Dynamic Query Class를 이용한 효율적인 쿼리 작성

신형기
고려대학교 컴퓨터정보통신 대학원 소프웨어공학과 재학중
email : *shk101@msn.com*

Writing Efficient Query using Dynamic Query Class

Shin Hyung Ki

요 약

프로그래밍 코드에서 쿼리문(INSERT, UPDATE,DELETE)을 작성할 때 주로 문자열을 조합해서 쿼리문을 작성하게 되는데, 이 방법은 일정한 체계가 없고, 또 테이블의 컬럼들이 많아지게 되면 코드 작성과 관리가 굉장히 불편한 작업들이 된다. Dynamic Query Class는 이러한 작업들을 효율적이고 생상성 있게 하기 위한 쿼리문 작성 클래스이다. 이 문서는 이 클래스들의 활용과 정의에 대해서 다룬다.

키워드 : Dynamic, Transaction, Query, Class

Ⅰ. 서론

데이터베이스 처리를 위한 프로그래밍에서 트랜잭션 쿼리문(Insert,Update,Delete)을 작성할 때 프로시저를 사용하거나 코드에서 직접 작성하게 됩니다. 코드에서 쿼리문을 직접 작성할 때는 주로 문자열을 조합해서 쿼리문을 만들게 된다. 이런 문자열 조합의 쿼리문 작성 방법은 일정한 규칙이 없이 매우 다양하게 작성되고 있으며, 이로 인해 체계가 없고 매우 유동적이다. 또한, 테이블의 필드가 많아지면 코딩과 유지관리가 불편해진다. 이런 단점을 보완하고 쿼리문을 체계적이고 효율적으로 작성할 수 있도록 한 것이 Dynamic Query Class이다. 이 class는 코드에서 쿼리문을 작성할 때 Class의 인스턴스를 선언해서 사용한다. 코딩과 샘플 예는 모두 Visual Studio.NET 2005버전에서 C#언어로 작성되었다.

Ⅱ. 기존의 쿼리 작성방법과 Dynamic Query Class 사용 비교

2.1 기존의 쿼리 작성 방법

Members라는 테이블에 INSERT 하는 쿼리문 "INSERT INTO Members(Name,Age,Email,CellPhone,Tell,Zip) VALUES('박선희',29,'parksh@msn.com','019 429 3333','02 123 4567','180 120')" 이 있다고 한다면, 이를 코드에서 구현하려고 하면, 보통 다음과 같이 string 변수를 선언해서 쿼리문을 조합해서 만들게 된다. (Age 필드는 숫자형 필드라고 가정한다.)

```
string str = null;
str+= "INSERT INTO ";
str+=
"Members(Name,Age,Email,CellPhone,Tell,Zip) ";
str+= "VALUES('박선
희',29,'parksh@msn.com','019-429-3333','02-
123-4567','180-120')";
```

또한, INSERT 하려는 값들이 위와 같이 고정된 것들이 아니라, 다음과 같이 화면에서 값을 입력하고 [저장] 버튼을 클릭할 때 이들 값들을 받아서 INSERT 쿼리문을 만들고자 한다면, Text속성 값들을 조합하고, 문자형 타입 필드에 대

해서는 다음과 같이 '문자열' 처리를 해줘야 하고 값들 사이에 쉼표(,)도 삽입해야 한다.

성명	박선희
나이	29
이메일 주소	parksh@msn.com
핸드폰	019-429-3333
전화번호	02-123-4567
우편번호	180-120

저장 수정 삭제

```
        string str = null;
      str += "INSERT INTO
Members(Name,Age,Email,CellPhone,Tell,
Zip) VALUES(";
      str += "'" + txtName.Text + "',";
      str +=  txtAge.Text + ",";
      str += "'" + txtEmail.Text + "',";
      str += "'" + txtCellPhone.Text
+"',";
      str += "'" +txtTell.Text +"',";
      str += "'" + txtZip.Text +"'";
      str += ")";

      //마지막에 조합된 str 변수 실행
```

또, 여기서 어느 컬럼은 INSERT할 필요가 없다고 한다면 중간에서 찾아서 삭제하거나 주석 처리를 해야 한다. 반대로 필드를 추가하고자 한다면 앞부분에 필드명을 추가하고 VALUES 부분에도 해당 값을 순서에 맞쳐서 추가한다. 코드 작성에서 이런 일련의 작업들은 상당히 신경이 쓰이고 자칫 사소한 부주의로 인해서 쿼리문을 잘못 작성할 확률이 매우 높다. 그리고 테이블의 컬럼들의 수가 많아지게 되면 코드에서 쿼리문 작성과 관리가 매우 불편해진다. 이런 식의 문자열 조합의 쿼리문 작성은 체계적이지 않고 매우 유동적이다.

2.2. Dynamic Query Class 사용

2.2.1 DqInsert 클래스를 사용하여 INSERT 쿼리문 생성하기

위에서 화면에서 입력된 필드 값들을 받아서 INSERT 쿼리문을 만들 때 문자형 데이터 필드에 대해서는 '문자열' 처리를 하고 문자열들을 조합해서 최종 INSERT 쿼리문을 만들었는데, DqInsert 클래스를 사용하면 문자형 타입 컬럼은 AddInsStr 함수를, 숫잣형 타입 컬럼은 AddInsInt 함수를 호출하고 마지막에 ExecuteQuery() 함수를 호출하면 내부적으로 "INSERT INTO Members(Name,Age,Email, CellPhone,Tell,Zip) VALUES('박선희',29,'parksh@msn.com','019 429 3333','02 123 4567','180 120')" 쿼리문이 만들어져 실행되게 된다. 다음은 그 구현 예이다.

```
//생성자에서 INSERT할 Members 테이블을 지
정한다.
      DqInsert dqi = new
DqInsert("Members");
      dqi.AddInsStr("Name", txtName.Text);
      dqi.AddInsInt("Age", txtAge.Text); //숫
자형 타입 컬럼
      dqi.AddInsStr("Email", txtEmail.Text);
      dqi.AddInsStr("CellPhone",
txtCellPhone.Text);
      dqi.AddInsStr("Tell", txtTell.Text);
      dqi.AddInsStr("Zip", txtZip.Text);

      //INSERT 쿼리문을 실행한다.
      dqi.ExecuteQuery();
```

AddInsStr 나 AddInsInt 함수에서 첫째 인자는 필드명이고 두번째 인자는 값을 나타낸다. 기본적인 INSERT 쿼리문의 패턴을 "INSERT INTO Members(Name,Age,Email ,CellPhone,Tell,Zip) VALUES('박선희',29,'parksh@msn.com', '019 429 3333','02 123 4567','180 120')" 쿼리문의 예로 들어 보면,

Name > '박선희'

Age > 29

Email > 'parksh@msn.com

CellPhone > '019 429 333'

.....

이런 식의 필드명과 값의 쌍으로 되어 있고 이들은 필드명 그룹과 값의 그룹으로 나누어 볼 수 있다. 이런 패턴을 이용하여 AddInsStr 나 AddInsInt 함수에서 첫째 인자에 필드명을 주고, 둘째 인자에는 값을 주면, 마지막에 ExecuteQuery() 함수에서 최종적으로 Insert 쿼리문을 만드는데, 이때 첫번째 인자값들과 두번째 인자값들을 각각 분리하여 순서에 맞쳐서 "INSERT INTO 테이블명(첫번째 인자값 그룹) VALUES(두번째 인자값 그룹)" 쿼리문을 만들어낸다. 따라서 사용자는 테이블의 필드명과 이것의 값을 AddInsStr 나 AddInsInt 함수에서 차례로 호출만 해주면 된다.

그리고 값들을 크게 문자형 타입(AddInsStr)과 숫자형 타입(AddInsInt)으로 구분하여, 문자형 타입에 대해서는 내부에서 '문자열' 처리가 되어 있어 "'" + txtName.Text + "'," 와 같은 조합은 신경을 쓰지 않아도 된다.

이와 같이 이 클래스를 사용하면 쿼리문을 간단하면서도 체계적으로 작성할 수 있을 뿐만 아니라 몇 가지 더 편리한 기능도 가지고 있다. 그 중에 하나가 쿼리문 작성 중에 필드

의 삭제와 추가가 매우 자유롭다. 예를 들어, Age 필드와 Tell 필드는 쿼리문에 추가할 필요가 없다면 다음과 같이 주석 처리(또는 삭제)만 해주면 된다.

```
//생성자에서 INSERT할 Members 테이블을 지정한다.
        DqInsert dqi = new
DqInsert("Members");

        dqi.AddInsStr("Name", txtName.Text);
        //dqi.AddInsInt("Age", txtAge.Text);//-
주석처리
        dqi.AddInsStr("Email", txtEmail.Text);
        dqi.AddInsStr("CellPhone",
txtCellPhone.Text);
        //dqi.AddInsStr("Tell", txtTell.Text); //-
-주석처리
        dqi.AddInsStr("Zip", txtZip.Text);

        //INSERT 쿼리문을 실행한다.
        dqi.ExecuteQuery();
```

주석 처리가 된 상태에서 ExecuteQuery() 함수가 호출되면 내부적으로 주석 처리가 된 필드가 제외된 "INSERT INTO Members(Name, Email, CellPhone, Zip) VALUES('박선희', 'parksh@msn.com', '019 429 3333', '180 120')" INSERT 쿼리 문이 만들어지게 된다.

또, 필드 순서에 상관 없이 추가가 가능하다. 다음과 같이 Age 필드를 맨 앞에서 호출했다면 최종적으로 "INSERT INTO Members(Age, Name, Email,CellPhone,Tell,Zip) VALUES(29, '박선희','parksh@msn.com','019 429 3333','02 123 4567','180 120')" 쿼리문이 만들어 지게 된다.

```
//생성자에서 INSERT할 Members 테이블을 지정한다.
        DqInsert dqi = new
DqInsert("Members");

dqi.AddInsInt("Age", txtAge.Text); //숫자형
타입 컬럼
        dqi.AddInsStr("Name", txtName.Text);
dqi.AddInsStr("Email", txtEmail.Text);
        dqi.AddInsStr("CellPhone",
txtCellPhone.Text);
        dqi.AddInsStr("Tell", txtTell.Text);
        dqi.AddInsStr("Zip", txtZip.Text);

        //INSERT 쿼리문을 실행한다.
        dqi.ExecuteQuery();
```

2.2.2 자동 생성된 쿼리문 보기

이렇게 ExecuteQuery() 함수를 호출하면 내부에서 쿼리문이 생성되어 실행되는데, 쿼리문에서 에러가 발생하거나 의심이 가는 경우 실제로 쿼리문이 제대로 생성되고 있는지 확인할 필요가 있다. 이때에는 GetQuery() 함수를 이용해서 생성되는 쿼리문을 확인할 수 있다. 다음은 그 사용 예이다.

```
//생성자에서 INSERT할 Members 테이블을
지정한다.
        DqInsert dqi = new
DqInsert("Members");

dqi.AddInsStr("Name", txtName.Text);
        dqi.AddInsInt("Age", txtAge.Text); //
숫자형 타입 컬럼
        dqi.AddInsStr("Email", txtEmail.Text);
        dqi.AddInsStr("CellPhone",
txtCellPhone.Text);
        dqi.AddInsStr("Tell", txtTell.Text);
        dqi.AddInsStr("Zip", txtZip.Text);

//생성되는 쿼리문을 본다.
         Response.Write(dqi.GetQuery());
//WinForm일 경우는 MessageBox.Show함수
로 호출한다.
         //MessageBox.Show(dqi.GetQuery());

        //INSERT 쿼리문을 실행한다.— 쿼리문
확인을 위해 잠시 주석 처리
        //dqi.ExecuteQuery();
```

여기서 GetQuery() 함수는 "INSERT INTO Members (Name,Age,Email,CellPhone,Tell,Zip) VALUES('박선희',29,'parksh@msn.com','019 429 3333','02 123 4567','180 120')" 쿼리문을 생성한다.

2.2.3 실행된 쿼리문의 성공 여부 확인하기

Insert 쿼리문을 실행했으면 제대로 실행이 되었는지 확인할 필요가 있다. 이때 에는 ExecuteQuery()함수의 리턴되는 결과값으로 확인한다. DqInsert인 경우 리턴되는 값은 추가된 행(Row)의 수를 나타낸다.

```
//생성자에서 INSERT할 Members 테이블을
지정한다.
        DqInsert dqi = new
DqInsert("Members");
        dqi.AddInsStr("Name", txtName.Text);
        dqi.AddInsInt("Age", txtAge.Text); //
숫자형 타입 컬럼
        dqi.AddInsStr("Email", txtEmail.Text);
        dqi.AddInsStr("CellPhone",
txtCellPhone.Text);
        dqi.AddInsStr("Tell", txtTell.Text);
```

```
        dqi.AddInsStr("Zip", txtZip.Text);

//추가되는 행의 수는 0보다 크다
        if (dqu.ExecuteQuery() > 0)
        {
            Response.Write("Success!");
            //WinForm일 경우는
MessageBox.Show함수로 호출
}
```

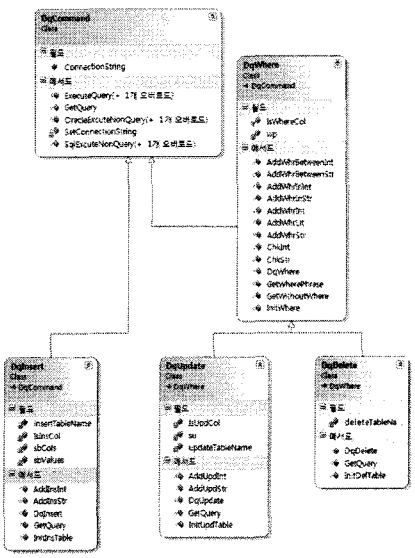
DqInsert 클래스 외에 DqUpdate 클래스와 DqDelete 클래스도 있는데, 이들은 다음에 이어지는 [Dynamic Query Class 구성]에서 설명한다.

I. Dynamic Query Class 구성

Dynami Query Class는 DqInsert 클래스, DqUpdate 클래스, DqDelete 클래스, DqWhere 클래스, 그리고DqCommand 클래스로 구성되어 있다.

3.1 클래스 다이어그램

다음은 클래스간 전체 관계를 나타내는 클래스 다이어그램이다.



쿼리문 생성 클래스들(DqInsert,DqUpdate,DqDelete)은 모두 DqCommand 클래스들을 상속받고, 여기에 있는 가상함수 GetQuery를 override하여 각자의 쿼리문(Insert, Update,Delete)을 만들게 된다. DqCommand 클래스는 ExecuteQuery 함수에서 GetQuery 함수에서 만들어진 최종 쿼리문을 받아서 DB에 접속해서 쿼리문을 실행한다. 그리고 DqUpdate 클래스와 DqDelete 클래스는 Where 구 부분을 생성하기 위해서 DqWhere 클래스를 상속받는다.

3.2 각 구성 클래스의 설명

다음은 각 클래스의 간단한 설명이다.

클래스 명	설 명
DqInsert	INSERT 쿼리문을 작성하는 클래스 컬럼(필드)를 추가할 때는 AddInsInt 함수나 AddInsStr함수를 사용함
DqUpdate	UPDATE 쿼리문을 작성하는 클래스 컬럼(필드)를 추가할 때는 AddUpdInt 함수나 AddUpdStr함수를 사용함
DqDelete	DELETE 쿼리문을 작성하는 클래스
DqWhere	DqUpdate와 DqDelete 클래스가 상속받음. 쿼리문에서 Where 구 부분 작성을 담당함.
DqCommand	DB연결 정보를 담고 있고 DqInsert, DqUpdate, DqDelete 클래스가 ExcuteQuery 함수를 통해서 DB에 연결하고 생성된 쿼리문을 실행함.

3.2.1 DqInsert 클래스를 사용하여 Insert 쿼리문 생성하기

이 부분은 [2.2.1 DqInsert 클래스를 사용하여 INSERT 쿼리문 생성하기]와 같음.

3.2.2 DqUpdate 클래스를 사용하여 Update 쿼리문 생성하기

Members 테이블에서 email 필드의 값 'parksh@msn.com'으로 Update 하고자 한다면, 기존에는 다음과 같은 문자열 조합으로 작성을 하였다.

```
string str = null;
str += "UPDATE MEMBERS SET ";
        str += "Name='" + txtName.Text +
"',";
        str += "Age=" + txtAge.Text + ",";
        str += "Email='" + txtEmail.Text +
"',";
        str += "CellPhone='" +
txtCellPhone.Text + "',";
        str += "Tell='" + txtTell.Text + "',";
        str += "Zip='" + txtZip.Text + "'";
str += "Where Email='" + txtEmail.Text +
"'";

        //마지막에 조합된 str 변수 실행
```

그러나 다음과 같이 DqUpdate 클래스를 이용하면 똑같은 쿼리문을 작성할 수 있다. ExecuteQuery() 함수 호출 단계에서 최종적으로 Update 쿼리문이 만들어져서 실행되게 된다.

```
//생성자에서 Update할 Members 테이블을 지정한다.
        DqUpdate dqu = new
DqUpdate("Members");
```

```
        dqu.AddUpdStr("Name",
txtName.Text);
        dqu.AddUpdInt("Age", txtAge.Text);
        dqu.AddUpdStr("CellPhone",
txtCellPhone.Text);
        dqu.AddUpdStr("Tell", txtTell.Text);
        dqu.AddUpdStr("Zip", txtZip.Text);

//Where 구 부분을 만든다.
        dqu.AddWhStr(null, "Email", "=",
txtEmail.Text);

        //Update 쿼리문을 실행한다.
          if (dqu.ExecuteQuery() > 0) //리턴
값은 Update된 행의 수를 나타낸다.
        {
            .....................................
        }
```

DqInsert 클래스와 마찬가지로 Age 필드와 Tell 필드는 업데이트할 필요가 없다면, 이들을 주석 처리하면 Age 필드와 Tell 필드가 제외된 Update 쿼리 문이 생성되어 실행되게 된다. Where 구 만드는 부분(AddWhStr)에 대해서는 DqWhere 클래스에서 자세히 설명한다.

역시 여기서도 생성되는 Update 쿼리문을 확인할 필요가 있다면 DdInsert 클래스의 예와 같이 GetQuery()를 호출한다.

3.3.3 DqDelete 클래스를 사용하여 Delete 쿼리문 생성하기

Members 테이블에서 email 필드의 값 'parksh@msn.com'으로 Delete하고자 한다면 다음과 같이 DqDelete 클래스를 사용한다.

```
//생성자에서 Delete할 Members 테이블을 지
정한다.
        DqDelete dqd = new
DqDelete("Members");

        //Where 구 부분을 만든다.
        dqd.AddWhStr(null, "Email", "=",
txtEmail.Text);

          //Delete 쿼리문을 실행한다.
         if (dqd.ExecuteQuery() > 0) //리턴
값은 Delete된 행의 수를 나타낸다.
        {
            .....................................
        }
```

ExecuteQuery() 함수 호출 단계에서 내부적으로 "DELETE FROM Members Where Email='parksh@msn.com'" 쿼리문이 생성되어 실행되게 된다.

이 클래스에서도 생성되는 Delete 쿼리문을 확인하고자 한다면 GetGuery() 함수를 호출한다.

3.3.4 DqWhere 클래스 보기

DqWhere 클래스는 DqUpdate, DqDelete 클래스가 상속받아서 사용하며, 쿼리문의 Where 구 부분을 생성한. 예를 들어 Members 테이블에서 Age가 30이고 Zip 코드가 '180 330' 인 행을 삭제 하고자 한다면 다음과 같이 작성한다.

```
//생성자에서 Delete할 Members 테이블을 지
정한다.
        DqDelete dqd = new
DqDelete("Members");

          dqd.AddWhInInt(null, "Age", "=",
"30");
        dqd.AddWhInStr("and", "Zip", "=",
"180-330");

        //Delete 쿼리문을 실행한다.
        dqd.ExecuteQuery();
```

AddWhInStr 함수나 AddWhInInt 함수를 맨 처음 호출할 때 첫째 인자는 null로 지정한다. 그리고 두번째 함수 호출부터는 첫번째 인자에 and 또는 or를 지정한다. 두번째 인자는 필드명,셋번째 인자는 연산자(=,like,is 등),네번째 인자는 해당 필드의 값을 입력한다. 그리고 숫자형 타입 컬럼에 대해서는 AddWhInInt 함수를,문자형 타입 컬럼에 대해서는 AddWhInStr 함수를 호출한다. 문자형 타입 컬럼 AddWhInStr 함수를 호출하면 내부적으로 필드명='값' 와 같이 ' 처리가 된다.

ExecuteQuery() 함수 호출 단계에서 내부적으로 "DELETE FROM Members Where Age=30 and Zip='180 330' 쿼리 문이 만들어져 실행되게 된다.

또한 Age 컬럼에 대해서는 조건이 필요없다면 다음과 같이 이 부분만 주석 처리하거나 삭제하면 "DELETE FROM Members Where Zip='180 330'" 쿼리문이 생성되게 된다.

```
 //생성자에서 Delete할 Members 테이블을 지
정한다.
        DqDelete dqd = new
DqDelete("Members");

        //dqd.AddWhInInt(null, "Age", "=",
"30"); //-주석처리
        dqd.AddWhInStr("and", "Zip", "=",
"180-330");

        //Delete 쿼리문을 실행한다.
        dqd.ExecuteQuery();
```

위와 같은 DqDelete 클래스에서의 Where 구의 사용 방법은 DqUpdate 클래스에서도 같은 방법으로 사용한다.

3.3.4.1 다양한 Where 구의 구현

Where 구에는 '=' 연산자뿐만 아니라 '>=', '<=', 'LIKE', 'Between', In()' 와 같은 다양한 연산자 조건이 올 수 있는데, 이 모든 것도 DqWhere에서 구현 가능하다.

"Where Age >=20" 구현 예

```
Dqd.AddWhInInt(null, "Age", ">=", "20");
```

"Where Age <=20" 구현 예

```
dqd.AddWhInInt(null, "Age", "<=", "20");
```

"Where Age Between 20 and 29" 구현 예

```
dqd.AddWhBetweenInt(null, "Age", "20", "29");
```

"Where Age in (19,23,28,30)" 구현 예

```
dqd.AddWhInInt(null, "Age", "19", "23", "28",
"30");
```

또, "Where Email = 'parksh@msn.com' or (Name='박선희' and Age=29)" 처럼 단순 조건이 아니라 괄호()가 들어가는 경우는 단순 문자열을 추가하는 AddWhrLit 함수를 이용해서 Where구를 만들 수 있다. 다음은 그 구현 예이다.

```
.....
// Where Email = 'parksh@msn.com' or (
Name = '박선희' and Age = 29 )
        dqd.AddWhrStr(null, "Email", "=",
"parksh@msn.com");// Email =
'parksh@msn.com'

        dqd.AddWhrLit(" or (");// or (
        dqd.AddWhrStr(null, "Name", "=", "박
선희");// Name = '박선희'
        dqd.AddWhrInt("and", "Age", "=",
29);// and Age = 29
        dqd.AddWhrLit(")");// )
        .....
```

AddWhrLit 함수는 인자로 들어오는 값을 그대로 Where 구에 추가한다. 그리고 AddWhrStr(null, "Name", "=", "박선희") 처럼 첫째 인자가 null이면 and나 or가 추가되지 않고 두번째 인자부터 Name='박선희' 와 같이 조건이 만들어 지게 된다.

3.3.5 DqCommand 클래스에서 DB 연결 설정하기

최종적으로 생성된 쿼리문을 실행하는 곳은 DqCommand 클래스의 ExcuteQuery() 함수이다. 이 함수는 SqlExcuteNonQuery 함수(SQL SERVER)나 OracleExcuteNonQuery 함수(Oracle)를 사용하는데, 여기에는 web.config(WinForm인 경우 app.config)의 ConnectionString섹션의 add name부분을 읽는 SetConnectionString() 함수를 호출한다.

따라서 config 파일에서 다음과 같이 〈add name="…."/〉 부분에 DB 연결 정보가 있다면,

```
<connectionStrings>
  <add
name="TestDatabaseConnectionString1"
connectionString="Data
Source=.₩SQLEXPRESS;AttachDbFilename=
|DataDirectory|₩TestDatabase.mdf;Integrate
d Security=True;User Instance=True"
   providerName="System.Data.SqlClient" />
</connectionStrings>
```

이 이름을 SetConnectionString() 함수에서 호출을 하는데, 다음과 같이 ConfigurationManager.ConnectionStrings["TestDatabaseConnectionString1"]처럼 지정해 준다.

```
/// <summary>
        /// DB 연결 문자열 정보 저장 필드
        /// </summary>
        public string ConnectionString;

        /// <summary>
        /// ConnectionString이 null이면 Config
파일에서 DB연결 정보를 읽는다.
        /// </summary>
        private void SetConnectionString()
        {
            if (ConnectionString == null)
            {
                //Config 파일의 ConnectionStrings
섹션의 name을 여기서 지정한다.- 임시로
TestDatabaseConnectionString1로 지정했다.
                ConnectionStringSettings conStr
=
ConfigurationManager.ConnectionStrings["Te
stDatabaseConnectionString1"];
                ConnectionString =
conStr.ConnectionString;
            }

        }
```

아니면, DB연결 정보를 config 파일에서 읽는 것이 아니라 코드에서 직접 지정해서 사용하고자 한다면 다음과 같이

ConnectionString 필드에서 지정한다.

```
/// <summary>
        /// DB 연결 문자열 정보 저장 필드
        /// </summary>
        public string ConnectionString =
"DB 연결 정보";
```

ExcuteQuery() 함수에서 SQL SERVER를 사용하면 SqlExcuteNonQuery 함수를, ORACLE을 사용한다면 OracleExcuteNonQuery 함수를 호출한다.

```
/// <summary>
        /// Query문을 실행시킨다.
        /// </summary>
        /// <returns></returns>
        public int ExecuteQuery()
        {
            //최종 쿼리문을 얻는다.
            string cmdText = GetQuery();

            //SQL SERVER
            return
SqlExcuteNonQuery(cmdText);

            //ORACLE 이라면
OracleExcuteNonQuery 함수를 호출한다.
            //return
OracleExcuteNonQuery(cmdText);
        }
```

My Sql이나 기타 다른 DB를 사용한다면, DdCommand 클래스에서 위와 같이 SqlExcuteNonQuery 함수나 OracleExcuteNonQuery 함수처럼 먼저 DB 연결 함수를 만들어서 추가하고, 이 함수를 ExcuteQuery() 함수에서 호출하도록 변경해서 사용한다.

II. 결론

Dynamic Query Class는 위에서 보인 것처럼 INSERT, DELETE, UPDATE 쿼리문을 체계적으로 작성할 수 있는 클래스이다. 또, 기존에는 코드에서 쿼리문을 작성할 때 해당 컬럼에 대해서 처리하는 값들이 변수로 되어 있다면 컬럼 데이터 타입이 문자(varchar)냐, 숫자(decimal)냐에 따라 문자열 조합 처리를 해주었는데, 이 클래스는 이런 처리도 내부에서 자동으로 처리되도록 설계가 되어 있다. 또한 쿼리문의 다양한 Where 구 구현이 가능하다. 이들 클래스의 특징을 정리하면 다음과 같다.

1) *INSERT, DELETE, UPDATE 쿼리문의 체계적 작성*
2) *컬럼 데이터 타입에 따른 문자열 처리 자동화*
3) *DELETE, UPDATE 쿼리문에서 다양한 Where 구의 작성*
4) *각 컬럼의 자유로운 추가와 삭제(주석) 따른 쿼리문 생성*
5) *쿼리문 작성과 관리의 편리성*