C++Builder ADO Programming (2) - TADOConnection

우리의 레밍이 ADO Programming 이라고 하는 모험에 멋지게 출발하려면 여러 가지 준비물이 필요한데 이제부터 한동안은 그 준비물에 대해 알아보도록 하자. 그 준비물이란 앞의 장에서 약간 훑어본 VCL Component 들에 대해서 숙지하는 것이다. (구체적으로 TADOConnection, TADOCommand, TADODataSet 이 3가지 Component와 함께 ADO 의 기본적인 아키텍쳐 구조가 어떻게 VCL 화 되어 있는지를 습득하는 것이다) 먼저 ADO의 Connection 객체가 VCL화 되어 있는 TADOConnection에 대해 알아보자.

TADOConnection Component (Connection 객체)

데이터베이스로부터 뭔가를 얻으려면 우선 데이터베이스에 대한 연결을 설정해야만 한다. ODBC를 포함한 기존의 다른 데이터베이스 접근 기술들도 연결의 설정은 가장 기본적인 작업이다. ADO는 데이터에 접근, 처리하는 방법에 있어서 기존의 방식보다는 훨씬 더 융통성이 있는데 ADO의 경우 Connection 객체를 이용해서 명시적으로 연결을 만들 수도 있지만 Connection 객체를 이용하지 않고도 직접 Command 객체나 Recordset 객체를 사용해서 즉석에서 연결하는 방법도 있다. 실제로 우리는 TRDSConnection을 제외한 ADO 탭의 모든 Component들이 ConnectionString 프로퍼티를 가지고 있는 것을 볼 수 있다. (이것은 각각의 Component들이 Connection 객체를 사용하지 않고 독립적으로 ConnectionString 을 이용하여 데이터 원본에 접근할 수 있다는 말이다) 물론 이것은 C++ Builder 프로그래머에게는 색다른 사실이 아니다. Data Access Component 탭의 TTable, TQuery 등의 Component를 TDatabase Component를 사용하지 않고 어플리케이션 내에서 자유롭게 사용하는 프로그래밍 스타일과 비슷하기 때문일 것이다. (추천할 만한 스타일은 아니며 Application내에서 데이터베이스 트랜잭션이나 각 모듈에서 공유하는 Data Module이 필요하지 않은 --- Database 관련 Component들을 직접 폼 위에 떨어뜨리는 방식으로 이루어 지는 --- 간단한 Application을 제작할 때 주로 사용되는 방법이다)

그렇다면 Connection 객체가 여전히 존재하는 이유는 무엇인지 그리고 이 강의가 왜Connection 객체에 대한 이야기를 하려는 이유가 궁금할 것이다. 이미 눈치가 빠른 사람이라면 앞의 글에서 TADOConnection이 TDatabase Component와 같은 기능과 역할을 할 것이라고 눈치를 챘을 것이다. 다시 설명하자면 Connection 객체가 여전히 필요한 이유는 데이터를 관리할 때에 몇 가지 특별한 기능을 제공하기 때문인데 ADO를 통해서 관리될 수 있는 트랜잭션은 오직 Connection 객체를 거쳐서만 만들어질 수 있고 ADO의 작업 도중에 발생한 에러들에 대한 정보를 담은 Errors 컬렉션들 또한 오직 Connection 객체를 통해서만 접근할 수 있다. 그것들은 이 강의 목적에 맞는 아주 중요한 사항이고 그 외에도 연결 풀링의 혜택, 데이터베이스 서버내의 Stored Procedure들과 명령에 대한 빠른 접근성 제공, 새로운 종류의 OLE DB 공급자들과 그에 대한 연결 제공등등 Connection 객체를 사용해야 될 이유는 여러 가지가 있다. 그렇다면 이번 강의에서는 Connection 객체의 개요와 연결의 생성과 해제에 대해서 알아보도록 하자.

Connection 객체의 개요

연결은 데이터에 대한 접근 수단을 제공해 주는 것으로 그러한 연결을 대표하는 것이 Connection 객체이고 C++ Builder에서는 그런 Connection 객체의 VCL Component인 TADOConnection Component를 사용한다는 사실을 이전의 설명을 통하여 알았을 것이다. 폼 이나 자료모듈, 웹 모듈에 TADOConnection Component를 떨어뜨리거나 코드를 통해 TADOConnection 클래스의 인스턴스를 생성한 후에 그것의 속성들에 데이터 원본 관련 정보들을 지정해 주면 데이터 소비자(클라이언트 어플리케이션)와 데이터 공급자(데이터 원본) 사이의 연결이 만들어진다. Access나 SQL Server와 같은 데이터 원본에 대한 연결이 만들어지고 나면 이 TADOConnection Component를 이용해서 직접 명령들을 수행하거나, 미리 정의된 저장 프로시저나 뷰 등 여러 가지 데이터베이스 서버 객체들을 수행해서 데이터를 조회, 조작할 수 있다. 또한 앞에서 언급한 대로 TADOConnection Component를 통해서 트랜잭션 범위를 제어할 수도 있다

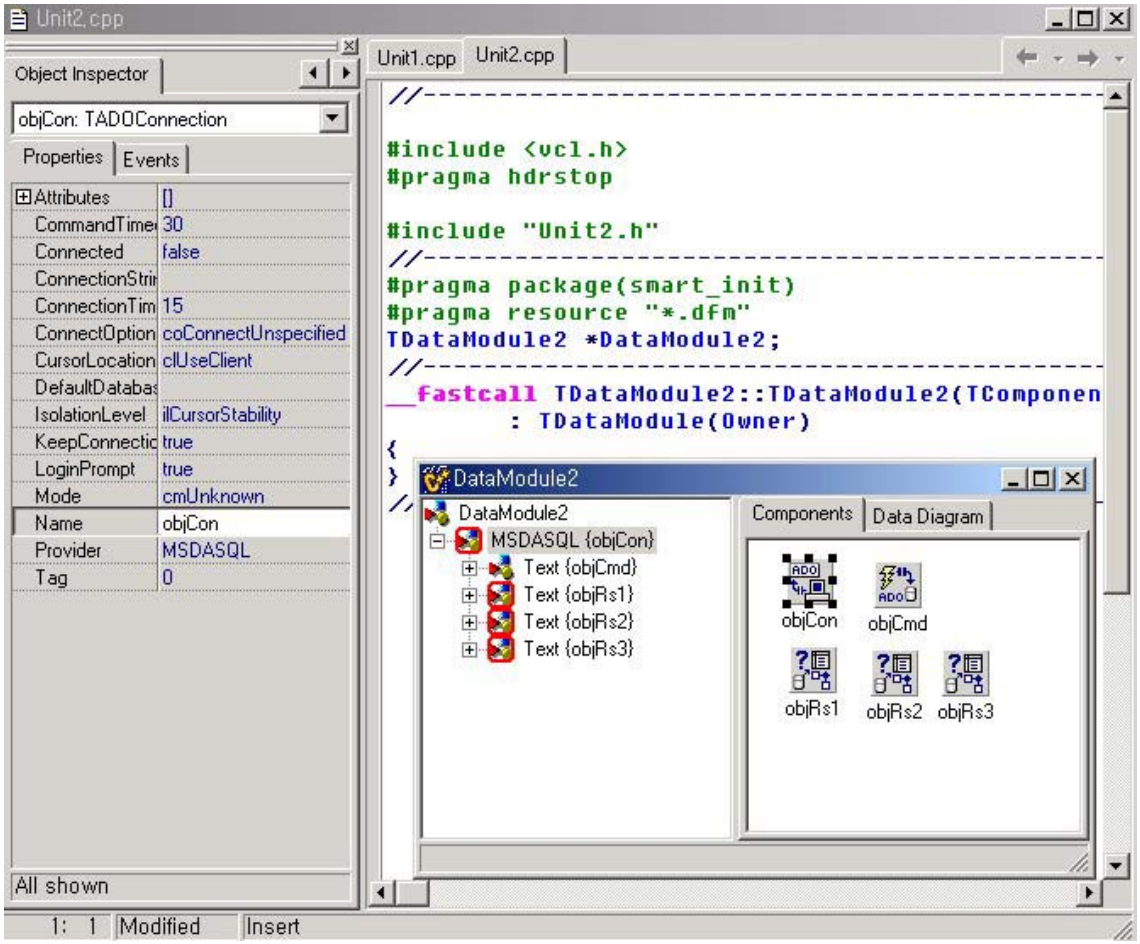
또한 이전 강의의 ADO 객체모델의 그림을 통해서도 알 수 있는 바와 같이 Connection 객체는 Errors 컬렉션을 제공하는데 물론 TADOConnection Component를 이용하면 Errors 컬렉션 전체에 쉽게 접근 할 수 있다. 그리고 Connection 객체는 연결 풀링의 장점도 제공해준다. 연결 풀링이란 연결을 명시적으로 종료한 후에도 그 연결을 일종의 풀에 보관해 두는 기법으로 역시 TADOConnection Component를 이용하면 연결 풀링의 장점도 얻을 수 있다.

솔직히 데이터베이스 프로그래밍을 할 때 가장 시간도 많이 잡아먹고 자원이 많이 소비되는 부분이 서버에로의 연결에 관계된 작업이므로 이 연결 풀링을 이용하면 Connection 객체를 사용하지 않는 경우보다 데이터의 요청이 빈번한 어플리케이션의 경우 커다란 성능의 향상을 볼 수 있다. 간단히 말해 연결 풀링의 혜택을 받기 위해서는 어플리케이션 제작 시 TADOConnection Component를 사용해 연결을 생성하고 다른 ADO Component의 Connection 프로퍼티에 생성된 TADOConnection을 설정하면 된다.

Connection의 생성과 해제

TADOConnection Component를 마우스로 클릭하여 폼 위나 자료 모듈, 웹 모듈 위에 떨어뜨리면 된다. 단지 그것만으로 Connection 객체의 인스턴스가 생성되고 이렇게 만들어진 TADOConnection Component를 이용해서 연결을 정의하고 열고 명령을 수행하거나 데이터를 조회하고 작업이 끝나면 종료하면 된다. 이때 만들어진 TADOConnection Component의 인스턴스는 데이터 공급자와의 고유한 세션을 대표하며 연결의 각 인자들을 다시 정의하고 그것을 이용해서 다른 데이터 공급자에 연결하는 것도 가능하다. 그러나 이런 방법은 권장할 것이 못 되며 실제적인 어플리케이션들에서는 각각의 데이터 공급자들마다 서로 다른 TADOConnection Component의 인스턴스들을 생성해서 사용하는 것이 일반적이다. 다음은 각각의 어플리케이션에서 쓰이는 TADOConnection Component의 예이다.

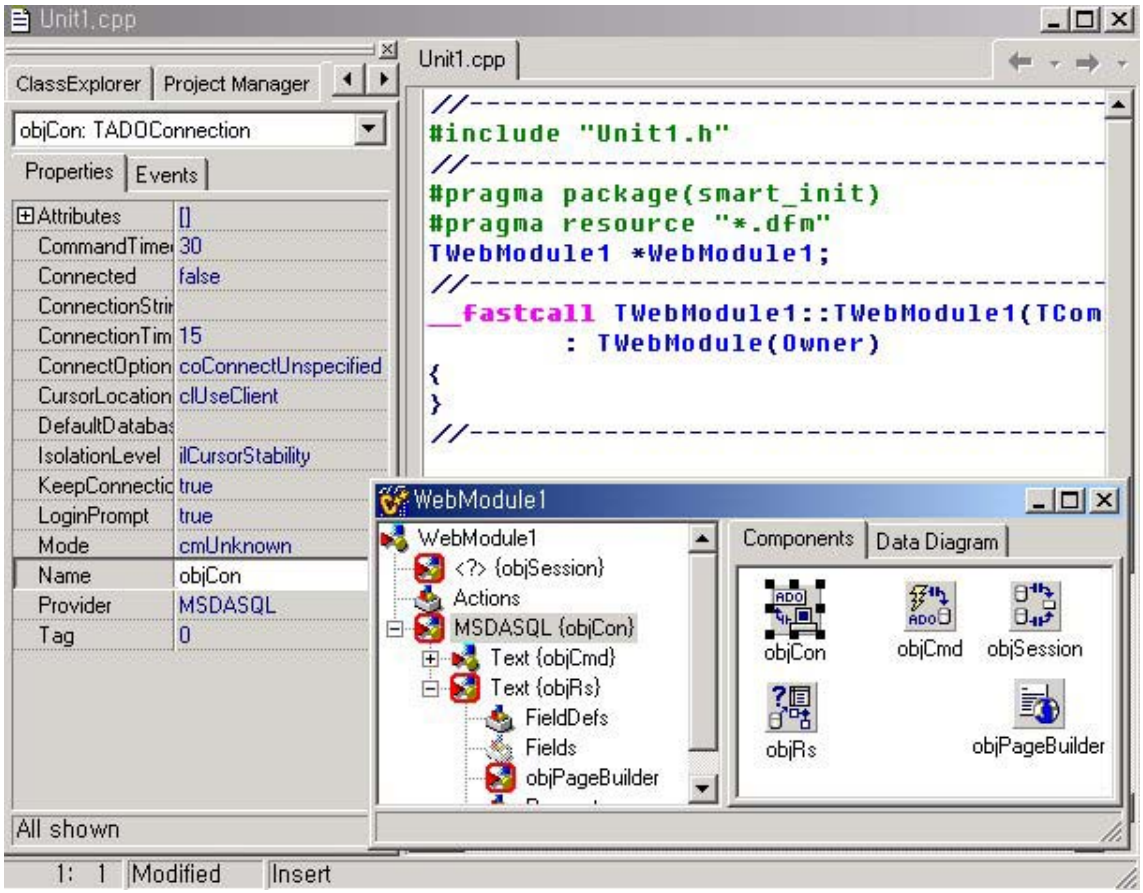
자료 모듈상에서의 TADOConnection Component의 인스턴스



아래 그림은 위의 자료모듈 헤더파일의 소스의 일부분이다. ADO탭의 TADOConnection Component를 사용하지 않고 코드를 통해 TADOConnection의 인스턴스를 생성할 경우 헤더나 소스파일에 ADODB.hpp와 DB.hpp를 추가해야 한다. 앞으로 나올 웹 모듈과 폼 상에서도 이 사항은 마찬 가지이다.

```
#include <Forms.hpp>
#include <ADODB.hpp>
#include <Db.hpp>
//---------------------------------------------------------
class TDataModule2 : public TDataModule
{
__published:    // IDE-managed Components
        TADOConnection *objCon;
        TADOCommand *objCmd;
        TADODataSet *objRs1;
        TADODataSet *objRs2;
        TADODataSet *objRs3;
private:        // User declarations
public:         // User declarations
        __fastcall TDataModule2(TComponent* Owner);
};
```

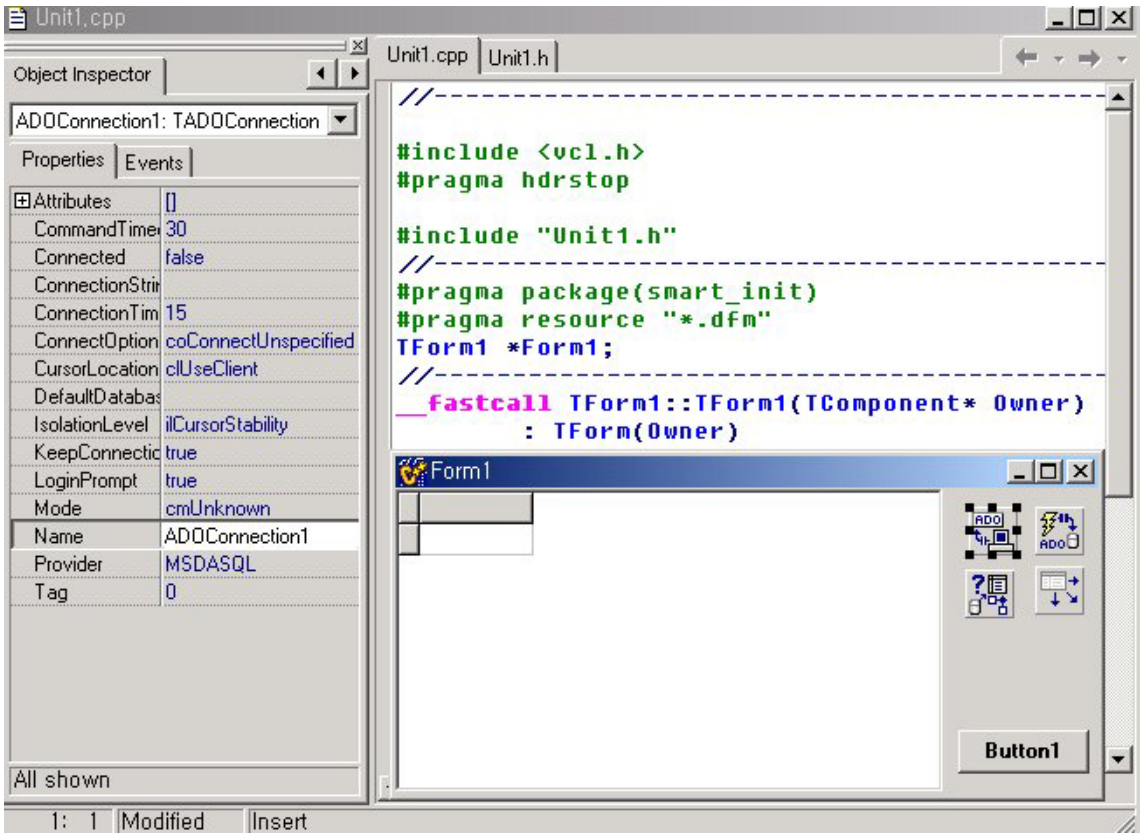
웹 모듈상에서의 TADOConnection Component의 인스턴스



아래 그림은 위의 웹 모듈의 헤더 소스 파일의 일부분이다. 역시 자료 모듈에서의 경우와 같이 ADODB.hpp와 DB.hpp 헤더 파일이 추가 되었음을 알 수 있다..

```
#include <ADODB.hpp>
#include <Db.hpp>
#include <DBTables.hpp>
#include <DSProd.hpp>
//---------------------------------------------
class TWebModule1 : public TWebModule
{
__published:    // IDE-managed Components
        TADOConnection *objCon;
        TADOCommand *objCmd;
        TADODataSet *objRs;
        TDataSetPageProducer *objPageBuilder;
        TSession *objSession;
```

폼 상에서의 TADOConnection Component의 인스턴스



아래 그림은 위의 폼의 헤더 소스 파일의 일부분이다. 역시 ADODB.hpp와 DB.hpp 헤더파일이 추가되었음을 알 수 있다.

```
#include <ADODB.hpp>
#include <Db.hpp>
#include <DBGrids.hpp>
#include <Grids.hpp>
//---------------------------------------
class TForm1 : public TForm
{
__published:    // IDE-managed Components
        TADOConnection *objCon;
        TADOCommand *objCmd;
        TADODataSet *objRs;
        TDataSource *DataSource1;
        TDBGrid *DBGrid1;
        TButton *Button1;
```

아래는 TADOConnection를 쓰지 않고 코드를 통해 동적으로 TADOConnection Class를 생성하고 연결을 설정한 후 사용하는 방법이다. 물론 헤더 파일이나 소스파일에 ADODB.hpp와 DB.hpp를 추가 해야 한다.

```
TADOConnection *objCon;  // Connection 객체 선언.

try
{
   objCon = new TADOConnection(AOwner);
   // Connection 객체 인스턴스 생성.
   //.........................
   // 여러가지 세부설정을 한다.
   objCon->Open();  // Connection 연결을 연다.
   //.........................
   // 여러가지 작업을 한다.

   if (objCon->State == (TObjectStates() << stOpen))
       objCon->Close(); // 연결을 닫는다.
}
__finally
{
   delete objCon;   // 혹은 objCon->Free();
   // 사용된 Connection 객체를 해제한다.
}
```

다섯째 줄의 TADOConnection의 생성자의 인자인 AOwner는 소유자를 나타낸다. 폼 위주의 어플리케이션일 경우 폼 이름이나 Application, Null등을 사용한다. 또 각종 서버 Component나 오브젝트(DLL 형태의 비즈니스 로직 객체를 말한다) 작성시, 쓰레드나 서비스 프로그램에서 ADO를 사용할 경우 연결 객체 생성시 Null이 무난하다.

Close 메소드를 호출한다는 것 --- 즉 대부분의 경우 연결을 닫고 연결에 할당된 시스템 자원들을 해제하는 것 --- 은 데이터 원본과의 연결이 끊어지는 것을 의미한다. 연결이 해제되기 전에 연결을 가지고 있던 모든 데이터 셋(레코드 셋)이 자동적으로 닫히게 된다. 그러나 연결을 닫기 전에 레코드 셋들을 명시적으로 닫아주는 것이 더 좋다.

이전의 활성 연결이 닫히고 다시 연결이 재 오픈 되었다고 해서 닫혔던 레코드 셋들이 자동적으로 열리거나 하지는 않는다. 그 Connection 객체에 연결된 어느 레코드 셋이든 그것들 각각을 다시 열어주어야 한다.

Close 메소드를 호출하는 것은 TADOConnection Component의 Connected 속성을 false로 설정하는 것과 같은데 코드의 명시성 때문에 필자는 Close 메소드를 선호한다. 연결이 닫히고 해제되면 그 연결은 연결 풀링으로 회수되며 일정 시간 동안 데이터 원본내의 설정에 맞게 대기 후 다시 빠르게 재사용되고 사용되지 않을 경우 폐기된다. 그러니까 연결 풀링은 데이터베이스 인스턴스가 서버 내에서 활성화 될 때 설정에 맞게 생성된 일종의 메모리 구조체 버퍼라고 생각하면 좋을 것이다. 연결 풀링에 대한 좀 더 자세한 내용은 차후의 강의에서 알아보도록 하자.

닫은 연결에 관계된 TADOCommand 객체들은 열려진 상태로 남게 되지만 Command 객체 자체의 Connection 속성은 비워지게 된다. 그리고 Command 객체를 사용하던 모든 레코드 셋들의 연결이 끊어지게 된다. 이들 세 Component의 관계는 차후의 강의에서 보다 자세하고 구체적으로 알아보도록 하자.

그리고 마지막 줄에서 보는 것과 같이 TADOConnection을 코딩을 통해 동적으로 생성하고 사용하는 경우 작업을 다한 후 연결을 해제하고 Connection 객체 자체를 제거하는 과정을 거쳐야 한다. (이 과정은 비단 프로

그래밍에만 국한되는 내용은 아닐 것이다. 자신이 벌이고 한 일은 책임을 져주는 것은 삶의 덕목일 것이다)

사실 이번 강의는 TADOConnection Component의 개요에 불과하다. 언제나 모든 서론들이 그렇듯이 적극적이고 열성적인 사람들에게는 지루하게 느껴질 수 있다. 야금야금 강의에 대단히 죄송하며 다음 강의는 ConnectionString을 포함한 TADOConnection Component의 핵심적인 속성들과 TADOConnection Component의 좀 더 자세한 설정 방법에 대해 광범위한 고찰을 할 예정이다.

Mortalpain