2NE4 팀의 발족 이후 본격적으로 프로젝트가 시작되었습니다. 프로젝트 내용에 대해선 차차 얘기를 해 보구요,

뭐 어떤 어플리케이션을 만들든 현재의 양상으로는 DB와 네트워크가 빠질 수가 없지요.

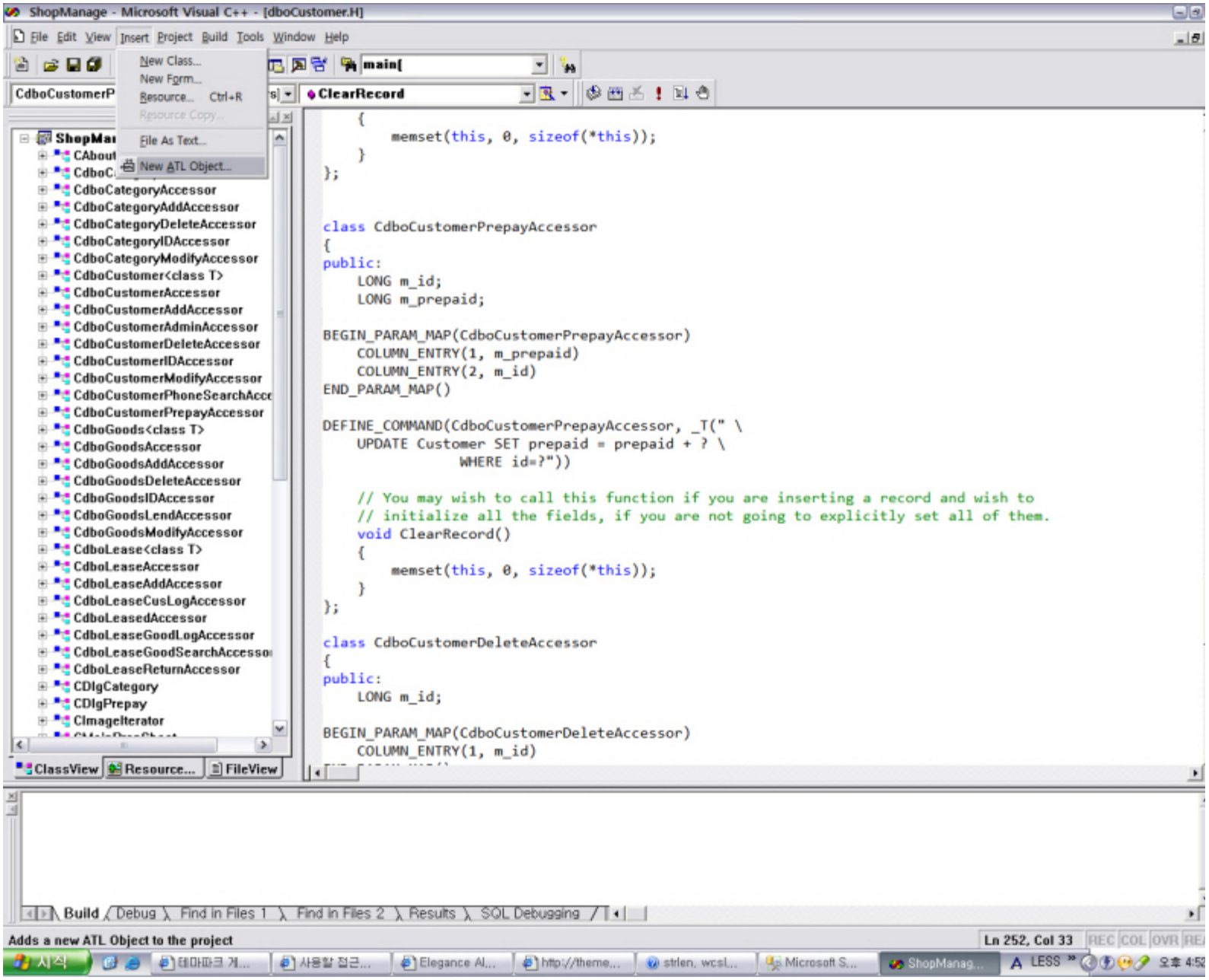
그래서 오늘은 먼저 DB에 대한 내용을 정리를 해 볼까 합니다.

저희가 공부를 할 때는 VC++ 6.0을 기반으로 ATL object를 추가하는 방식으로 DB를 연결했었는데 VC++ 2008로 넘어오면서 인터페이스의 변화가 생기며 조금의 혼동이 생겼습니다.

그것을 해결하기위해 검색을 했지만 딱히 정리가 된 부분이 없어서.. 한번 정리를 해 보고자 합니다.

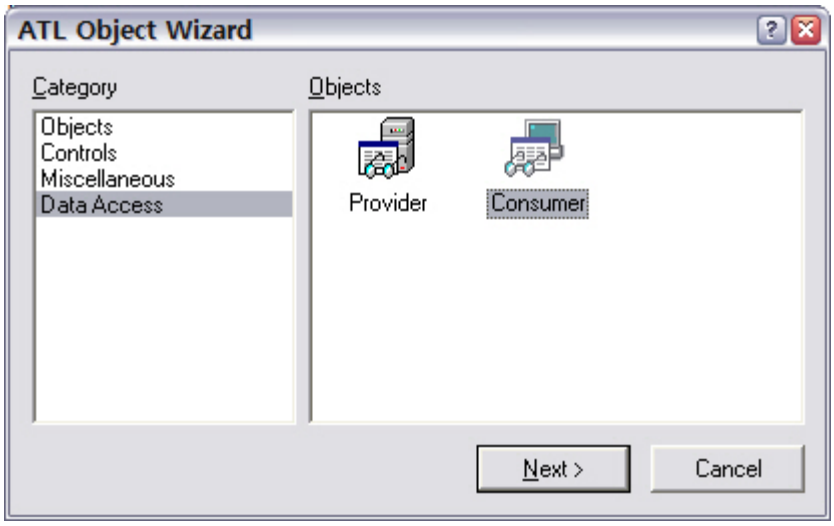
먼저 2008 버젼으로 해보기 전, 6.0에서 ATL 개체를 추가하는 방식으로 DB연결하는 것을 리뷰해 보겠습니다.

작업하던 MFC 프로젝트에서 Insert의 New ATL Object를 클릭합니다.

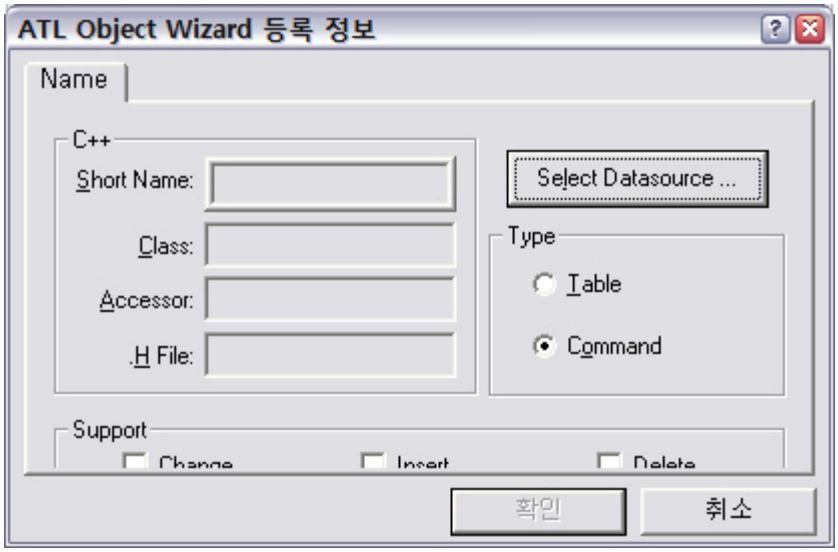


위 화면은 제가 예전에 비디오샵 관리 프로그램을 만들던 당시의 화면입니다. 생각해보니 다른 DB를 연결해야해서 새로운 프로젝트를 하나 생성해서 계속 진행하였습니다.

어쨌든 ATL Object 추가를 누르면 아래와 같은 창이 나타납니다.

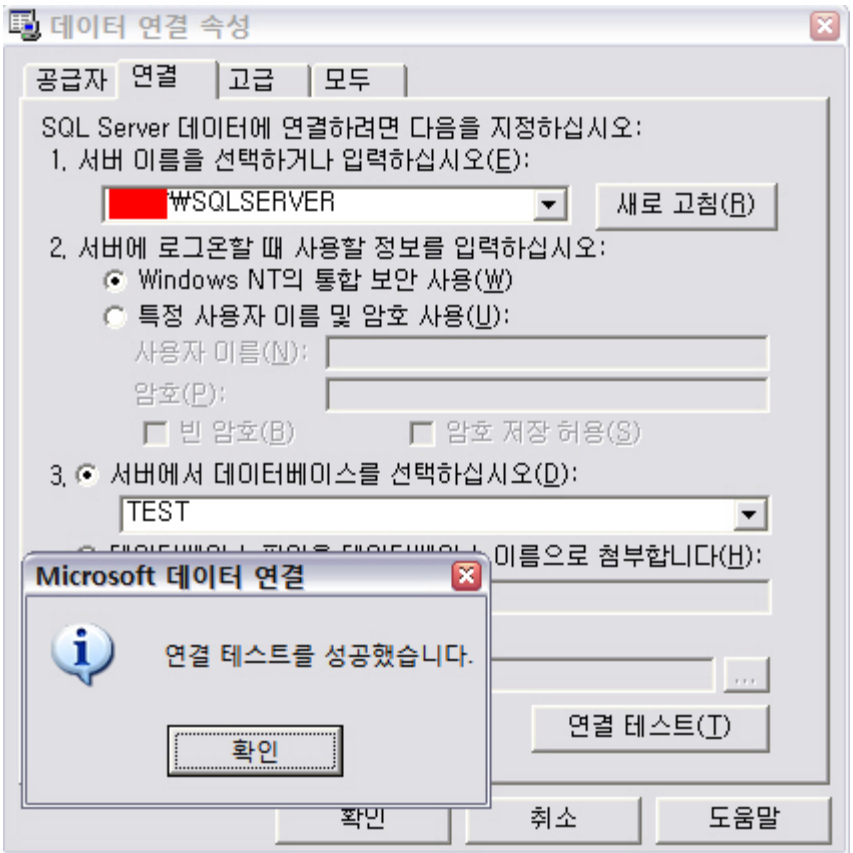


위와 같은 창이 뜨면 Data Access의 Consumer를 선택하고 Next를 클릭합니다.



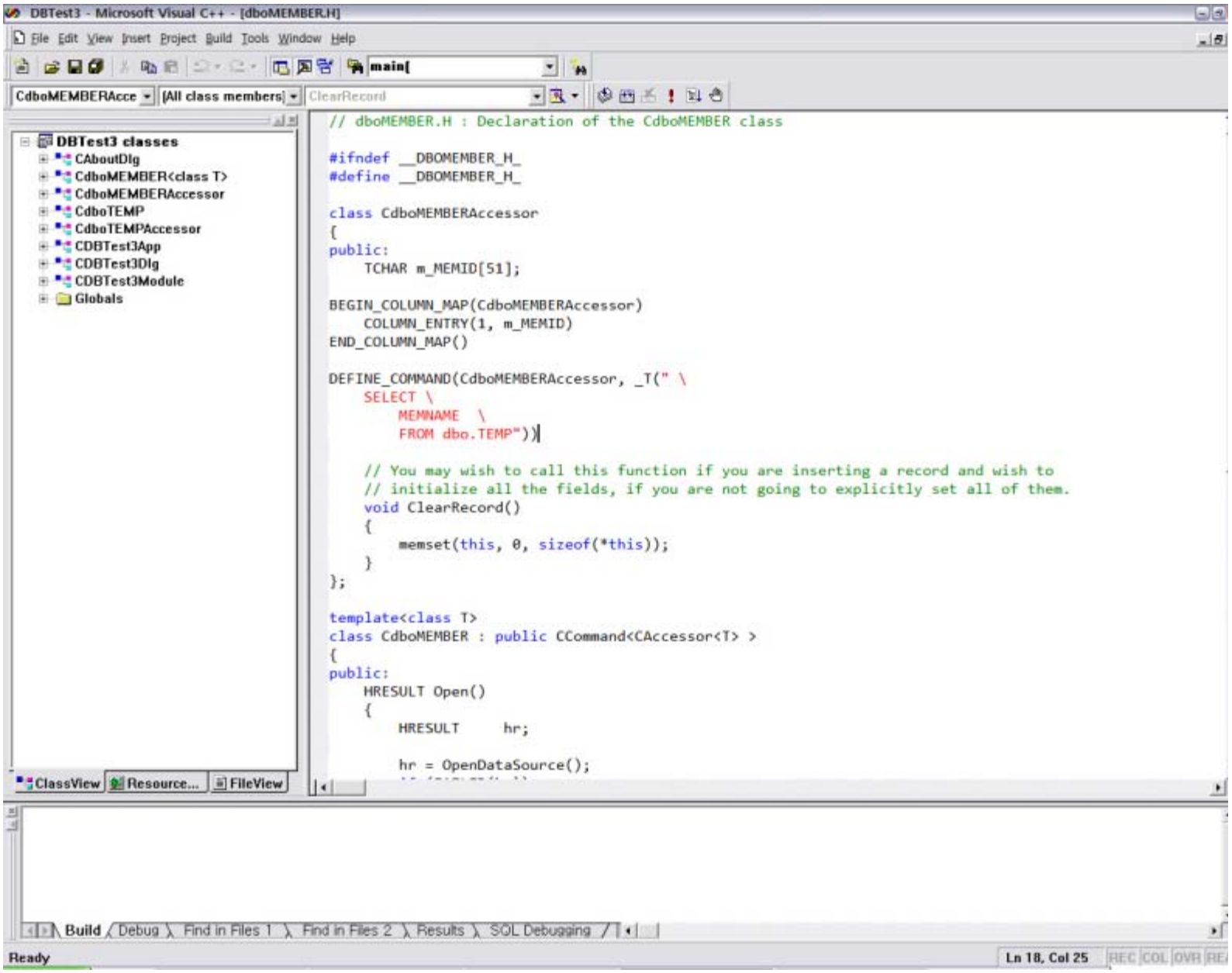
그럼 원본 DataBase에 연결을 해야겠죠? Select Datasource를 클릭합니다.

저희는 MS SQL을 사용합니다. 그래서 OLE DB Provider for SQL Server를 선택하였습니다. 다른 DB를 사용하시는 분은 다른 항목을 선택해 주세요.
선택이 완료되었으면 다음을 누릅니다.



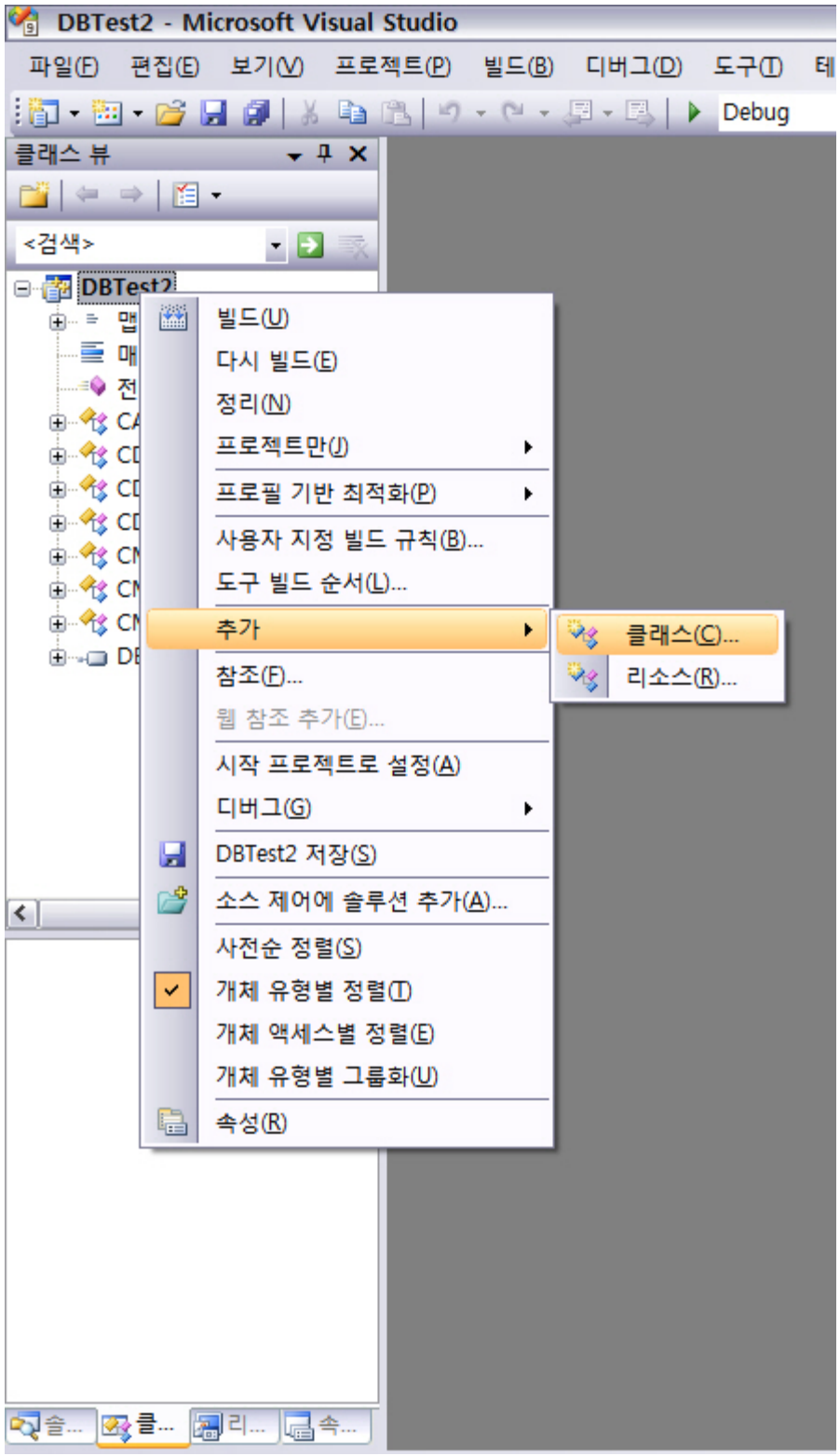
그럼 위와 같은 창이 뜨는데 SQL Server에 로그인 할 때 사용하는 서버 이름과 설정을 그대로 입력하시면 됩니다. 다 입력하셨으면 연결테스트를 눌러

보시고 위와 같은 창이 뜨면 성공적으로 연결을 한 것입니다. 확인을 누르고 설정을 마칩니다.

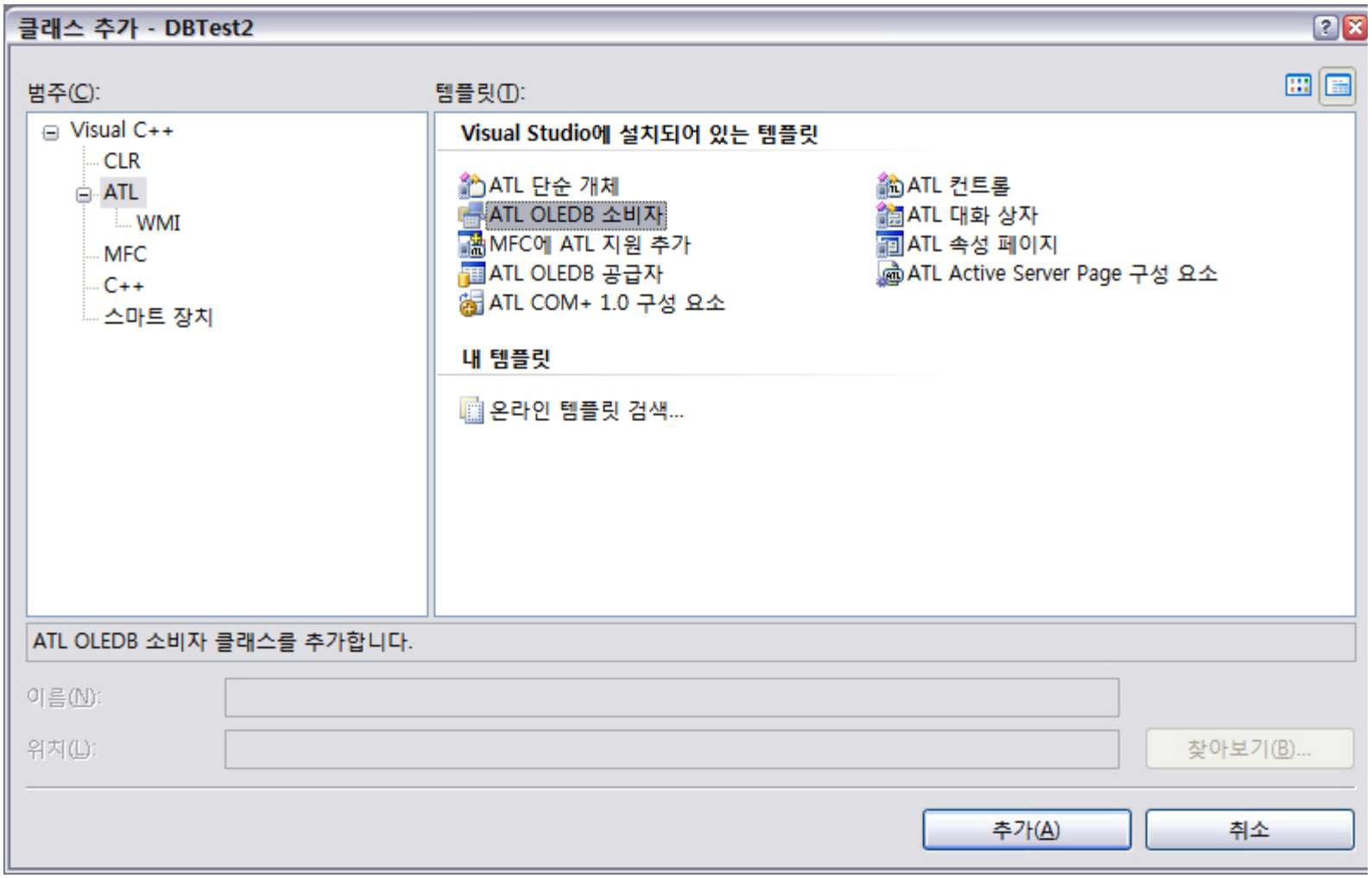


그럼 위와같은 화면이 나올 것입니다. 붉은 색의 스트링 문자열이 실제로 적용될 쿼리문이겠죠. 일단 6.0에서는 이런 방식으로 DB를 사용했었고, 2008에서는 어떻게 바뀌었는지 확인해 보겠습니다.

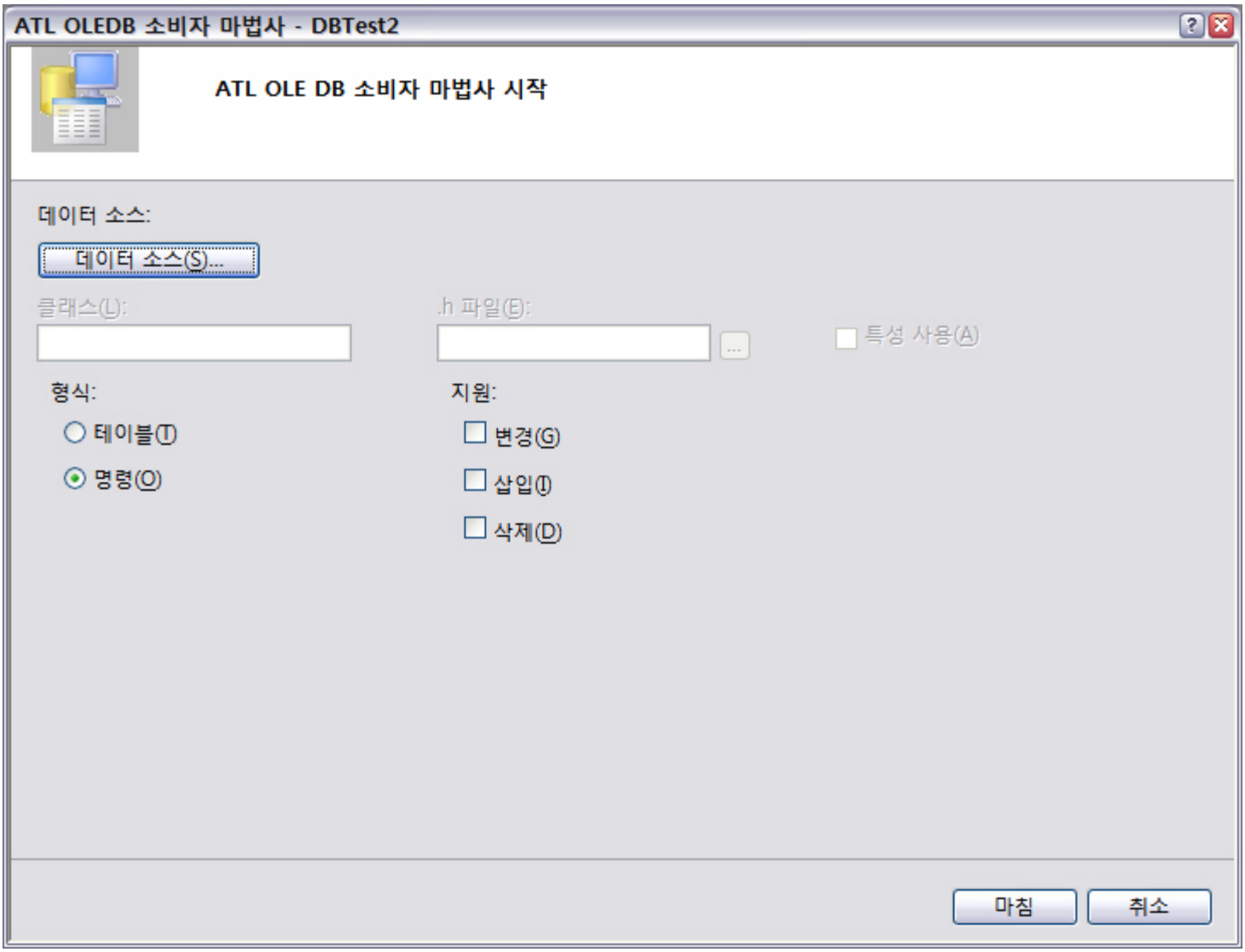
먼저 MFC 프로젝트를 하나 생성하고 클래스 뷰를 열어봅니다.



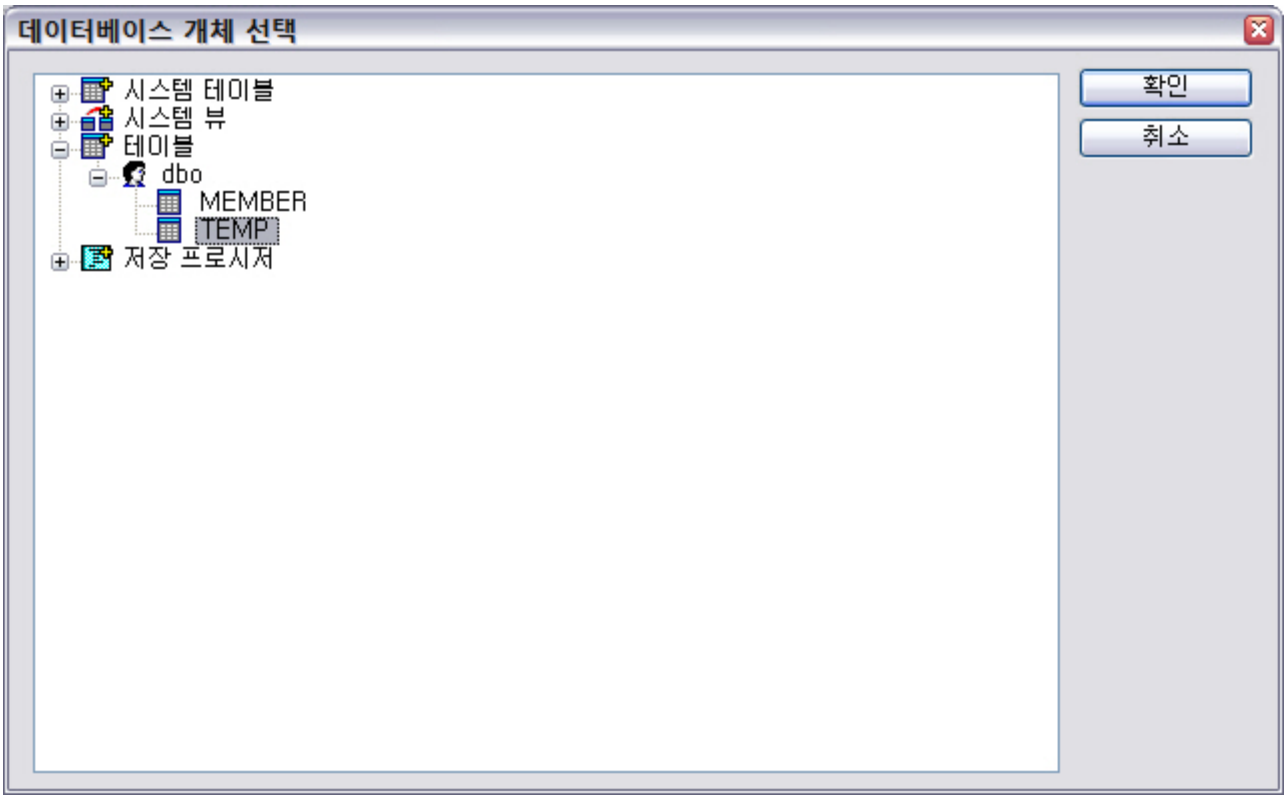
최상단의 프로젝트를 오른쪽 버튼 클릭 하셔서 추가->클래스를 선택합니다.



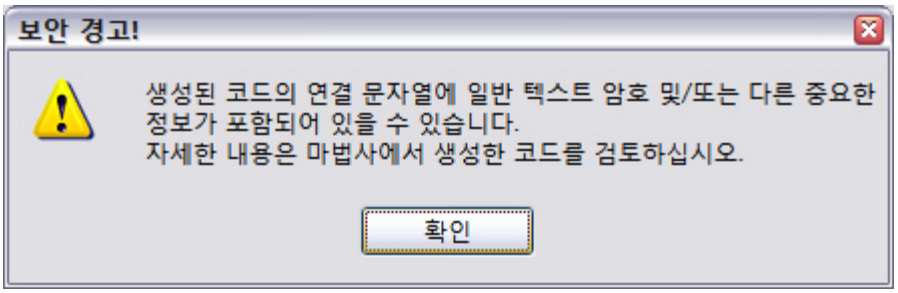
New ATL Object가 어디갔나 했더니 여기 있었습니다. 우리는 DB를 사용할 것이니까 ATL란의 ATL OLEDB 소비자를 선택합니다.



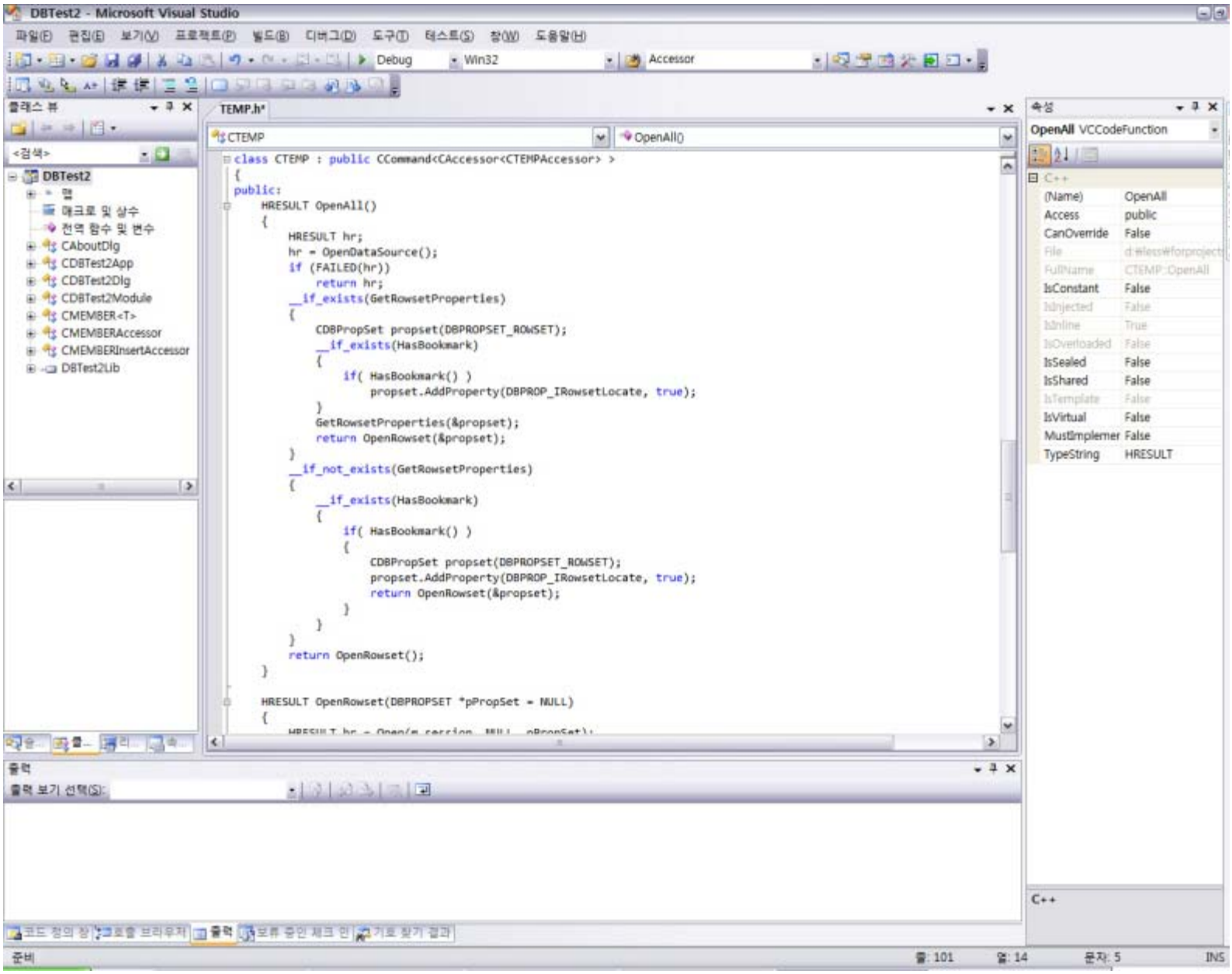
비슷비슷한 모습이죠? 여기서도 데이터 소스를 선택합니다. 그럼 위의 6.0에서의 설정 화면과 똑같은 모습이 반복됩니다. 과정은 그와 동일하고 사용할 주 테이블까지 선택을 하시면 연결이 완료됩니다.(여기서 선택하는 테이블만 우리가 사용할 수 있는 것은 아닙니다. 그냥 메인 테이블 선택한다고 생각하시면 될 것 같습니다.)



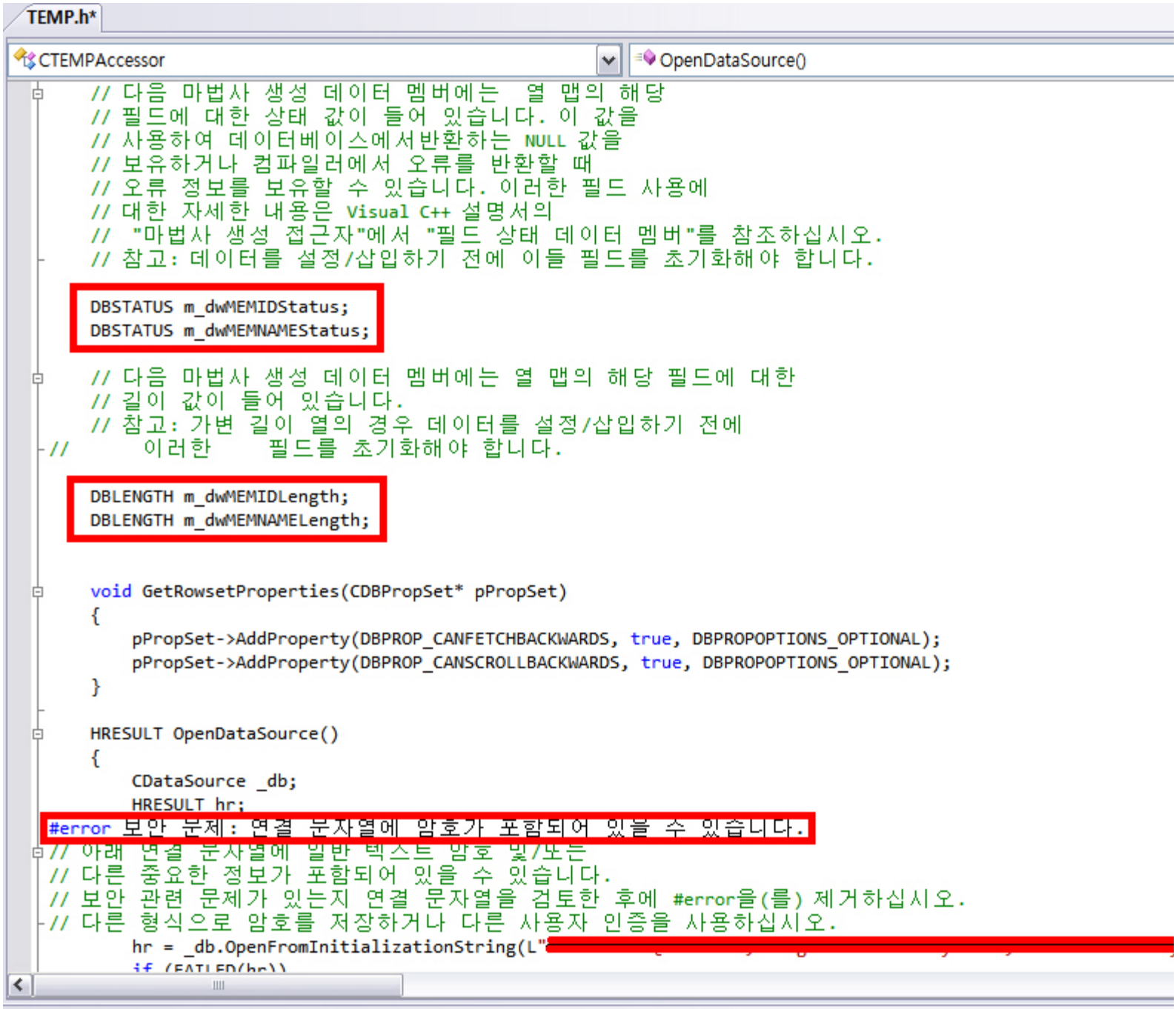
전 TEMP라는 테이블에 간단하게 ID와 이름을 저장할 수 있는 공간을 만들었습니다. 확인을 눌러서 설정을 마칩니다.



그럼 무섭게도 이런 메세지가 나타납니다. 크게 중요한 내용은 아니니 그냥 슬 넘어갑니다.



그럼 다음과 같은 소스가 만들어 집니다. 그럼 이 소스를 기반으로 쿼리를 조작하면 됩니다. 다만 6.0과 조금 다른부분이 생겼는데 그 부분을 집중적으로 보도록 하겠습니다.



먼저 아까 메세지 박스로 언급되었던 보안 문제가 있습니다. 이 보안문제는 별건 아니고 아래 OpenFromInitializationString을 통해 DB에 접속하게 되는데 그 안에 보안과 관련된 스트링이 포함되어 있어서 이걸 정말 사용할 것인지를 재차 확인하는 것입니다. 가볍게 주석처리해줍시다.

그리고 위에 보면 DBSTATUS와 DBLENGTH라는 타입의 변수들이 있는데 이는 실제로 사용할 변수들의 정보를 담고 있다고 보면 됩니다. DBSTATUS의 경우 DB가 정상이라고 그냥 정의해주면 되고 DBLENGTH는 DB에 입력될 문자열(혹은 숫자)의 길이를 입력해주면 되는 듯 합니다.(정확하진 않지만 일단은..)

TEMP.h*

CTEMPAccessor | GetRowsetProperties(CDBPropSet * pPropSet)

```
    }

    void CloseDataSource()
    {
        m_session.Close();
    }

    operator const CSession&()
    {
        return m_session;
    }

    CSession m_session;

    DEFINE_COMMAND_EX(CTEMPAccessor, L" \
    SELECT \
        MEMID, \
        MEMNAME \
        FROM dbo.TEMP")

    // 일부 공급자와 관련된 몇몇 문제점을 해결하기 위해 아래 코드에서는
    // 공급자가 보고하는 것과 다른 순서로 열을 바인딩할 수 있습니다.

    BEGIN_COLUMN_MAP(CTEMPAccessor)
        COLUMN_ENTRY_LENGTH_STATUS(1, m_MEMID, m_dwMEMIDLength, m_dwMEMIDStatus)
        COLUMN_ENTRY_LENGTH_STATUS(2, m_MEMNAME, m_dwMEMNAMELength, m_dwMEMNAMEStatus)
    END_COLUMN_MAP()
};

class CTEMP : public CCommand<CAccessor<CTEMPAccessor> >
{
public:
    HRESULT OpenAll()
    {
        HRESULT hr;
        hr = OpenDataSource();
        if (FAILED(hr))
```

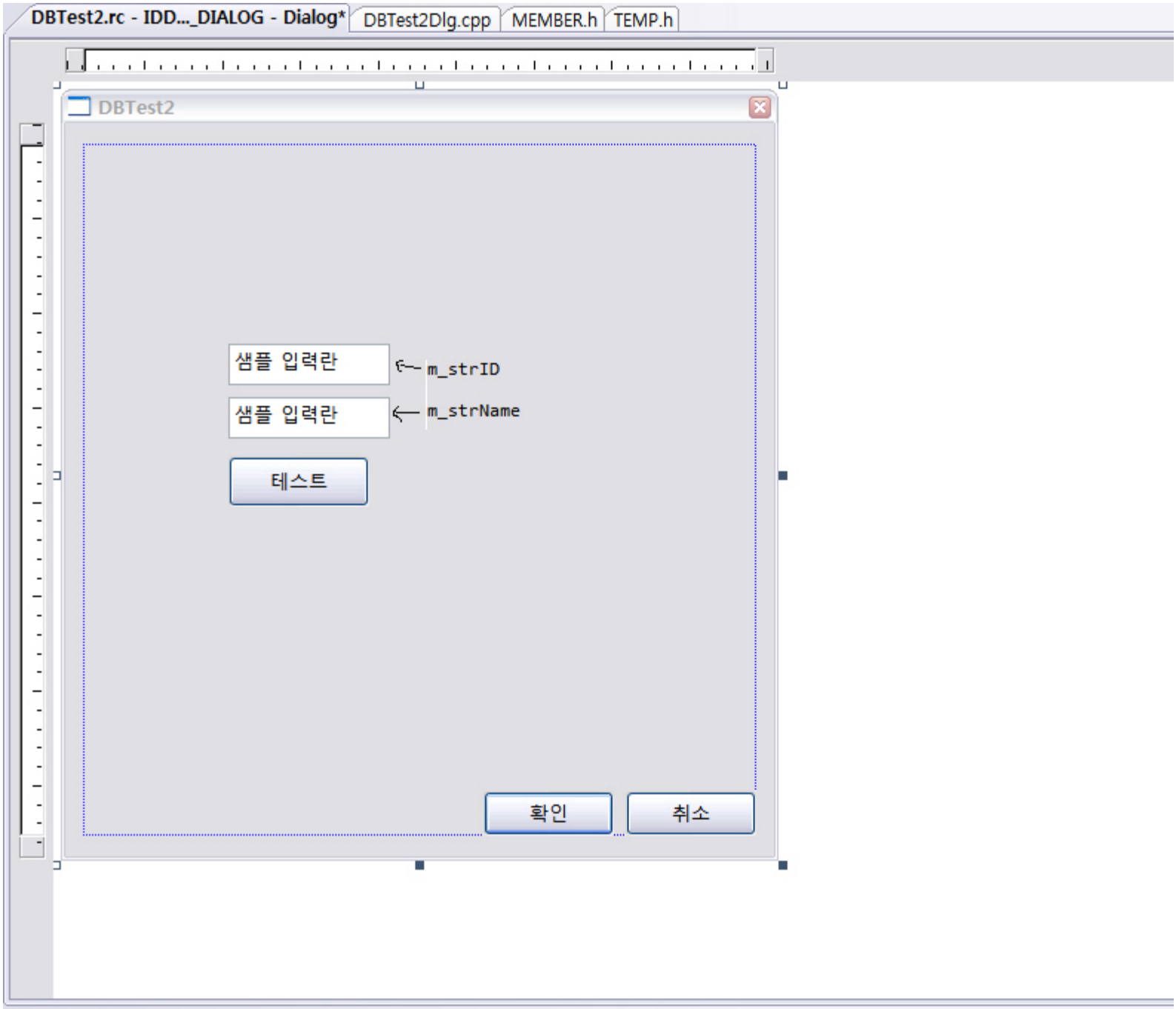
쿼리문은 DEFINE_COMMAND_EX라는 매크로를 통해 정의가 가능합니다. 하나의 쿼리를 실행하기 위해서는 하나의 접근자(Accesor)가 필요하므로 이 클래스를 양식으로 여러 쿼리를 만들어 관리할 수 있습니다.

DB로 부터 값을 받아올 데이터의 변수를 지정해주는 BEGIN_COLUMN_MAP매크로는 대체로 같습니다만 COLUMN_ENTRY가 조금 변경되었습니다. 위에서 선언되었던 DBLENGTH와 DBSTATUS의 값을 저장할 변수를 지정해주는 매크로인 COLUMN_ENTRY_LENGTH_STATUS를 사용해야 합니다. 일단 이녀석을 사용하는게 컴파일러의 의도에 맞춰 주는 것 같습니다.

```
template<class T>
class CMEMBER : public CCommand<CAccessor<T> >
```

그리고 CCommand뒤의 CAccessor뒤에 접근자 클래스가 명시가 되어있습니다만(위 그림에선 다른 클래스에 적용시켜서 CTEMP가 CMEMBER가 되어 있습니다만 융통성있게 그냥 이해하세요 ㅎㅎ) 템플릿을 이용해서 어떤 접근자도 다 받아들여 공용으로 사용할 수 있도록 하는게 편합니다. 쿼리 클래스 하나당 별도의 클래스가 하나씩 있어야한다면 그만큼 비효율적인 것도 없습니다.

그럼 이제 테스트를 위해 간단하게 다이얼로그를 하나 만듭니다.



참 간단합니다. 확인/취소는 더블클릭만해서 OnOK()와 OnCancel()만 만들어 주고, 텍스트 박스를 두개, 버튼을 하나 만듭니다. 당장 에디트 박스를 쓸 것은 아니지만 입력때도 사용하도록 m_strID와 m_strName이란 이름으로 컨트롤 변수도 만들어 줍니다.(value, CString)

DBTest2.rc - IDD..._DIALOG - Dialog* | DBTest2Dlg.cpp* | MEMBER.h | TEMP.h

CDBTest2Dlg | OnBnClickedBtnlogin()

```
HCURSOR CDBTest2Dlg::OnQueryDragIcon()
{
    return static_cast<HCURSOR>(m_hIcon);
}


void CDBTest2Dlg::OnBnClickedOk()
{
    // TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.
    OnOK();
}

void CDBTest2Dlg::OnBnClickedCancel()
{
    // TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.
    OnCancel();
}

void CDBTest2Dlg::OnBnClickedBtnlogin()
{
    // TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.

    CMEMBER<CMEMBERAccessor> Login;
    UpdateData();

    if( Login.OpenAll() == S_OK )
        if( Login.MoveNext() == S_OK )
            MessageBox( Login.m_MEMID );
```

그리고 테스트 버튼을 더블클릭하여 위와같이 코드를 작성합니다.
CMEMBER는 접속을 위한 클래스, CMEMBERAccessor는 DB접속 후 무슨 쿼리를 실행할지가 정의된 클래스라고 생각하면 쉽습니다.

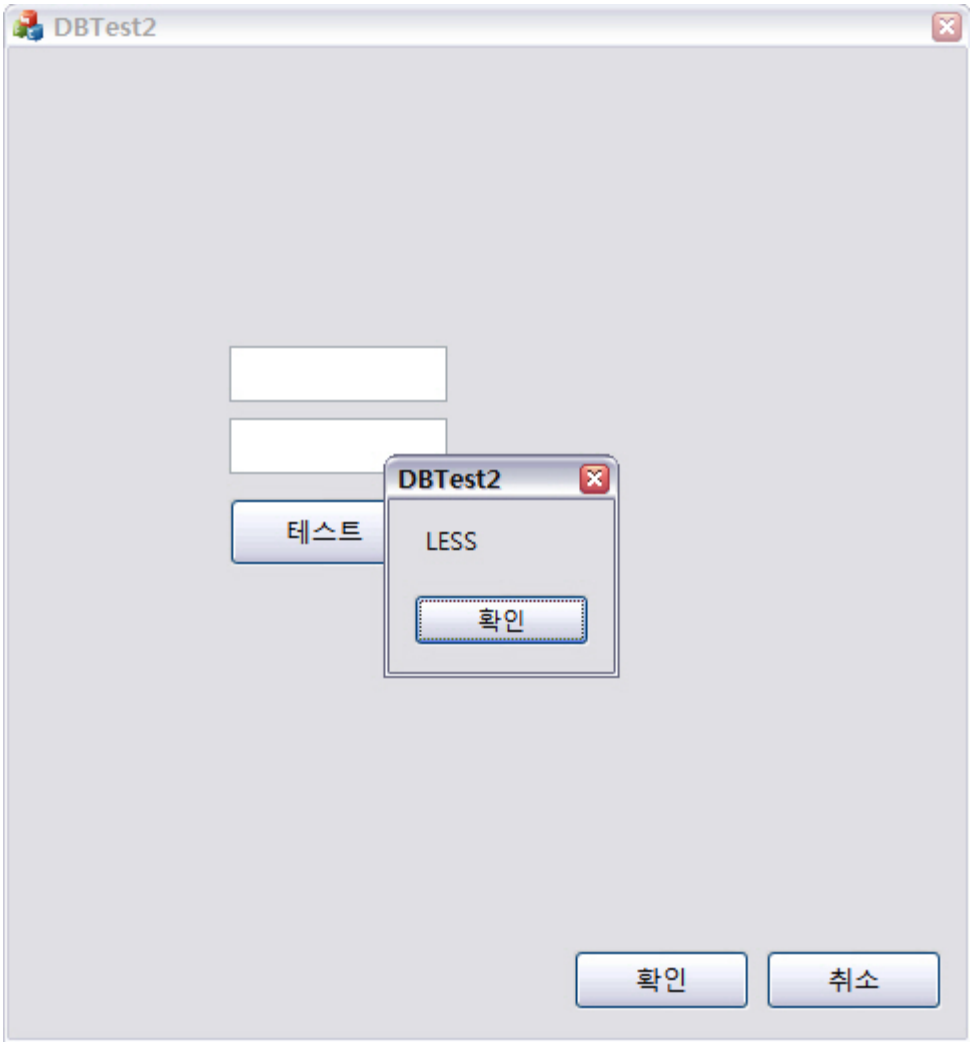
또 6.0과 차이점이 하나 있습니다만 기존에는 Open() 메소드를 사용했지만 이젠 OpenAll()이란 메소드를 사용합니다. 현재의 버전에서 Open()을 실행하게 되면 CMEMBER 클래스에는 정의된 Open() 메소드가 없어서 상위 클래스의 메소드를 호출하게되어 의도치 않은 결과를 낳게 됩니다. 실제로는 OpenAll() 메소드가 과거의 Open() 메소드의 발전형태라고 생각하시면 될 것 같습니다.

그리고 우리가 사용하고자 하는 쿼리를 CMEMBERAccessor 클래스 내부에서 다음과 같이 정의합니다.

```
DEFINE_COMMAND_EX(CMEMBERAccessor, L" \
SELECT MEMNAME FROM dbo.TEMP")
```

저는 TEMP라는 테이블의 MEMNAME의 열에서 데이터를 가져올 것입니다.

그럼 빌드 후 실행을 시켜보겠습니다. 실행 후 테스트 버튼을 클릭하면 다음과 같은 화면이 나옵니다.



출력이 잘 되었습니다.

만약 전체 리스트를 출력하고 싶다면 if(Login.MoveNext() == S_OK)를 while(Login.MoveNext() == S_OK)로 바꾸어서 Login.m_MEMID 로 접근이 가능할 것입니다.

그럼 이제 입력을 해 보겠습니다. 입력을 위해서 접근자 클래스를 하나 더 만들어 보겠습니다.

이름은 CMEMBERInsertAccessor, 코드의 변경점은 아래와 같습니다.

```
class CMEMBERInsertAccessor
{
public:
    TCHAR m_MEMID[51];
    TCHAR m_MEMNAME[51];

    DBSTATUS m_dwMEMIDStatus;
    DBSTATUS m_dwMEMNAMEStatus;

    DBLENGTH m_dwMEMIDLength;
    DBLENGTH m_dwMEMNAMELength;

    void GetRowsetProperties(CDBPro
    {
```

(중략)

```
DEFINE_COMMAND_EX(CMEMBERInsertAccessor, L" \
INSERT INTO TEMP(MEMID, MEMNAME) VALUES(?,?)")


BEGIN_PARAM_MAP(CMEMBERInsertAccessor)
    COLUMN_ENTRY_LENGTH_STATUS(1, m_MEMID, m_dwMEMIDLength, m_dwMEMIDStatus)
    COLUMN_ENTRY_LENGTH_STATUS(2, m_MEMNAME, m_dwMEMNAMELength, m_dwMEMNAMEStatus)
END_PARAM_MAP()
```

이번에는 우리가 값을 전달할 것이기에 COLUMN_MAP 대신 PARAM_MAP을 생성합니다. 안에 들어갈 내용은 동일합니다.

그리고 테스트 버튼을 클릭하였을때의 코드를 조금 수정해줍시다.

MEMBER.h | DBTest2.rc - IDD..._DIALOG - Dialog | DBTest2Dlg.cpp | TEMP.h

CDBTest2Dlg | OnBnClickedBtnlogin()

```
void CDBTest2Dlg::OnBnClickedCancel()
{
    // TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.
    OnCancel();
}

void CDBTest2Dlg::OnBnClickedBtnlogin()
{
    // TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.
/*
    CMEMBER<CMEMBERAccessor> Login;
    UpdateData();

    if( Login.OpenAll() == S_OK )
        if( Login.MoveNext() == S_OK )
            MessageBox( Login.m_MEMID );
*/

    CMEMBER<CMEMBERInsertAccessor> Insert;
    UpdateData();

    wcscpy(Insert.m_MEMID, m_strID);
    Insert.m_dwMEMIDLength = wcslen(m_strID)*2;
    Insert.m_dwMEMIDStatus = DBSTATUS_S_OK;

    wcscpy(Insert.m_MEMNAME, m_strName);
    Insert.m_dwMEMNAMELength = wcslen(m_strName)*2;
    Insert.m_dwMEMNAMEStatus = DBSTATUS_S_OK;


    if(Insert.OpenAll() == S_OK)
        MessageBox(_T("등록되었습니다"));



}
```

CMEMBERInsertAccessor 클래스로 바꾸고 이름도 Insert로 바꾸었습니다.
그리고 텍스트박스에 값을 입력했을 것이니 UpdateData() 함수를 이용해 변수에 값을 저장하였습니다.

wcscpy를 통해 Insert객체 내의 변수들에 값을 저장하였습니다. 컴파일을 하게 되면 이 wcscpy 함수의 보안 문제로 경고 메세지가 뜨겠지만 지금은 일단 넘어가겠습니다.

m_dw~Length 에는 해당하는 텍스트의 길이를 지정하기 위에 wcslen함수를 사용하였고, 곱하기 2를 하였습니다. 왜 곱하기 2를 해야하는가- 라고 물어보신다면 유니코드라는 키워드만 던져드리겠습니다.

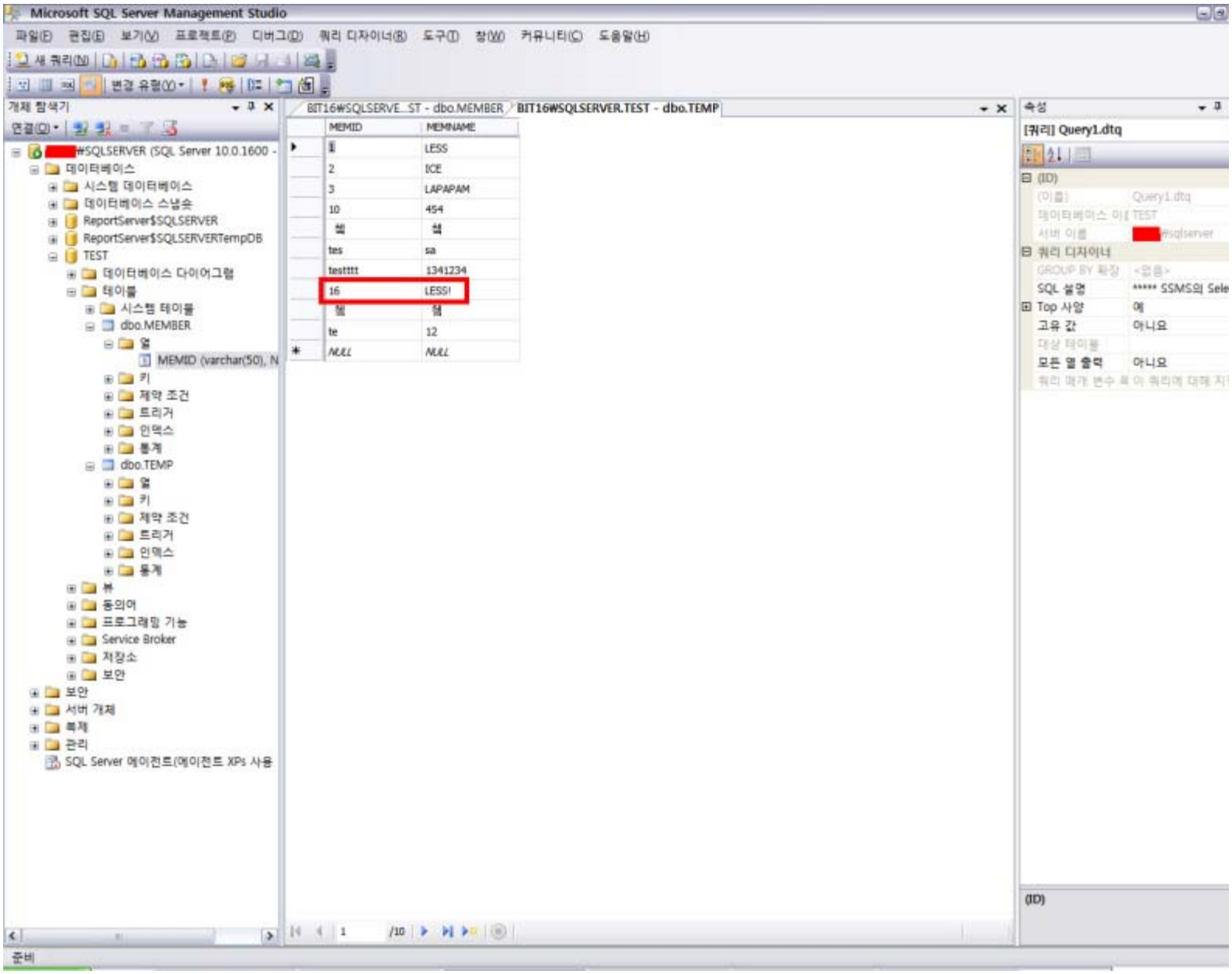
m_dw~Status 에는 DB_STATUS_S_OK 라는 값을 넣어주었는데 사실 저도 이건 잘 모르겠습니다. 책 보고 참고했을 뿐 (..) (열혈강의 Visual C++ 2008 MFC 윈도우 프로그래밍, 최호성)

어쨌든 INSERT 쿼리는 따로 값을 받아올 필요 없이 값만 던져준 다음 쿼리문만 실행하면 되기에 OpenAll만 해줘도 실행이 됩니다. S_OK의 값이 리턴되면 등록에 성공한 것입니다.

빌드하고 실행해보겠습니다.

정상적으로 등록이 되었답니다.

그럼 정말 제대로 올라갔는지 확인해 보겠습니다.



다행히도 사기친 것 같진 않습니다. 입력한 16/LESS!라는 문구가 제대로 저장이 되었습니다.

위의 SQL 테이블을 보시면 test라고 치려다가 반이 잘려나간 te 라든가 ' 쉐ㄱ'(제가 쓰고 있는 네이버 글자체가 이 글자를 표현을 못하는군요) 등은 왜 문제가 생긴채로 들어갔었을까요? 답은 문자열 처리 도중에 문제가 발생한 것이지요. 혼자 뻘짓하다가 내놓은 결과물인데 dw~LENGTH와 관련이 있지요. 물론 쉐ㄱ의 경우는 PARAM_MAP의 설정에서 문제가 생긴 케이스이고 te는 wcslen에서 문제가 발생했었습니다. 어쨌든 문제는 해결되었고 잘~ 됩니다.

이런 식으로 조금 더 응용하면 수정, 삭제는 일도 아니겠지요.

어쨌든 이정도로 VC++ 2008에서 MSSQL을 사용하는 방법을 간단하게 살펴보았습니다.

제가 검색능력이 부족해서 이런 레퍼런스를 찾지 못한 것인지.. 아님 아직 2008이 전체적으로 확대 보급이 안되서 레퍼런스가 부족한건진 모르겠습니다만 어쨌든 6.0으로 공부하시다가 2008로 넘어오면서 고생 하시는 분들이 계시다면 함께 공부하자고 말씀드리면서 이 글을 마치도록 하겠습니다.