Columns["CID"];

    DataRelation dr = new DataRelation("도서출판", dc_publishers_cid, dc_books_cid);
    ds.Relations.Add(dr);
}

```

```

private static void PublishersTableDesign(DataTable dt)
{
    DataColumn dc_cname = new DataColumn();
    dc_cname.ColumnName = "CompanyName";
    dc_cname.DataType = typeof(string);
    dc_cname.AllowDBNull = false;
    dt.Columns.Add(dc_cname);

    DataColumn dc_cid = new DataColumn("CID", typeof(int));
    dc_cid.Unique = true;
    dc_cid.AllowDBNull = false;
    dc_cid.AutoIncrement = true;
    dc_cid.AutoIncrementSeed = 1;
    dc_cid.AutoIncrementStep = 1;
    dt.Columns.Add(dc_cid);

    DataColumn[] pkeys = new DataColumn[1];
    pkeys[0] = dc_cid;
    dt.PrimaryKey = pkeys;
}

private static void BooksTableDesign(DataTable dt)
{
    DataColumn dc_title = new DataColumn();
    dc_title.ColumnName = "Title";
    dc_title.DataType = typeof(string);
    dc_title.AllowDBNull = false;
    dt.Columns.Add(dc_title);

    DataColumn dc_isbn = new DataColumn("ISBN", typeof(string));
    dc_isbn.Unique = true;
    dc_isbn.AllowDBNull = false;
    dt.Columns.Add(dc_isbn);

    DataColumn dc_author = new DataColumn();
    dc_author.ColumnName = "Author";
    dc_author.DataType = typeof(string);
    dc_author.AllowDBNull = false;
    dt.Columns.Add(dc_author);
}

```

```

        DataColumn dc_price = new DataColumn();
        dc_price.ColumnName = "Price";
        dc_price.DataType = typeof(int);
        dc_price.AllowDBNull = false;
        dt.Columns.Add(dc_price);

        DataColumn dc_cid = new DataColumn("CID", typeof(int));
        dc_cid.AllowDBNull = false;
        dc_cid.DefaultValue = 0;
        dt.Columns.Add(dc_cid);

        DataColumn[] pkeys = new DataColumn[1];
        pkeys[0] = dc_isbn;
        dt.PrimaryKey = pkeys;
    }
}
}

```

[소스 11.1] DataSet 설계와 데이터 추가 예제 코드

DataSet 클래스도 DataTable 처럼 DataSet 구조를 스키마 파일로 저장하거나 로딩하는 메서드를 제공하고 있으며 내용을 저장하거나 로딩하는 메서드를 제공하고 있습니다. 사용법이 큰 차이가 없기 때문에 설명을 하지 않겠습니다.

## 12. DataView

ADO.NET 기술에서는 DataTable의 사용자 지정 뷰인 DataView 클래스를 제공합니다. DataTable에 있는 내용을 정렬하거나 탐색 및 필터링을 하기 위해 DataView 개체를 이용합니다. DataView 개체는 데이터를 저장하지 않으면 DataView 개체와 의존 관계에 있는 DataTable 개체의 뷰를 나타냅니다.

DataView에서는 원본 DataTable의 데이터에 관한 서로 다른 뷰를 동적으로 제공합니다. 따라서 뷰와 관련 없는 다른 테이블에 영향을 받지 않고 DataView에 없는 데이터를 사용자로부터 보호할 수 있습니다. 실제 윈도우즈 응용 프로그램이 웹 프로그래밍에서 데이터 바인딩 컨트롤을 사용할 때 DataView 개체를 많이 이용합니다.

▷ 클래스 상속 계층

System.Object

System.ComponentModel.MarshalByValueComponent

System.Data.DataView

### 12. 1 DataView 생성자

```
public DataView ( );  
public DataView (DataTable table);  
public DataView (DataTable table, string row_filter, string sort, DataViewRowState state);
```

다음은 Books 테이블에 있는 데이터 중에서 저자가 홍길동인 책들을 ISBN 순으로 정렬한 DataView 개체를 생성하는 코드입니다.

```
DataView view = new DataView(dt_books, "Author=홍길동", "ISBN", DataViewRowState.CurrentRows);
```

### 12. 2 DataView 속성

다음은 DataView 클래스의 주요 속성입니다.

속성	가져오기/설정하기	설명
AllowDelete	가져오기/설정하기	삭제 허용 여부
AllowEdit	가져오기/설정하기	편집 허용 여부
AllowNew	가져오기/설정하기	새 행을 추가할 수 있는지 여부
ApplyDefaultSort	가져오기/설정하기	기본 키로 정렬할 지 여부
Item	가져오기	의존 관계에 있는 테이블의 데이터(행)
Sort	가져오기/설정하기	정렬에 사용할 열
Table	가져오기/설정하기	의존 관계에 있는 테이블

[표 12.1] DataView 속성

## 12. 3 DataView 사용 예

이번에는 DataView 개체를 사용하는 간단한 예를 살펴볼게요.

여기에서는 Books 테이블에서 특정 저자가 쓴 책들을 ISBN 순으로 정렬한 DataView 개체를 생성한 후에 새로운 DataRowView 개체를 생성하여 추가하고 기존에 있던 데이터를 변경하는 예제입니다.

먼저 Books 테이블에서 저자가 홍길동인 책들을 ISBN 순으로 정렬한 DataView 개체를 생성합니다.

```
DataView dv = new DataView(dt, "Author='홍길동'", "ISBN", DataViewRowState.CurrentRows);
```

그리고 DataView 개체에 있는 정보를 출력해 보면 Books 테이블에 있는 데이터 중에 저자가 홍길동인 책들만 ISBN 순으로 출력됨을 확인할 수 있습니다.

```
Console.WriteLine("※ DataView 정보");
DataRowView drv = null;
for (int i = 0; i < dv.Count; i++)
{
    drv = dv[i];
    foreach (DataColumn dc in dt.Columns)
    {
        Console.WriteLine("{0}:{1}", dc.ColumnName, drv[dc.ColumnName]);
    }
    Console.WriteLine("-----");
}
```

```
※ Books 테이블 정보
테이블 명: Books
행 개수:4
Title:ADO.NET
ISBN:111-11-1111-111-1
Author:홍길동
Price:12000
-----
Title:XML.NET
ISBN:111-11-1141-111-1
Author:홍길동
Price:12000
-----
Title:ASP.NET
ISBN:111-11-1112-111-1
Author:홍길동
Price:12000
-----
Title:.NET
ISBN:111-11-1512-111-1
Author:강감찬
Price:12000
-----
```

```
※ DataView 정보
Title:ADO.NET
ISBN:111-11-1111-111-1
Author:홍길동
Price:12000
-----
Title:ASP.NET
ISBN:111-11-1112-111-1
Author:홍길동
Price:12000
-----
Title:XML.NET
ISBN:111-11-1141-111-1
Author:홍길동
Price:12000
-----
```

[그림 12.1] Books 테이블과 필터링 한 DataView 개체 정보 비교

DataGridView 개체를 이용해서 새로운 데이터를 추가할 때는 AddNew 메서드를 이용해서 DataRowView 개체를 생성합니다. 그리고 데이터를 설정후에 EndEdit 메서드를 이용하여 편집을 마감하여 의존 관계에 있는 테이블에 데이터를 추가합니다.

```
DataRowView drv = dv.AddNew();  
drv["ISBN"] = "111-11-1111-111-9";  
drv["Title"] = "C#";  
drv["Author"] = "홍길동";  
drv["Price"] = 1000;  
drv.EndEdit();
```

```
※ Books 테이블 정보  
테이블 명: Books  
행 개수:5  
Title:ADO.NET  
ISBN:111-11-1111-111-1  
Author:홍길동  
Price:12000  
-----  
Title:XML.NET  
ISBN:111-11-1141-111-1  
Author:홍길동  
Price:12000  
-----  
Title:ASP.NET  
ISBN:111-11-1112-111-1  
Author:홍길동  
Price:12000  
-----  
Title:.NET  
ISBN:111-11-1512-111-1  
Author:강감찬  
Price:12000  
-----  
Title:C#  
ISBN:111-11-1111-111-9  
Author:홍길동  
Price:1000  
-----
```

[그림 12.2] DataGridView 개체로 새로운 데이터를 추가한 결과 화면



DataGridView 개체는 인덱서를 제공하여 Items 컬렉션에 있는 DataRowView 요소를 참조할 수 있습니다.

```
DataRowView drv2 = dv[0];
```

```
drv2["Price"] = 20000;
```

```
-----2-----
※ Books 테이블 정보
테이블 명: Books
행 개수:5
Title:ADO.NET
ISBN:111-11-1111-111-1
Author:홍길동
Price:20000
-----
Title:XML.NET
ISBN:111-11-1141-111-1
Author:홍길동
Price:12000
-----
Title:ASP.NET
ISBN:111-11-1112-111-1
Author:홍길동
Price:12000
-----
Title:.NET
ISBN:111-11-1512-111-1
Author:강감찬
Price:12000
-----
Title:C#
ISBN:111-11-1111-111-9
Author:홍길동
Price:1000
-----
```

[그림 12.3] DataGridView 개체로 기존 데이터를 변경한 결과 화면

```

using System;
using System.Data;

namespace 예제_12.1_DataView_개체사용
{
    class ExUsingDataView
    {
        DataTable dt = null;

        static internal void Run()
        {
            ExUsingDataView eudv = new ExUsingDataView();
            eudv.UsingDataView();
        }

        ExUsingDataView()
        {
            dt = new DataTable("Books");
            Init();
            ViewAll();
        }

        void UsingDataView()
        {
            DataView dv = new DataView(dt, "Author='홍길동'", "ISBN", DataViewRowState.CurrentRows);
            ViewDataView(dv);
            Console.WriteLine("-----1-----");
            DataRowView drv = dv.AddNew();
            drv["ISBN"] = "111-11-1111-111-9";
            drv["Title"] = "C#";
            drv["Author"] = "홍길동";
            drv["Price"] = 1000;
            drv.EndEdit();
            ViewAll();
            Console.WriteLine("-----2-----");
            DataRowView drv2 = dv[0];
            drv2["Price"] = 20000;
            ViewAll();
        }
    }
}

```

```

private void ViewDataView(DataView dv)
{
    Console.WriteLine("※ DataView 정보");

    DataRowView drv = null;

    for (int i = 0; i < dv.Count; i++)
    {
        drv = dv[i];

        foreach (DataColumn dc in dt.Columns)
        {
            Console.WriteLine("{0};{1}", dc.ColumnName, drv[dc.ColumnName]);
        }
        Console.WriteLine("-----");
    }
}

void Init()
{
    DesignTable();

    AddBooks("111-11-1111-111-1", "ADO.NET", "홍길동", 12000);
    AddBooks("111-11-1141-111-1", "XML.NET", "홍길동", 12000);
    AddBooks("111-11-1112-111-1", "ASP.NET", "홍길동", 12000);
    AddBooks("111-11-1512-111-1", ".NET", "강감찬", 12000);
}

```

```
void AddBooks(string isbn, string title, string author, int price)
```

```
{  
    try  
    {  
        DataRow dr = dt.NewRow();  
        dr["ISBN"] = isbn;  
        dr["Title"] = title;  
        dr["Author"] = author;  
        dr["Price"] = price;  
        dt.Rows.Add(dr);  
    }  
    catch (Exception e)  
    {  
        Console.WriteLine("{0} 추가 실패", title);  
        Console.WriteLine("이유:{0}", e.Message);  
    }  
}
```

```
void DesignTable()
```

```
{  
    DataColumn dc_title = new DataColumn();  
    dc_title.ColumnName = "Title";  
    dc_title.DataType = typeof(string);  
    dc_title.AllowDBNull = false;  
    dt.Columns.Add(dc_title);  
  
    DataColumn dc_isbn = new DataColumn("ISBN", typeof(string));  
    dc_isbn.Unique = true;  
    dc_isbn.AllowDBNull = false;  
    dt.Columns.Add(dc_isbn);  
  
    DataColumn dc_author = new DataColumn();  
    dc_author.ColumnName = "Author";  
    dc_author.DataType = typeof(string);  
    dc_author.AllowDBNull = false;  
    dt.Columns.Add(dc_author);  
}
```

```

        DataColumn dc_price = new DataColumn();
        dc_price.ColumnName = "Price";
        dc_price.DataType = typeof(int);
        dc_price.AllowDBNull = false;
        dt.Columns.Add(dc_price);
        DataColumn[] pkeys = new DataColumn[1];
        pkeys[0] = dc_isbn;
        dt.PrimaryKey = pkeys;
    }

    void ViewAll()
    {
        int row_count = dt.Rows.Count;
        Console.WriteLine("※ Books 테이블 정보");
        Console.WriteLine("테이블 명: {0}", dt.TableName);
        Console.WriteLine("테이블 명: {0}", dt.TableName);
        Console.WriteLine("행 개수:{0}", row_count);
        DataRow dr = null;
        for (int i = 0; i < row_count; i++)
        {
            dr = dt.Rows[i];
            foreach (DataColumn dc in dt.Columns)
            {
                Console.WriteLine("{0};{1}", dc.ColumnName, dr[dc]);
            }
            Console.WriteLine("-----");
        }
    }
}

class Program
{
    static void Main(string[] args)
    {
        ExUsingDataView.Run();
    }
}
}

```

[소스 12.1] DataView 사용 예제 코드

## 13. SqlDataAdapter

SqlDataAdapter는 SQL 데이터 소스와 DataSet 간의 연결에 사용됩니다. Fill 메서드를 이용하여 데이터 소스의 데이터를 얻어와 DataSet을 채우고 Update 메서드를 이용하여 DataSet의 데이터로 데이터 소스의 데이터를 일치시키게 합니다.

SqlDataAdapter에 검색, 추가, 변경, 삭제에 사용할 SqlCommand를 초기에 설정한 후에 데이터 소스의 데이터를 Fill 메서드를 이용해 DataSet을 채우고 이 후에 작업은 DataSet으로 데이터를 관리하다가 데이터 소스를 변경할 필요가 있을 때 Update 메서드를 이용하여 데이터 소스에 반영시키는 것이 일반적인 사용입니다.

▷ 클래스 상속 계층

System.Object

System.MarshalByRefObject

System.ComponentModel.Component

System.Data.Common.DataAdapter

System.Data.Common.DbDataAdapter

System.Data.SqlClient.SqlDataAdapter

다음은 SqlDataAdapter 클래스에서 제공하는 생성자입니다.

```
public SqlDataAdapter ( );  
public SqlDataAdapter ( SqlCommand sel_command);  
public SqlDataAdapter ( string sel_text, SqlConnection con);  
public SqlDataAdapter ( string sel_text, string con_str);
```

SqlDataAdapter 개체는 Fill 메서드를 이용하여 SelectCommand로 탐색 결과를 DataSet 개체의 데이터를 채워줍니다. 이는 데이터 소스인 SQL 서버의 테이블의 데이터로부터 메모리 상의 DataSet 개체의 테이블 데이터를 채워주는 역할을 합니다.

```
adapter.SelectCommand = new SqlCommand(select * from Books, con);  
adapter.Fill(ds, "Books");
```

다음은 SqlDataAdapter 개체를 생성하여 Books 테이블과 Publishers 테이블의 내용을 DataSet 개체에 반영하는 예제 코드입니다.

```

static void Main(string[] args)
{
    string constr = @"Data Source=[서버 이름];Initial Catalog=[DB 명]; User ID=[ID];Password=[PW]";
    string comtxt_book = "select * from Books;";
    string comtxt_pub = "select * from Publishers";
    DataSet ds = new DataSet("Library");
    using (SqlConnection con = new SqlConnection(constr))
    {
        SqlDataAdapter adapter = new SqlDataAdapter();
        adapter.SelectCommand = new SqlCommand(comtxt_book, con);
        adapter.Fill(ds, "Books");
        adapter.SelectCommand = new SqlCommand(comtxt_pub, con);
        adapter.Fill(ds, "Publishers");
        ViewDataSet(ds);
    }
}

private static void ViewDataSet(DataSet ds)
{
    Console.WriteLine("DataSet 명:{0}", ds.DataSetName);
    Console.WriteLine("Table 수:{0}", ds.Tables.Count);
    foreach (DataTable dt in ds.Tables)
    {
        ViewDataTable(dt);
    }
}

private static void ViewDataTable(DataTable dt)
{
    Console.WriteLine("Table 명:{0}", dt.TableName);
    foreach (DataRow dr in dt.Rows)
    {
        foreach (DataColumn dc in dt.Columns)
        {
            Console.Write("{0} ", dr[dc.ColumnName]);
        }
        Console.WriteLine();
    }
}
}

```

[소스 13.1] SqlDataAdapter 개체 생성 및 Fill 메서드 사용 예제 코드

```

C:\Windows\system32\cmd.exe
DataSet 명:Library
Table 수:2
Table 명:Books
ADO.NET 12000 홍길동 123-456-789 1
XML.NET 15000 강감찬 456-111-111 2
Table 명:Publishers
언제나휴일 1
언제나평일 2
계속하려면 아무 키나 누르십시오 . . .

```

[그림 13.1] [소스 13.1] 실행 결과

프로그램 상의 DataSet의 내용을 데이터 소스인 SQL 서버에 반영할 때는 Update 메서드를 사용합니다.

```

public int Update ( DataRow[] rows );
public int Update ( DataSet ds );
public int Update ( DataTable dt );
public int Update ( DataSet ds, string table_name );

```

Update 메서드를 사용하려면 SqlDataAdapter개체의 InsertCommand, DeleteCommand, UpdateCommand 속성에 적절한 SqlCommand 개체를 설정해야 합니다.

```

string cmd_text = "update Books set Price=@Price where (ISBN = @ISBN)";
SqlCommand cmd = new SqlCommand(cmd_text, con);
cmd.Parameters.Add("@Price", SqlDbType.Int,4,"Price");
cmd.Parameters.Add("@ISBN", SqlDbType.VarChar,50,"ISBN");

```

```

DataTable dt_books = ds.Tables["Books"];
foreach (DataRow dr in dt_books.Rows)
{
    dr["Price"] = 0;
}
adapter.UpdateCommand = cmd;
adapter.Update(ds, "Books");

```

다음은 SqlDataAdapter 개체를 생성하여 Books 테이블과 Publishers 테이블의 내용을 DataSet 개체에 반영한 후에 Books 테이블의 모든 책의 가격 정보를 0으로 변경하고 이를 다시 SQL 서버의 Books 테이블에 반영하는 예제 코드입니다.



```

class Program
{
    static void Main(string[] args)
    {
        string constr = @"Data Source=[서버 이름];Initial Catalog=[DB 명]; User ID=[ID];Password=[PW]";
        string comtxt_book = "select * from Books;";
        string comtxt_pub = "select * from Publishers";

        DataSet ds = new DataSet("Library");
        using (SqlConnection con = new SqlConnection(constr))
        {
            SqlDataAdapter adapter = new SqlDataAdapter();
            adapter.SelectCommand = new SqlCommand(comtxt_book, con);
            adapter.Fill(ds, "Books");
            adapter.SelectCommand = new SqlCommand(comtxt_pub, con);
            adapter.Fill(ds, "Publishers");
            ViewDataSet(ds);
            DataTable dt_books = ds.Tables["Books"];
            foreach (DataRow dr in dt_books.Rows)
            {
                dr["Price"] = 0;
            }
            adapter.UpdateCommand = MakeUpdateCommand(con);
            adapter.Update(ds, "Books");
            Console.WriteLine("-----");
            ViewDataSet(ds);
        }
    }

    private static SqlCommand MakeUpdateCommand(SqlConnection con)
    {
        string cmd_text = "update Books set Price=@Price where (ISBN = @ISBN)";
        SqlCommand cmd = new SqlCommand(cmd_text, con);
        cmd.Parameters.Add("@Price", SqlDbType.Int, 4, "Price");
        cmd.Parameters.Add("@ISBN", SqlDbType.VarChar, 50, "ISBN");
        return cmd;
    }
}

```

```

private static void ViewDataSet(DataSet ds)
{
    Console.WriteLine("DataSet 명:{0}", ds.DataSetName);
    Console.WriteLine("Table 수:{0}", ds.Tables.Count);
    foreach (DataTable dt in ds.Tables){ ViewDataTable(dt); }
}

private static void ViewDataTable(DataTable dt)
{
    Console.WriteLine("Table 명:{0}", dt.TableName);
    foreach (DataRow dr in dt.Rows)
    {
        foreach (DataColumn dc in dt.Columns)
        {
            Console.Write("{0} ", dr[dc.ColumnName]);
        }
        Console.WriteLine();
    }
}
}

```

[소스 13.2] SqlDataAdapter Update 메서드 사용 예제 코드

```

C:\Windows\system32\cmd.exe
DataSet 명:Library
Table 수:2
Table 명:Books
ADO.NET 12000 홍길동 123-456-789 1
XML.NET 15000 강감찬 456-111-111 2
Table 명:Publishers
언제나휴일 1
언제나평일 2
-----
DataSet 명:Library
Table 수:2
Table 명:Books
ADO.NET 0 홍길동 123-456-789 1
XML.NET 0 강감찬 456-111-111 2
Table 명:Publishers
언제나휴일 1
언제나평일 2
계속하려면 아무 키나 누르십시오 . . .

```

[그림 13.2] [소스 13.2] 실행 결과

이상으로 XML.NET 기술과 ADO.NET 기술의 소개를 마칠게요. 보다 깊은 학습과 숙련을 하시기 바랍니다.