ADO.Net 2 연결과 명령

연결

- 실무에서는 중앙의 DB 서버가 데이터를 관리하고 클라이언트는 서버로부터 데이터를 공급받아 사용하는 것이 일반적이다.
- 비연결형이라도 일단은 서버의 데이터를 받아와야 하므로 연결이 필요하다.
- 연결 클래스는 공급자별로 이름이 다르다.

공급자	연결	명령	리더	어댑터
SQL	SqlConnection	SqlCommand	SqlDataReader	SqlDataAdapter
OLE DB	OleDbConnection	OleDbCommand	OleDbDataReader	OleDbDataAdapter
오라클	OracleConnection	OracleCommand	OracleDataReader	OracleDataAdapter
ODBC	OdbcConnection	OdbcCommand	OdbcDataReader	OdbcDataAdapter

- 인터페이스 수준에서 호환되므로 하나만 제대로 익히면 나머지도 동일한 방법으로 사용할 수 있다.
- 서버마다 장단점과 고유의 특징이 있고 내부적인 구현 방법이 상이해 약간씩의 차이는 있다.

연결

- **일단 서버와 연결해야 명령을 내릴 수 있다.**
  - public SqlConnection ()
  - public SqlConnection (string connectionString)
- **연결 문자열로 접속 대상을 지정하며 프로퍼티로 속성을 지정한다.**

프로퍼티	타입	설명
ConnectionString	string	연결 문자열이다.
ConnectionTimeout	int ⓡ	연결될 때까지의 대기 시간이며 디폴트는 15초이다. 이 시간동안 연결되지 않으면 에러가 리턴된다.
Database	string ⓡ	연결된 데이터베이스 이름이다.
DataSource	string ⓡ	연결된 SQL Server의 인스턴스 이름이다.
PacketSize	int ⓡ	SQL 서버와 통신할 때 사용할 패킷의 크기이다. 디폴트는 8192바이트이다.
ServerVersion	string ⓡ	SQL 서버의 버전 정보를 조사한다.
State	ConnectionState ⓡ	연결의 현재 상태이다. Open, Closed 등이 있다.

- **메서드 목록**

메서드	설명
Open	연결 문자열을 참조하여 데이터베이스에 연결한다. 실패시 예외가 발생한다.
Close	연결을 해제한다. 실행중인 트랜잭션이 있으면 모두 롤백된다.
BeginTransaction	새로운 트랜잭션을 시작하고 Transaction 객체를 리턴한다. 인수로 트랜잭션의 이름을 지정하거나 격리 수준을 설정한다.
CreateCommand	연결 객체와 관련된 명령 객체를 생성한다.
ChangeDatabase	현재 데이터베이스를 변경한다.

연결 문자열

- 연결 문자열 : 연결에 필요한 주소, ID, 비밀 번호 등의 정보를 일반적인 포맷인 문자열 형태로 정의한 것이다.
- 키=값" 대입문을 세미콜론으로 구분하여 여러 정보를 포함한다.

키	디폴트	설명
Data Source 또는 Server		연결할 SQL 서버의 인스턴스 이름을 지정한다. 로컬 인스턴스를 지정할 때는 (local)이라고 적는다. 서버명앞에 프로토콜 접두어를 붙일 수 있고 서버명 뒤에 포트 번호를 지정한다.
Initial Catalog 또는 Database		최초 접속할 데이터베이스의 이름이다.
Integrated Security 또는 Trusted_Connection	false	윈도우의 인증 모드를 사용한다. 이 값이 false이면 사용자 ID와 암호를 지정해야 한다. true, yes, sspi 등은 혼합모드를 쓰며 false, no는 쓰지 않는다.
User ID		로그인 계정이다.
Password, Pwd		로그인 계정에 대한 ID이다.
Packet Size	8192	PacketSize 프로퍼티와 동일한 의미이다.
Connect Timeout		ConnectionTimeout 프로퍼티와 동일한 의미이다.

- 전체 키 목록은 레퍼런스를 참고한다.
- 직접 만드는 것보다는 예제의 문자열을 조금씩 변형해 쓰는 것이 편리하다.

연결 문자열

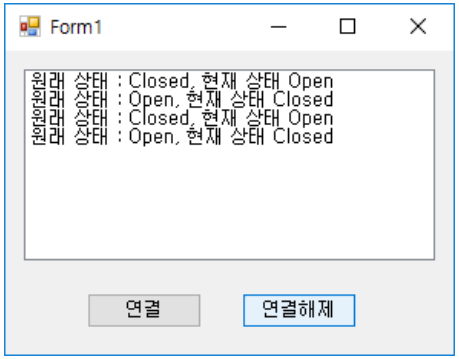
- 로컬 서버의 ADOTest DB에 연결하기

```
Con = new SqlConnection();
Con.ConnectionString = "Server=(local);database=ADOTest;" +
    "Integrated Security=true";
try {
    Con.Open();
}
catch (Exception ex) {
    MessageBox.Show(ex.Message);
}
....
Con.Close();
```

- 접속 실패 확률이 높아 예외 처리 구문이 반드시 필요하다.
- 다 사용한 후 Close나 Dispose로 연결을 닫아야 한다.
- SQL 서버의 통합 인증 모드가 아니라면 계정과 ID를 지정한다.
  - Con.ConnectionString = "Server=(local);database=ADOTest;User ID=sa;Pwd=";
- SQL 서버는 통합 보안을 더 권장한다.

연결의 이벤트

- StateChange 이벤트 : State 프로퍼티의 값이 바뀔 때마다 즉, 연결 상태가 바뀔 때마다 전달된다.
- InfoMessage 이벤트 : SQL 서버로부터 경고나 정보 메시지가 리턴될 때 발생한다.
- ConnectionEvent 예제 : 이벤트 발생 사실을 리스트 박스에 출력한다.



- 연결시 할 작업을 StateChange 이벤트에 정의하되 그보다는 Open 메서드 호출 직후에 하는 것이 더 간편하다.

SQL문 실행

- **명령은 SqlCommand 클래스로 표현한다.**
  - public SqlCommand (string cmdText,SqlConnection connection)
  - public SqlCommand ()
- **명령 문자열과 연결 객체에 대한 참조를 전달한다. 빈 명령 객체를 생성한 후 프로퍼티로 지정할 수도 있다.**

프로퍼티	타입	설명
Connection	SqlConnection	연결 객체에 대한 참조이다.
CommandText	string	실행할 SQL 명령 문자열이다.
CommandTimeout	int	명령 실행 타임아웃값이며 디폴트는 30초이다.
CommandType	CommandType	명령의 유형이다. Text이면 SQL 명령이고 StoredProcedure이면 저장 프로시저이고 TableDirect이면 테이블의 이름이되 OLE DB 공급자에서만 가능하다. Text가 디폴트이다.
Parameters	SqlParameterCollection ⓡ	SQL이나 저장 프로시저로 전달되는 인수의 목록이다.
Transaction	SqlTransaction	명령이 실행되는 트랜잭션 객체이다.

- **명령의 종류에 따라 호출하는 메서드가 준비되어 있다.**

메서드	설명
ExecuteReader	SQL 명령을 실행하고 SqlDataReader 결과셋을 리턴한다. 주로 SELECT 명령에 대해 사용한다.
ExecuteScalar	SQL 명령을 실행하고 첫 번째 행의 첫 번째 열을 리턴한다. 집계 함수에 대해 사용한다.
ExecuteNonQuery	SELECT 이외의 쿼리를 실행하고 영향받은 행의 개수를 리턴한다. INSERT, UPDATE, DELETE 명령에 대해 사용한다.
ExecuteXmlReader	SQL 명령을 실행하고 XmlReader 결과셋을 리턴한다.

SQL문 실행

- ExecuteReader 메서드는 SELECT 명령문을 실행하고 그 결과셋을 SqlDataReader로 리턴한다.

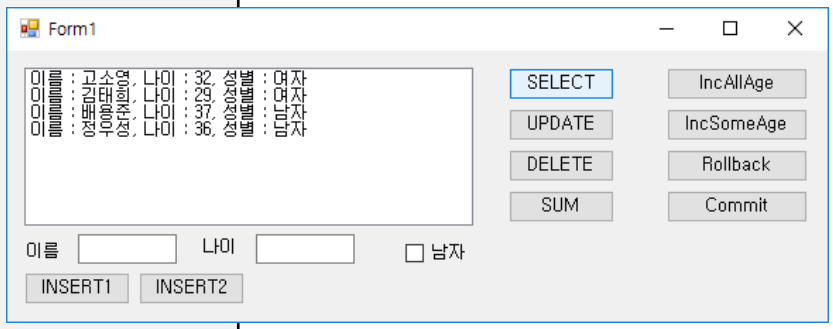
프로퍼티	타입	설명
FieldCount	int ⓡ	열 개수를 조사한다.
HasRows	bool ⓡ	결과셋에 행이 있는지 조사한다.
RecordsAffected	int ⓡ	SQL 실행 결과 영향을 받은 행의 개수를 조사한다. SELECT문은 -1이 리턴된다. 리더를 완전히 닫아야만 이 프로퍼티를 참조할 수 있다.
Connection	SqlConnection ⓡ	관련된 연결 객체이다.

- 다음 두 메서드로 결과셋을 순회한다.
  - public override bool Read ()
  - public override bool NextResult ()
- 리더 객체는 현재 읽은 레코드 하나만 메모리에 보유하며 Read를 호출할 때마다 다음 레코드를 읽어들인다.
- 보통 인덱서로 R["Name"] 식으로 참조한다.
- CommandTest 예제 : 명령으로 읽고 쓰고 지우는 실습 예제. Form1_Load 이벤트에서 로컬 SQL 서버의 ADOTest에 연결했으며 폼이 닫힐 때 연결을 끊는다.

테이블 읽기

- SELECT문을 실행하는 가장 일반적인 형태이다.

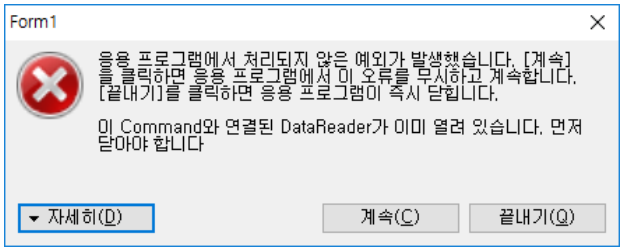
```
string Rec;
SqlCommand Com = new SqlCommand("SELECT * FROM tblPeople", Con);
SqlDataReader R;
R = Com.ExecuteReader();
listBox1.Items.Clear();
while (R.Read()) {
    Rec = string.Format("이름 : {0}, 나이 : {1}, 성별 : {2}",
        R["Name"], R["Age"], (bool)R[2] ? "남자" : "여자");
    listBox1.Items.Add(Rec);
}
R.Close();
```



- 명령을 실행하여 리더 객체를 구한다. Read 메서드가 false를 리턴할 때까지 반복하며 인덱서로 행 정보를 읽는다.
- GetInt32, GetString, GetDouble 등의 메서드로도 열을 읽으면 변환이 필요 없어 속도는 더 빠르다.
  - Rec = string.Format("이름 : {0}, 나이 : {1}, 성별 : {2}",
  - R.GetString(0), R.GetInt32(1), R.GetBoolean(2) ? "남자" : "여자");
- 타입이 정확해야 하며 이름으로 참조할 수 없는 불편함이 있다.

테이블 읽기

- 결과셋을 다 사용한 후에는 반드시 Close 메서드를 호출하여 닫아야 한다. 연결 하나당 하나의 결과셋만 유지할 수 있기 때문에 결과셋을 닫지 않으면 다음 명령을 수행할 수 없다.
- 닫지 않으면 다시 조회할 때 예외가 발생한다.



- 연결은 비관리 자원이며 가비지 컬렉터가 자동으로 해제하지 않는다.
- 리더 객체를 생성할 때마다 아예 R.Close()를 먼저 작성하는 습관을 들이는 것이 좋다.
- 공급자마다 연결을 닫는 방식에서 약간씩 차이가 있다.

복수 개의 결과셋 읽기

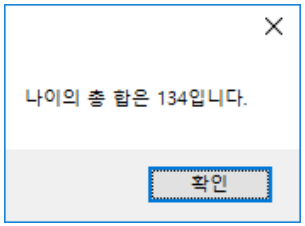
- 리문을 세미콜론으로 구분하여 두 개 이상의 명령을 동시에 실행할 수 있다.
- 각 결과셋 그룹을 이동할 때는 리더의 NextResult 메서드를 호출한다.

```
private void PrintTable() {
    string Rec;
    SqlCommand Com = new SqlCommand("SELECT * FROM tblPeople;" +
        "SELECT * FROM tblSale", Con);
    SqlDataReader R;
    R = Com.ExecuteReader();
    listBox1.Items.Clear();
    while (R.Read()) {
        Rec = string.Format("이름 : {0}, 나이 : {1}, 성별 : {2}",
            R["Name"], R["Age"], (bool)R[2] ? "남자" : "여자");
        listBox1.Items.Add(Rec);
    }
    R.NextResult();
    while (R.Read()) {
        Rec = string.Format("번호 : {0}, 고객 : {1}, 상품 : {2}, 날짜 : {3}",
            R["OrderNo"], R["Customer"], R["Item"], R["ODate"]);
        listBox1.Items.Add(Rec);
    }
    R.Close();
}
```

- 가급적이면 하나의 명령으로 하나의 결과셋을 읽는 것이 깔끔하다.

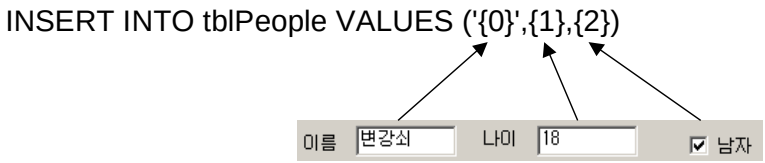
명령문 실행

- SqlCommand 객체로 임의의 SQL문을 실행할 수 있다.
- INSERT, DELETE, UPDATE, CREATE, ALTER, DROP .....
- 다음 명령은 서버의 데이터를 직접 수정한다.
  - string Sql = "UPDATE tblPeople SET Age = Age + 1 WHERE Name = '정우성'";
- ExecuteNonQuery메서드를 호출하며 결과셋이 아닌 영향받은 행의 수를 리턴한다.
- DELETE 쿼리문으로 삭제하되 무결성을 지켜야 한다.
- 집계 함수는 결과셋을 리턴하는 것이 아니라 단일값을 리턴하므로 ExecuteScalar 메서드로 실행한다.
  - string Sql = "SELECT SUM(Age) FROM tblPeople";



파라미터 사용

- SQL 쿼리문에는 검색 조건이나 삭제 대상, 수정할 값을 전달받는 인수가 필요하다.
- 적당한 컨트롤을 배치하고 사용자가 입력한 값을 SQL 쿼리문에 포함시킨다.
  - string Sql = string.Format("INSERT INTO tblPeople VALUES ('{0}',{1},{2})",
  - textName.Text, textAge.Text, checkMale.Checked ? 1:0);
- 사용자가 어떤 값을 입력하는가에 따라 쿼리문의 내용이 달라지며 실제로 삽입되는 레코드도 달라진다.



- 이런 조립 INSERT INTO tblPeople VALUES ('변강쇠',18,1) 다.
- 실행중에 결정되는 값에 @으로 시작하는 파라미터 이름만 적어 놓고 실제값은 쿼리문을 실행하기 전에 제공한다.

파라미터 사용

- **파라미터를 쓰는 쿼리문은 한 번만 컴파일되며 이후부터는 파라미터만 바꿔가며 실행할 수 있어 빠르다.**
  - public SqlParameter (string parameterName,SqlDbType dbType,int size, string sourceColumn)
- **이름, 타입, 크기, 컬럼 등을 인수로 받으며 프로퍼티로 변경한다.**

프로퍼티	타입	설명
ParameterName	string	파라미터의 이름이다. 첫 글자가 반드시 @이어야 한다.
DbType 또는 SqlDbType	DbType	파라미터의 타입이다. 디폴트는 NVarChar이다. SqlDbType 열거형에 SQL 서버가 지원하는 대부분의 타입이 포함되어 있다.
Size	int	파라미터의 최대 크기를 바이트 단위로 지정하며 유니코드 문자열에서는 문자의 개수이다. 주로 문자열 타입에 대해 사용되며 고정 길이 타입에 대해서는 무시된다. 생략시 실제값으로부터 유추한다.
Scale	byte	Decimal 타입에 대해 소수 자리수를 지정한다.
Direction	ParameterDirection	파라미터의 방향을 나타낸다. Input, Output, InputOutput, ReturnValue 네가지가 있으며 디폴트는 Input이다.
IsNullable	bool	NULL 허용 여부를 조사한다.
Value	object	파라미터의 값이다. 파라미터가 있는 명령은 실행전에 Value에 값을 대입한다. 출력용 인수도 Value를 통해 값을 확인한다.
SourceColumn	string	소스의 열이름이다.
SourceVersion	DataRowVersion	UPDATE를 수행하는 동안 필드의 어떤 값을 읽을 것인가를 지정한다. Original은 원래 들어 있던 값이고 Current는 현재값이다. 디폴트는 Current이다.

파라미터 사용

- **객체를 미리 생성한 후 컬렉션의 Add 메서드로 추가하거나 컬렉션의 Add 메서드로 파라미터의 주요 프로퍼티를 전달하여 생성과 동시에 추가한다.**
  - public SqlParameter Add (SqlParameter value)
  - public SqlParameter Add (string parameterName,SqlDbType sqlDbType, [int size, string sourceColumn])
- **파라미터의 실제값은 Value 프로퍼티를 통해 제공한다.**
- **마커 자리에 파라미터의 Value가 들어간다.**

```
private void btnInsert2_Click(object sender, EventArgs e) {
    string Sql = "INSERT INTO tblPeople VALUES (@Name,@Age,@Male)";
    SqlCommand Com = new SqlCommand(Sql, Con);
    Com.Parameters.Add("@Name", SqlDbType.NVarChar, 10);
    Com.Parameters.Add("@Age", SqlDbType.Int);
    Com.Parameters.Add("@Male", SqlDbType.Bit);

    Com.Parameters["@Name"].Value = textName.Text;
    Com.Parameters["@Age"].Value = textAge.Text;
    Com.Parameters["@Male"].Value = checkMale.Checked ? 1 : 0;

    Com.ExecuteNonQuery();
    PrintTable();
}
```

파라미터 사용

- 비슷한 명령을 반복적으로 실행한다면 Value 프로퍼티만 바꾸어 가며 호출할 수 있어 간편하다.
- 서버의 입장에서도 파라미터만 다른 명령은 실행 계획이 동일하며 캐시에 이미 컴파일되어 있어 더 빠르다.
- 파라미터를 쓰는 방법은 공급자별로 차이가 있다.
- OLE DB와 ODBC에서는 파라미터 마커로 이름을 쓸 수 없고 ?를 사용한다.
  - string Sql = "INSERT INTO tblPeople VALUES (?, ?, ?)";
- 파라미터의 이름이 없으므로 ?의 순서대로 파라미터가 대응된다.
- 형식이나 사용법은 달라도 쿼리문 실행전에 실제값을 제공하는 목적은 동일하다.

프로시저 실행

- 저장 프로시저는 서버에 미리 정의해 놓은 함수이다. 다음 구문으로 생성한다.
  - CREATE PROCEDURE IncAllAge
  - AS
  - UPDATE tblPeople Set Age = Age + 1;
- 코드에서 다음과 같이 호출한다.
  - private void btnIncAllAge_Click(object sender, EventArgs e) {
  - SqlCommand Com = new SqlCommand("IncAllAge", Con);
  - Com.CommandType = CommandType.StoredProcedure;
  - int nRow;
  - nRow = Com.ExecuteNonQuery();
  - PrintTable();
  - MessageBox.Show("영향받은 행수 = " + nRow);
  - }
- 프로시저 이름을 명령에 쓰고 타입을 StoredProcedure로 지정한다.

프로시저 실행

- 인수나 리턴값을 가질 수도 있다. 지정 고객의 나이를 증가시키고 나이를 리턴한다. SQL 서버에서 직접 호출할 수 있다.

```
CREATE PROCEDURE IncSomeAge
@Name NVARCHAR(10),
@Age INT OUTPUT
AS
UPDATE tblPeople SET Age = Age + 1 WHERE Name = @Name;
SELECT @Age = Age FROM tblPeople WHERE Name = @Name;
```

```
DECLARE @Age INT;
EXECUTE IncSomeAge '배용준', @Age OUTPUT;
SELECT @Age;
```

- 코드에서 호출할 때는 다음 코드를 사용한다.
- Direction 프로퍼티로 방향을
- 명시한다.
- @Name은 입력
- @Age는 출력
- 프로시저 자체의 리턴도 가능
- 하나 잘 사용하지 않는다.

```
private void btnIncSomeAge_Click(object sender, EventArgs e) {
    // 명령 객체 생성
    SqlCommand Com = new SqlCommand("IncSomeAge", Con);
    Com.CommandType = CommandType.StoredProcedure;

    // 파라미터 설정
    Com.Parameters.Add("@Name", SqlDbType.NVarChar, 10);
    Com.Parameters.Add("@Age", SqlDbType.Int);
    Com.Parameters["@Age"].Direction =
ParameterDirection.Output;

    // 호출
    Com.Parameters["@Name"].Value = "김태희";
    Com.ExecuteNonQuery();

    // 결과 출력
    PrintTable();
    int Age = (int)Com.Parameters["@Age"].Value;
    MessageBox.Show("프로시저 호출 후 나이 = " + Age);
}
```

트랜잭션 처리

- 트랜잭션은 작업을 일관되게 처리하는 서버측의 장치이다. 연결 수준에서 관리한다.
  - public SqlTransaction BeginTransaction ([IsolationLevel iso,string transactionName])
- 트랜잭션 시작 후의 모든 명령은 트랜잭션에 포함된다.
- 작업 후 다음 두 메서드로 실행 취소를 결정한다.
  - public override void Commit ()
  - public void Rollback ([string transactionName])
- 나이 이체의 예
  - UPDATE tblPeople SET Age = Age + 1 WHERE Name = '정우성';
  - UPDATE tblPeople SET Age = Age - 1 WHERE Name = '배용준';
- 둘 다 성공하거나 둘 다 실패해야 한다.

널 값

- 널(NULL)은 0과는 다른 특수한 값이다. 알 수 없음, 정의되어 있지 않음을 의미한다.
- 값이 아니라 일종의 상태이기 때문에 다룰 때 주의할 필요가 있다.
  - string Sql = "SELECT SUM(Age) FROM tblPeople WHERE Age > 100";
- 이 명령의 결과는 없으므로 NULL이다. 특정값이 NULL인지 알고 싶을 때는 리더 객체의 IsDBNull 메서드를 호출한다.
  - public override bool IsDBNull (int i)
- NULL이면 에러 처리한다.
  - if (R.IsDBNull(0) == true) {
  - textBox1.Text = "알 수 없는 값";
  - } else {
  - textBox1.Text = R[0].ToString();
  - }
- 널 가능 컬럼은 항상 값의 유효성을 점검해야 한다.