C++Builder ADO Programming (4) - ConnectionString Core

지난 번 강의에서 우리의 레밍은 ADO 라는 마법의 빗자루를 가지고 고달프고 거친, 가혹한 Database의 세계를 모험하기 위하여 가장 기본이 되는 주문을 배웠다. 그것은 다름이 아닌 ConnectionString이었으며 "열려라, 데이타베이수!"의 거의 90% 이상을 차지하는 막강한 대 주문이었다. 하지만 우리의 레밍은 그 약삭빠른 알리바바가 아니므로 흉악하고 탐욕적인 40인의 도적과 같은 각종 데이터베이스 연결 에러들을 물리치고 데이터를 가지기 위해선 더 많은 것을 알아야 한다. 그것은 연결에 관련이 있는 TADOConnection Component의 속성들과 여러 메소드들이며 이번 강의는 그것을 다룰 것이다.

DefaultDatabase 속성

DefaultDatabase 속성은 연결이 열렸을 때 연결 될 데이터베이스를 지정하는 것이다. 이 속성은 읽기/쓰기 속성이며, 이미 TADOConnection 객체에 의해 열린 연결이 존재하는 동안에도 이 속성의 값을 변경할 수 있다. 하지만 일단 한 번 설정된 뒤에는 빈 문자열로 설정할 수는 없다. 이 속성의 이점은 2개 이상의 데이터베이스를 사용하는 어플리케이션을 작성할 때 유용하며 일단 기존의 얻은 연결을 다시 설정할 필요가 없이 이 속성만을 설정함으로 원하는 결과를 얻을 수 있다는 점이다. 유의할 점이 있다면 모든 OLEDB 공급자들이 이 속성을 사용할 수 있다는 것이 아니라는 점이다. OLEDB 공급자들 중에는 데이터베이스가 아닌 형태의 데이터에 대한 공급자들이나 기본 데이터베이스라는 개념을 구현하지 않은 데이터베이스 시스템에 대한 공급자들 중에는 이 속성을 지원하지 않는 것들이 있다. (ORACLE의 경우가 대표적인데 ORACLE은 기본적으로 사용자 위주의 스키마를 가진 RDBMS 이기 때문이다. 이 기본 데이터베이스라는 말은 MS-SQL Server와 같은 파일 위주의 스키마를 가진 RDBMS 위주의 개념인 것이다.)

Provider 속성

Provider 속성은 접근하고자 하는 데이터 원본의 종류를 결정한다. 코드를 통해 연결을 동적으로 얻을 경우 이 속성에 어떤 값을 설정했다면 ConnectionString에 Provider 항목을 지정할 필요가 없다. 만약 ConnectionString에 공급자 정보를 입력하지 않았다면 연결이 열릴 때 ConnectionString에는 현재 공급자인 Provider 속성에 설정되어 있는 값이 지정된 것으로 간주한다. 반대로 ConnectionString에만 공급자 정보가 들어있고 Provider 속성이 설정되지 않은 상태에서 연결을 열었을 경우 ConnectionString에 들어있는 공급자 정보가 Provider에 설정된다. IDE를 통해서 ConnectionString을 얻을 경우엔 ConnectionString을 생성시 자동적으로 그에 맞는 데이터 공급자가 Provider 속성에 설정된다. 이 속성은 다양한 값을 가지는데 ConnectionString에서 알아보지 않은 다른 공급자 값들은 이 후의 강의에서 소개가 될 것이다.

State 속성

연결이 열려있는지 아니면 닫혀 있는지를 가리킨다. 이 속성은 당연히 읽기 전용이며 TObjectStates 열거 형의 다음의 다섯 가지 값만을 갖는다. 연결 객체의 상태에 대해서 알아보고 싶을 경우 이 속성을 체크하면 될 것이다. stExecuting과 stFetching의 경우 연결객체를 가지고 쿼리 명령을 실행시키는 경우, 상태를 점검할 수 있는데 이 방법은 그지 추천할 만한 방법이 아니다.

stClosed	연결객체가 비활성화 되어있고 데이터베이스에 연결되지 않았다.
stOpen	연결객체가 비활성화 되어 있으나 데이터베이스에 연결되어 있다.
stConnecting	연결객체가 데이터베이스에 연결되는 과정중이다.
stExecuting	연결객체가 실행중이다.
stFetching	연결객체가 데이터베이스로부터 데이터를 가져오는 중이다.

Version 속성

Version 속성은 ADO 버전 번호를 문자열 형태로 돌려주는 읽기 전용 속성이다.

Mode 속성

이 속성은 연결에 대한 읽기/쓰기 허용방식을 결정한다. 이 속성에 넣을 수 있는 값은 TConnectMode 열거형의 다음 8가지 값이다.

cmUnknown	허용 방식이 결정될 수 없거나 아직 설정되지 않았다. (기본값)
cmRead	읽기 전용 연결
cmWrite	쓰기 전용 연결
cmReadWrite	읽기/쓰기 연결
cmShareDenyRead	다른 사용자가 읽기 허용으로 연결을 열지 못하게 함
cmShareDenyWrite	다른 사용자가 쓰기 허용으로 연결을 열지 못하게 함
cmShareExclusive	다른 사용자가 연결을 열지 못하게 함
cmShareDenyNone	다른 사용자가 어떠한 허용 방식으로도 연결을 열지 못하게 함.

연결이 열려진 상태에서는 이 속성은 읽기 전용이 된다. 연결이 닫히면 다시 읽기/쓰기가 가능해지므로 다음 연결에 대한 허용 모드를 변경할 수 있다. 코드를 통한 연결의 동적생성시 연결을 열기 전에 반드시 이 속성을 설정해 줘야 되는 것은 아니다. 이 속성을 명시적으로 설정하지 않은 경우 연결이 열릴 때 데이터 원본 자체에 설정되어 있는 사용자의 접근 권한이 이 속성에 설정된다.

KeepConnection 속성

어떤 DataSet이 열려져 있지 않더라도 어플리케이션이 데이터베이스에 대한 연결을 유지 할 지에 대해 결정하는 속성이다. 원격지 데이터베이스 서버에 대한 연결과 DataSet을 빈번하게 열고 닫는 어플리케이션에서 네트웍 트래픽을 줄이고 어플리케이션의 스피드를 올리며 연결이 재 설정될 때 마다 데이터베이스에 로그인하는 것을 방지하기 위해 이 속성을 true로 설정하는 것이 좋다. 반대로 이 속성이 false로 설정되면 열려진 DataSet이 없을 때 연결이 드롭 된다. 연결이 드롭 되면 연결에 할당된 시스템 자원들이 해제 되지만 후에 DataSet이 하나라도 열려져 데이터베이스를 사용하게 된다면 연결은 재 설정되고 초기화 되어져야 한다. 이것은 분명히 추가적인 네트웍 트래픽과 시간을 요구한다는 것을 의미한다.

ConnectionTimeout 속성

이 속성은 이름 그대로 데이터 원본으로 연결을 시도하는 데 소비할 수 있는 최대 시간을 뜻한다. 시간 단위는 초 단위이며 그 시간이 지나도 연결이 열리지 않으면 에러가 발생한다. 밑의 CommandTimeout 속성과 마찬 가지로 이 속성에 0을 넣으면 시간 제한을 두지 않는다는 뜻이 된다. 기본 값으로 15초로 설정되어 있으며 연결이 열린 후에는 읽기 전용이 된다.

CommandTimeout 속성

이 속성은 하나의 명령이 실행될 수 있는 최대 시간을 결정한다. 시간 단위는 초 단위이며 어떤 명령이 이 속성의 기본 값인 30초 이상 걸릴 가능성이 있다면 그런 명령을 수행하기 전에 이 속성을 좀 더 큰 값으로 설정해야 제대로 된 결과를 얻을 수 있다. 지정된 시간 안에 명령이 완료되지 않으면 그 명령은 취소되며 에러가 발생한다. 시간 제한을 아예 없애려면 이 속성 값을 0으로 설정하면 된다. 이 속성은 TADOConnection 객체를 통해 직접 수행되는 명령에만 영향을 미친다. TADOCommand 객체에도 이 속성이 있으며 만약 그 Command 객체가 Connection 객체에 연결이 되어 있다 해도 Command 객체의 CommandTimeout 속성이 우선한다.

ConnectOptions 속성

TADOConnection에 의해 설정된 연결이 동기적인지 비동기적인지를 결정하는 속성이다. Default 값은 동기적인 연결 값인 coConnectUnsuspected 이다. 대부분의 어플리케이션에서 사용되는 연결은 동기적인 연결이다.

비동기적인 연결 값인 coAsyncConnect 는 대용량의 데이터 처리가 일어나거나 데이터베이스가 커져 튜닝이 필요한 느린 서버상에서 사용할 경우 성능을 보충하는데 사용된다.

LoginPrompt 속성

새로운 Connection이 열리기 전에 로그인 창이 뜰지를 결정할 수 있다. 이 속성이 true 이면 연결이 설정되기 전에 사용자의 이름과 패스워드를 입력하는 로그인 창이 제공된다. 로그인 창은 TDatabase Component와 마찬가지로 OnLogin Event Handler가 제공된다면 OnLogin Event에서 입력된 Event Handler가 제공되지 않았을 경우 BeforeConnect Event 다음 AfterConnect Event 전에 활성화 된다. 정보가 틀릴 경우 Connection은 실패하게 된다. 만약 이 속성을 false로 설정할 경우 OnLogin Event가 발생하지 않고 로그인 창이 뜨지 않는다. 이것은 ConnectionString에서 사용자 정보를 제공해야 한다는 것을 의미한다.

CursorLocation 속성

커서란 데이터베이스나 레코드 셋 안의 한 레코드를 가리키는 포인터이다. 커서는 엄밀히 말해 Recordset 객체의 영역에 속하는 것이므로 이번 Connection 객체의 장에서는 Connection 객체의 CursorLocation 부분에 대해서만 이야기하고 커서에 대한 자세한 이야기들은 TADODataSet 부분의 강의에서 하도록 하겠다 미리 말해두지만 이 속성은 대단히 중요하다. Connection 객체의 이 속성은 커서 엔진의 위치를 결정한다. 이 속성의 값들은 다음과 같다.

★ clUseClient – 클라이언트 쪽 커서를 사용한다.
Default 값으로 더 많은 유연성을 제공한다. 모든 데이터를 로컬 머신쪽으로 가져오며 그리고 그 쪽에서 조작한다. 이 말은 서버가 데이터 정렬과 재정렬 그리고 추가적인 필터링과 같은 조작을 지원하지 않도록 한다는 의미이다. SQL 문은 서버에서 실행되고 그 결과 집합은 로컬 커서로 가져오게 된다. 그러므로 Where 절과 같은 조건 문으로 제한된 다소 적은 결과 집합을 가져오는데 유용한 속성이다.

★ clUseServer – 서버 쪽 커서를 사용한다.
클라이언트 쪽 커서 보다는 다소 유연성이 적다. 하지만 주로 큰 결과집합을 가져올 때 유용하며 많은 이점이 있다. 결과 집합의 크기가 클라이언트 쪽 커서를 생성하는데 필요한 디스크 용량을 훨씬 넘어설 때 서버 쪽 커서를 사용하는 것이 필요해 진다. 또 한가지는 많은 서버들이 한쪽 방향만의 커서를 지원한다는 것인데 이 말은 반환된 결과집합내의 데이터 셋에서 레코드 포인터를 뒤로 움직이는 것을 차단한다는 것이다. (전진 전용 커서라는 이야기이다) 더 자세한 커서에 대한 내용은 TADODataSet을 다루는 이후의 강의에서 다루도록 한다.

DataSets 속성

Connection Component에 연결된 모든 데이터 셋들의 인덱스화 된 배열을 제공하는 속성이다. 따라서 Connection Component의 연결되어 연결 풀링의 혜택을 받는 모든 데이터 셋을 이 속성을 사용하여 액세스 할 수 있다. 다음에 나오는 DataSetCount 속성과 함께 사용되는 예를 보자.

```
for (int i = 0; i < objConn->DataSetCount; i++)
  ListBox1->Items->Add(objConn->DataSets[i]->Name);
```

DataSetCount 속성

이 속성은 Connection Component와 연결된 데이터 셋들의 개수를 나타낸다. 대개 위의 DataSets 속성과 함께 사용된다.

Commands 속성

DataSets 속성과 비슷한 형태로 Connection Component에 연결된 모든 ADO 커맨드들의 인덱스화 된 배열을 제공한다. 역시 다음에 나오는 CommandCount 속성과 함께 사용된다. 이 속성은 유용한데 여러 종류의

커맨드를 설정하고 각각을 Connection Component와 연결 한 후에 각 명령들을 실행시킬 수 있다. 명령들을 연속적으로 쏘는 리발버 대포의 기능을 하는데 포탄이나 총알은 뒤에 나올 것이다. 더 자세한 사항은 이후의 TADOCommand를 다룬 강의에서 살펴보도록 한다.

```
for (int i = 0; i < objConn->CommandCount; i++)
{
  // 각 커맨드에 맞는 세부 설정과 조건을 지정한다.

  objConn->Commands[i]->Execute();
  // 커맨드를 실행하거나 반환되는 처리를 한다.

  // 커맨드를 실행후에 해야될 조작들을 정의한다.
}
```

CommandCount 속성

이 속성은 Connection Component와 연결된 Command Component의 수를 나타낸다. 주로 위의 Commands 속성과 함께 사용된다.

OpenSchema 메소드

말 그대로 데이터베이스 스키마를 열어서 스키마의 정보 데이터 셋을 얻는 메소드이다. 굉장히 매력적인 메소드이며 유용하다.

```
void __fastcall OpenSchema(const TSchemaInfo Schema,
                           const OleVariant &Restrictions,
                           const OleVariant &SchemaID,
                           TADODataSet* DataSet);
```

중요하게 봐야 할 것은 첫 번째, 네 번째 인자이다. 첫번째는 말 그대로 관련 데이터 공급자의 어느 스키마 정보를 가져올지를 결정하는 인자로 그 열거형들을 아래에서 볼 수 있다. 네 번째는 인자는 스키마 정보를 가져올 TADODataSet을 지정한다. 아래의 예를 보자.

```
TADODataSet *objRsSchema;

DataSource1->DataSet = objRsSchema;
DBGrid1->DataSource = DataSource1;

objConn->OpenSchema(siTables, EmptyParam, EmptyParam, objRsSchema);
// 테이블에 관계된 스키마를 얻는다.

while ( !objRsSchema->Eof )
{
  Ls->Items->Add(objRsSchema->FieldByName("TABLE_NAME")->AsString);
  objRsSchema->Next();
}

// 그에 맞는 조작을 한다.

objConn->OpenSchema(siProcedures, EmptyParam, EmptyParam, objRsSchema);
// 저장 프로시저에 관계된 스키마를 얻는다.
// 그에 맞는 조작을 한다.

objConn->OpenSchema(siIndexes, EmptyParam, EmptyParam, objRsSchema);
// 인덱스에 관계된 스키마를 얻는다.
// 그에 맞는 조작을 한다.
```

2번째, 3번째 인자는 OleVariant 형으로 제한조건과 GUID 형태로 된 SchemaID 상수를 넣어 줘야 되나 보통 위의 필자의 코드에서 보다시피 아무 값도 넘기지 않는 EmptyParam을 사용한다. 이 인자는 COM에서 선택

적인 인자가 사용되지 않을 때 – 아무 인자도 건네 주고 싶지 않을 때 쓰인다. 아래는 1 번째 인자인 스키마 정보의 열거형과 그에 대한 설명, 지원하는 드라이버들의 목록이다. 필자의 소스코드에서 보듯이 얻어진 레코드 셋의 필드 명에 대해선 도움말을 참조하기 바란다. 걍 그리드에 뿌려놓으면 도움말에 있는 필드 명보다 훨씬 많이 기 나오기 때문에 그리드에 나오는 필드 명을 보는 게 낫다.

Schema 정보	설명	지원하는 드라이버들
siAsserts	카탈로그에 정의된 Assertions	SQL, ODBC Jet, ODBC SQL
siCatalogs	데이터베이스의 카탈로그의 물리적 특성 SQL Server – 데이터베이스 Access – 현재 데이터베이스	SQL, ODBC Jet, ODBC SQL
siCharacterSets	카탈로그가 지원하는 문자셋들	
siCollations	카탈로그의 데이터 정렬방식	
siColumns	테이블들의 컬럼들.	SQL, ODBC Jet, ODBC SQL
siCheckConstraints	컬럼들에 허용되는 유효한 값들.	SQL, ODBC SQL
siConstraintColumnUsage	참조 무결성 조건, 유일성 조건, CHECK 조건에 쓰이는 컬럼들	
siConstraintTableUsage	참조 무결성 조건, 유일성 조건, CHECK 조건에 쓰이는 테이블들.	
siKeyColumnUsage	카탈로그 내의 키 컬럼 들과 테이블들의 이름	
siReferentialConstraints	카탈로그의 참조 무결성 조건들	
siTableConstraints	참조 테이블 조건들	SQL
siColumnsDomainUsage	무결성을 위해 도메인을 이용하는 필드들.	
siIndexes	카탈로그 안의 색인들의 목록	SQL, ODBC Jet, ODBC SQL
siColumnPrivileges	주어진 사용자를 위한 테이블 컬럼 들에 대한 특권	SQL, ODBC SQL
siTablePrivileges	테이블들의 사용자 특권들	SQL
siUsagePrivileges	한 사용자에게 사용 가능한 용법 특권들	
siProcedures	저장 프로시저들 혹은 질의들.	SQL, ODBC Jet, ODBC SQL
siSchemata	한 특정한 사용자가 소유한 스키마들	SQL, ODBC SQL
siSQLLanguages	카탈로그가 지원하는 일치 수준들과 기타 옵션들.	
siStatistics	카탈로그 통계 정보	SQL
siTables	한 카탈로그 내의 테이블들	SQL, ODBC Jet, ODBC SQL
siTranslations	카탈로그가 지원하는 문자 해석들	
siProviderTypes	공급자가 지원하는 데이터 형들	SQL, ODBC Jet, ODBC SQL
siViews	카탈로그 안의 뷰들	
siViewColumnUsage	뷰들 안에서 쓰이는 컬럼들	
siViewTableUsage	뷰들 안에서 쓰이는 테이블들	
siProcedureParameters	저장 프로시저들의 인자들	SQL, ODBC Jet, ODBC SQL
siForeignKeys	참조 무결성 점검에 쓰이는 외부 키 컬럼들.	SQL, ODBC SQL

siPrimaryKeys	카탈로그 내의 기본 키들과 테이블들.	SQL, ODBC SQL
siProcedureColumns	프로시저들 내에 쓰이는 필드들.	ODBC Jet

이 메소드는 주로 SM 분야에서 백도어나 유지보수 도구, 분석도구 같은 것을 만들 때 유용할 것이다. 그러나 많은 스키마 정보 중에서도 제일 중요하고 비중 있게 사용되는 것은 역시 테이블들과 저장 프로시저들이다. --- 물론 위의 표와 소스에서 보듯이 테이블들과 저장 프로시저들은 스키마 정보인 siTables와 siProcedures를 통해 얻을 수 있다. 하지만 데이터 셋이 꼭 필요하지 않을 때도 있다. 이것은 프로그램의 크기와 직접적인 관련이 있다. --- 특히 이 데이터베이스의 객체들의 이름이 중요한데 --- 앞에서 필자가 리발버 대포를 언급한 적이 있는데 이 객체들은 거기에 장전될 대포알이나 총알이 될 것이다 --- 이것을 위해 TADOConnection 객체는 OpenSchema 메소드 이외에 3가지 아주 유용한 메서드를 제공한다.

GetProcedureNames 메소드

말 그대로 데이터베이스 내에 있는 저장 프로시저들의 이름을 가진 스트링 리스트를 만들어 낸다. 원형은 다음과 같다.

```
void __fastcall GetProcedureNames(Classes::TStrings* List);
```

GetTableNames 메소드

이 메소드도 말 그대로 데이터베이스 내에 있는 테이블들의 이름을 가진 스트링 리스트를 만들어 낸다. 2 번째 인자는 시스템 테이블을 포함할 것인지의 여부를 결정한다. 원형은 다음과 같다.

```
void __fastcall GetTableNames(Classes::TStrings* List,
                              bool SystemTables);
```

여기서 한가지 주의 깊게 볼 사항은 이 메소드가 만들어내는 리스트에 뷰도 포함 된다는 사실이다. 그러므로 데이터베이스 객체를 만들 때 테이블과 뷰의 명확한 구분을 짓는 것이 좋다.

GetFieldNames 메소드

이 메소드는 테이블의 이름과 스트링 리스트를 받아서 그 테이블의 필드 명으로 리스트를 채운다. 메소드의 원형은 다음과 같다.

```
void __fastcall GetFieldNames(const AnsiString TableName,
                              Classes::TStrings* List);
```

아래의 코드는 지금까지의 메소드 3 개의 사용 예이다. 각각 연결 객체를 통해 프로시저들의 리스트, 테이블들의 리스트, 필드들의 리스트를 얻을 수 있다.

```
TStringList *ProcedureList = new TStringList();
TStringList *TableList = new TStringList();
TStringList *FieldList = new TStringList();

String path = "D:₩₩Develop Project₩₩Dog Farm₩₩";

objConn->GetProcedureNames(ProcedureList);
ProcedureList->SaveToFile(path + "ProcLists.txt");

objConn->GetTableNames(TableList, false);
TableList->SaveToFile(path + "TableLists.txt");

for ( int i = 0 ; i < TableList->Count ; i++ )
{
  objConn->GetFieldNames(TableList->Strings[i], FieldList);
  FieldList->SaveToFile(path + TableList->Strings[i] + ".txt");
  FieldList->Clear();
}

delete FieldList, TableList, ProcedureList;
```

Execute 메소드

Command 객체를 명시적으로 생성하지 않고 Connection 객체만으로도 데이터 공급자에 대해 SQL 질의 문을 수행할 수 있으며 --- 단 데이터를 돌려주지 않는 작업만, select 질의 문이나 레코드 셋을 반환하는 저장 프로시저는 여기서 제외된다. --- 레코드 셋을 생성할 수도 있다.

Connection 객체의 Execute 메소드는 데이터를 돌려주지 않는 어떠한 명령을 수행하거나 레코드 셋을 생성하는데 쓰인다. --- 단 레코드 셋을 생성하는 경우는 별로 쓰이지 않을 것이고 사용하지 말 것을 추천한다. 굳이 TCustomADODataSet에서 파생된 Component들을 놔두고 Connection 객체로 레코드 셋을 생성할 이유가 없다.
메소드의 원형엔 2가지가 있으며 자세히 살펴보자.

```
_di__Recordset __fastcall Execute(const WideString CommandText,
                                  const TCommandType CommandType,
                                  const TExecuteOptions ExecuteOptions);

void __fastcall Execute(const WideString CommandText,
                        int &RecordsAffected,
                        const TExecuteOptions ExecuteOptions);
```

위의 원형을 보면 리턴 값이 서로 다른 2 개의 메소드가 있음을 볼 수 있다. 첫번째 메소드는 보는 것과 같이 델파이 인터페이스로 된 레코드 셋을 리턴 하는데 TADODataSet의 Recordset 속성으로 받을 수 있다. 이 속성은 VCL Component 가 아닌 직접 ADO 레코드 셋을 조작하는 방법으로 다양한 접근법을 제공하려는 배려로 볼 수 있다. 그러나 여러분이 ADO SDK나 ADO Recordset에 대해 잘 모른다면 굳이 이 메소드를 통해 레코드 셋을 얻을 필요가 없다. TADODataSet의 TDataSet으로부터 물려받은 친근한 속성들과 메소드를 사용하기를 적극 추천한다. 이 메서드의 2 번째 인자인 CommandType에 대해서는 TADOCommand와 TADODataSet을 다룬 이후의 강의에서 자세하게 알아보자.

각 메서드의 1 번째 인자인 CommandText는 수행하고자 하는 명령문을 뜻하는데 원래 여기에는 테이블이름, SQL 문장, 저장 프로시저 이름, Command 객체나 파일등을 지정할 수 있는데 2번째 메서드의 경우 데이터를 돌려주지 않는 작업의 명령문이나 객체만이 가능하다. 2번째 정수형 인자인 RecordAffected는 말 그대로 명령의 실행에 의해서 영향을 받은 레코드의 수를 나타낸다. 이 인자에 어떤 변수를 지정해서 메소드를 호출하고 나면 그 변수에 레코드 수가 설정된다. 만일 공급자가 영향을 받은 레코드 개수를 결정할 수 없거나

비동기적 작업을 지시한 경우라면 이 인자는 -1을 돌려준다. 뒤에서 나오지만 실행옵션을 통해서 비동기적 작업을 지시하면 그 작업이 완료되기 전에 실행의 제어권이 돌아오므로 작업에 의해 영향을 받은 레코드 개수가 정확하지 않을 수 있다. 다음의 예는 Connection 객체의 Execute 메소드를 통해 SQL을 수행하는 것을 보여 주고 있다.

```
int RecCount;

String Order1, Order2;

Order1 = "update dogs set alias = '파트라슈' where master = '네로'";
Order2 = "delete from dogs where name = '^0^n'";

objConn->Execute(WideString(Order1),
                 RecCount,
                 TExecuteOptions() << eoExecuteNoRecords);

ShowMessage(IntToStr(RecCount) + "마리가 파트라슈 되었슴다!");

objConn->Execute(WideString(Order2),
                 RecCount,
                 TExecuteOptions() << eoExecuteNoRecords);

ShowMessage("개" + IntToStr(RecCount) + " 마리가 도살되었슴다!");
```

위의 코드에서 수정 및 삭제된 레코드 개수가 RecCount 변수에 설정되며 그 변수의 값을 출력하는 등 써먹을 수 있다. 아래는 3번째 인자인 실행옵션으로 TExecueOptions 열거형 값을 가진다.

Execute Options	의미
eoAsyncExecute	명령이 비동기적으로 실행된다.
eoAsyncFetch	데이터를 비동기적으로 가져온다.
eoAsyncFetchNonBlocking	데이터를 비동기적, 비블로킹 방식으로 가져온다.
eoExecuteNoRecords	데이터를 돌려주지 않는 명령에 쓰인다.

SQL 질의뿐만 아니라 저장 프로시저도 Connection 객체로 호출할 수 있다. SQL 서버 나라에 다음과 같은 저장 프로시저가 살고 있다고 하자.

```
CREATE PROC procAddAuthor

@au_id char(11),
@au_lname varchar(40),
@au_fname varchar(20),
@phone char(14),
@contract bit

AS

INSERT authors (au_id, au_lname, au_fname, phone, contract)
VALUES (@au_id, @au_lname, @au_fname, @phone, @contract)

GO
```

간단한 인서트 문으로 구성되어 있지만 서버 상에서 미리 문법 체크와 컴파일이 완료된 상태이며 메모리 캐시내에 저장되어 있을 수도 있기 때문에 어플리케이션에서 직접 날리는 SQL 문보다 훨씬 성능상의 이점을 볼 수 있다. 위의 한 예에서 가능성을 보여주기 위해 SQL문을 Execute로 실행시켰지만 필자는 개인적으로 어플리케이션에서 SQL문을 날리는 것을 매우 싫어한다. 자 그럼 우리의 레밍이 어떻게 이 데이터베이스 객체를 이용할 것인가를 알아보자.

```
String Order;
int RecCount;

Order  = "EXEC procAddAuthor ";
Order += "'" + Au_ID->Text + "',";
Order += "'" + Au_LName->Text + "',";
Order += "'" + Au_FName->Text + "',";
Order += "'" + Au_Phone->Text + "',";
Order += (Au_Contract->Checked ? "1" : "0");

objConn->Execute(WideString(Order),
                 RecCount,
                 TExecuteOptions() << eoExecuteNoRecords);

ShowMessage(IntToStr(RecCount) + " 명이 추가되었습다!");
```

뭐 별 다른 것은 없을 것이다. 첫 번째 인자인 CommandText를 어떻게 DML이나 DDL로 바꾸느냐의 차이일 것이다. 이런 데이터베이스 객체들에 관계된 명령의 수행에 대한 자세한 내용은 앞으로 다룰 Command 객체나 Recordset 객체의 장에서 광범위하게 살펴볼 것이다.

Cancel 메소드

Cancel 메소드는 Execute나 Open에 의해 시작된 비동기적 작업을 취소한다. 연결이 비동기적으로 열리거나 명령이 비동기적으로 실행될 경우 이 메소드를 통해 그것을 취소 시킬 수 있다.

이번 강의는 여기 까지 이다. 눈치가 빠른 사람이면 아직 다룰 메소드나 속성이 남아 있다는 것을 알 것이다. 그것은 또 다른 주제들로 묶어볼 수 있으므로 편의상 다음 강의에서 다룰 것이다. 다음 강의는 ADO의 트랜잭션과 그것과 관련된 속성과 메소드들, 그리고 Connection 객체의 이벤트들, 에러처리와 Errors 컬렉션들을 다룰 것이다.

Mortalpain