Bab 14

ADO.NET (1)

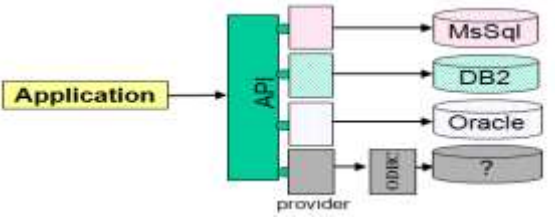
Sebuah ADO.NET adalah teknologi yang dipakai oleh .NET untuk mengakses data yang terstruktur, dimana teknologi ini melakukan penyeragaman terhadap sumber data yang berbeda dengan membuat sebuah *interface* . Sumber Data dapat terdiri dari:

- Database relasional
- Data XML
- Sumber Data lain, text, CSV dll

Teknologi ADO.NET didesain untuk dipakai pada suatu aplikasi yang bersifat terdistribusi dan aplikasi berbasis Web. Teknologi ini memiliki dua mode cara pengaksesan yaitu:

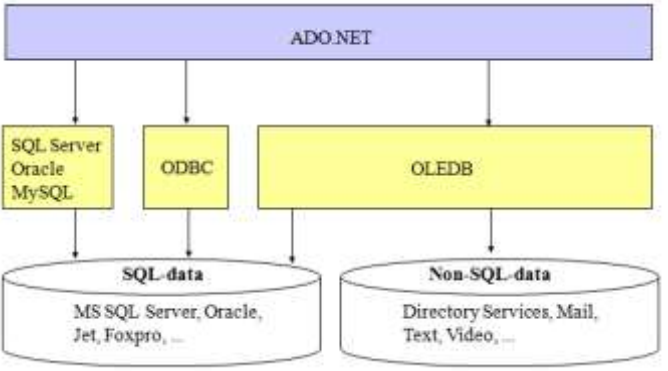
1. Connection-oriented
2. Connectionless

Konsep penyeragaman data merupakan sebuah kelanjutan dari pemakaian konsep pemrograman bersifat obyek, dimana sebuah data dianggap sebagai obyek, sehingga tidak dipermasalahkan lagi sumber datanya. Penyeragaman ini membutuhkan sebuah 'jembatan', yang didalam pemrograman ini disebut sebagai API (*application programming interface*) yang akan mengkases sumber data lewat sebuah penyedia (*provider*). Hubungan antara sebuah aplikasi, pemrograman dan sumber data ditunjukkan pada gambar 14.1 dibawah ini:



Gambar 14.1 Hubungan Aplikasi dan Sumber Data

Didalam dunia pemrograman dikenal istilah *provider* sebagai penyedia data. Suatu provider membuat sumber data dapat diakses oleh aplikasi, berikut ini peran dari provider (warna kuning) :



Gambar 14.2 Peran Provider

Sebelum lahir ADO.NET, terlebih dahulu lahir teknologi ADO yang merupakan singkatan dari *Active Data Object* . Sebuah teknologi dari microsoft untuk mengakses sumber data. Rangkaian teknologi ini dimulai dari ODBC sampai lahir teknologi ADO.NET yang merupakan evolusi dari teknologi teknologi sebelumnya.

- ODBC
  - OLE DB
    - ADO (*ActiveX Data Objects*)
      - ADO.NET

Secara umum perbedaan antara ADO dan ADO.NET diperlihatkan pada tabel 14.1 dibawah ini:

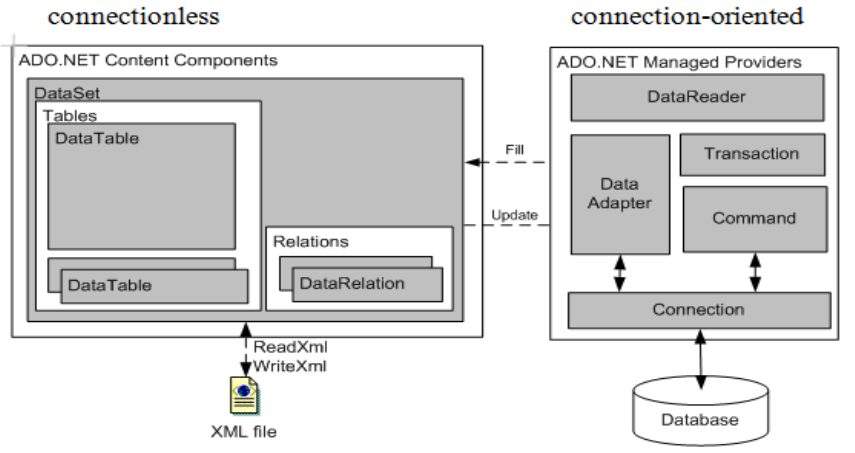
Tabel 14.1 Perbedaan ADO dan ADO.NET

ADO	ADO.NET
connection-oriented	connection-oriented + connectionless
sequential access	sequential access + main-memory representation with direct access
only one table supported	more than one table supported
COM-marshalling	XML-marshalling

Teknologi ADO hanya menyediakan sebuah koneksi yang bersifat *connection-oriented* dan teknologi ini dikembangkan lebih jauh lagi oleh teknologi ADO.NET menjadi *connection-oriented* dan *connectionless*. Cara akses dari ADO terbatas pada sekuens sedangkan pada ADO.NET, sebuah data dinyatakan pada memori sehingga cara akses dapat dilakukan secara langsung.

14.1 Arsitektur ADO.NET

Arsitektur ADO.NET terbagi menjadi dua bagian, yaitu bagian connectionless dan connection-oriented, seperti terlihat gb 14.3. Pada bagian *connection-oreinted* biasanya disebut pula sebagai *provider* .NET yang menangani koneksi ke database, penyedia data, pembacaan data dan perintah-perintah yang dikenal dalam transaksi databsase.



Gambar 14.3 Arsitektur ADO.NET

Pada arsitektur gb 14.3diatas, arsitektur terbagi dalam dua bagian besar yaitu bagian connection-oriented pada ADO.NET managed provider dan connectionless pada ADO.NET Content Component. Dua bagian ini memiliki peran yang berbeda:

Connection-oriented :

- Menjaga koneksi ke database agar tetap tersambung
- Data selalu up-to-date
- Aplikasi yang memakai metode ini disarankan hanya pada traksaksi yang singkat dan mengakses operasi secara pararel tidak terlalu sering

Connectionless

- Tidak ada koneksi yang permanent ke sumber data
- Data di 'cache' pada memori utama
- Perubahan data yang ada di memori utama dapat menyebabkan konflik dengan perubahan di sumber data
- Aplikasi yang memakai metode ini biasanya adalah pengguna yang banyak sehingga memiliki sifat akses ke aplikasi yang pararel dan aksesnya bersifat tahan lama, contohnya adalah Aplikasi berbasis Web.

Akses pada ADO.NET melibatkan pustaka-pustaka yang terbagi menjadi dua bagian: pustaka untuk Assembly melibatkan sebuah pustaka System.Data.dll dan pustaka untuk NameSpaces yang melibatkan pustaka yang bergantung pada jenis data antara lain:

- System.Data untuk tipe data yang umum
- System.Data.Common untuk mengimplementasi providers
- System.Data.OleDb untuk implementasi OLE DB provider
- System.Data.SqlClient untuk implementasi Microsoft SQL Server provider
- System.Data.SqlTypes untuk implementasi tipe data pada SQL Server
- System.Data.Odbc untuk implementasi ODBC provider
- System.Data.OracleClient untuk implementasi Oracle provider

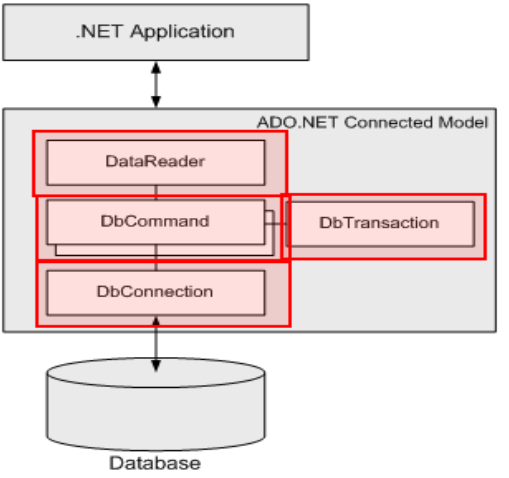
14.2 Akses Secara Connection Oriented

Salah satu kerangka kerja dalam .NET adalah *Data Provider* yang merupakan komponen yang telah secara eksplisit dirancang untuk manipulasi data dan akses secara cepat, akses bersifat forward-only, akses read-only data. Kerangka kerja Data Provide ini berisi obyek-obyek:

- *Connection* : sebuah obyek *Connection* menyediakan konektivitas ke sumber data.
- *Command: s*ebuah obyek *Command* memungkinkan akses ke perintah database untuk mengembalikan data, memodifikasi data, menjalankan prosedur yang tersimpan, dan mengirimkan atau mengambil informasi dengan parameter parameter tertentu.
- *dataReader : s*ebuah obyek dataReader menyediakan performa tinggi untuk mengalirkan data dari sumber data.
- *DataAdapter* : sebuah DataAdapter menyediakan jembatan antara objek DataSet dan sumber data. Obyek DataAdapter menggunakan obyek Command untuk menjalankan perintah

SQL pada sumber data, memuat data ke DataSet, mensinkronkan perubahan yang dilakukan terhadap data dalam DataSet untuk dilakukan pula dalam sumber data.

Arsitektur .NET yang mengakses secara *Connection-oriented* terdiri dari kelas-kelas *connection, command, datareader* dan *dataAdapter*.



Gambar 14.4 Model ADO.NET Connected

Kelas-kelas yang tersedia pada model gb14.4 merupakan sebuah interface yang secara umum dituliskan sebagai IDbConnection , IDbCommand , IDbTransaction , IDataReader . Implementasi interface tsb biasanya bergantung pada jenis sumber datanya, secara umum implementasi OleDb untuk implemetasi OLEDB, Sql untuk implemetasi SQL Server, Oracle untuk implemetasi, Odbc untuk implemetasi.

Pola-pola umum pemrograman untuk mengakses sumber data secara *connection oriented* terdiri dari:

```
Deklarasi koneksi
try {
       Permintaan koneksi ke database
       Jalankan perintah SQL
       Proses hasil dari perintah SQL
       Lepaskan koneksi
} catch ( Exception ) {     Handle exception }
finally { try { tutup koneksi} catch (Exception)
  { Handle exception  }}
```

Contoh program yang menerapkan pola umum diatas adalah:

```
using System;
using System.Data;
using System.Data.OleDb;

public class EmployeeReader
{
    public static void Main()
    {
        string connStr = "provider=SQLOLEDB;data source=
              DWI-PC\\SQLEXPRESS;initial catalog=samples;user
              id=sa; Trusted_Connection=true";
        IDbConnection con = null;
        try
        {
            con = new OleDbConnection(connStr);
            con.Open();
            IDbCommand cmd = con.CreateCommand();
            cmd.CommandText = "SELECT EmployeeID, LastName,
                              FirstName FROM Employees";
            IDataReader reader = cmd.ExecuteReader();
            object[] dataRow = new object[reader.FieldCount];

            while (reader.Read())
            {
                int cols = reader.GetValues(dataRow);
                for (int i = 0; i < cols; i++)
                    Console.Write("| {0} ", dataRow[i]);
                Console.WriteLine();
            }
```

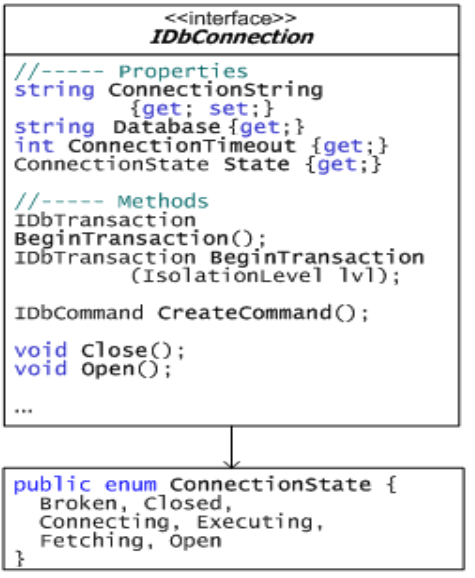
```
            reader.Close();
        }
         catch (Exception e)
         {  Console.WriteLine(e.Message);   }
         finally
         {try
             { if (con != null)    con.Close(); }
         catch (Exception ex) { Console.WriteLine(ex.Message); }
         }
         Console.ReadKey();
    }
}
```

14.2.1. Kelas interface IDbConnection

Sebuah IdbConnection merupakan representasi dari koneksi ke sumber data yang diimplementasikan oleh kerangka kerja penyedia .NET untuk akses data pada database. Sebuah Antarmuka (*interface)* IDbConnection memungkinkan kelas untuk mewarisi dan menerapkan kelas Connection. Sebuah aplikasi tidak menciptakan sebuah instance dari interface IDbConnection secara langsung, tetapi menciptakan sebuah instance dari kelas yang turunan dari IDbConnection.

Kelas yang mewarisi IDbConnection harus mengimplementasikan semua anggota yang diwariskan, dan biasanya menentukan anggota tambahan untuk menambahkan fungsi khusus operator. Sebagai contoh, interface IDbConnection mendefinisikan property ConnectionTimeout, maka sebuah kelas SqlConnection akan ikut pula mewarisi properti tsb

Kelas interface *connection* secara umum terdiri dari property dan metode yang diperlihatkan pada gambar 14.5 dibawah ini:



Gambar 14.5 Kelas interface connection

Pada gambar 14.5, terlihat beberapa property dan metode yang masing-masing berfungsi antara lain:

- Koneksi string berfungsi untuk mendefinsikan koneksi ke database yaitu `string ConnectionString {get; set;}`
- Buka dan tutup koneksi yaitu `void Open(); void Close();`
- Properti dari obyek koneksi antara lain: `string Database {get;},int ConnectionTimeout {get;},ConnectionState State {get;}`
- Pembuatan obyek command yaitu `IDbCommand CreateCommand();`
- Pembuatan obyek transaction yaitu

Properti dari connectionString berisi sebuah informasi lengkap dari penyedia data, setiap informasi dipisahkan oleh semikolom (;). Biasanya konfigurasi ini terdiri dari, Nama penyedia, identifikasi dari sumber data, user dan setting-setting tertentu.

Contoh untuk penyedia OLEDB, untuk koneksi ini antara lain:

- SQL Server :

```
"provider=SQLOLEDB; data source=127.0.0.1\\NetSDK;
initial catalog=Northwind; user id=sa; password=; "
```

- MS Access:

```
"provider=Microsoft.Jet.OLEDB.4.0;data
source=c:\bin\LocalAccess40.mdb;"
```

- Oracle:

```
"provider=MSDAORA; data source=ORACLE8i7; user id=OLEDB;
password=OLEDB;“
```

Contoh penyedia MS-SQL-Server, yaitu

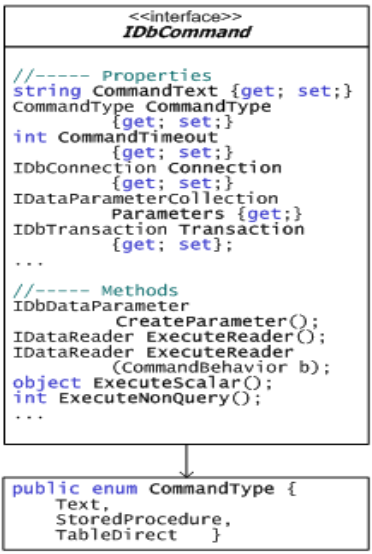
```
"data source=(local)\\NetSDK; initial catalog=Northwind;
user id=sa;pooling=false; Integrated Security=SSPI;
connection timout=20;"
```

14.2.2.Kelas interface IdbCommand

Sebuah IdbCommand merupakan representasi dari perintah SQL yang dieksekusi ke sumber data. Kelas ini diimplementasikan oleh kerangka kerja penyedia .NET untuk akses data pada database. Sebuah antarmuka IDbCommand memungkinkan instan sebuah kelas untuk mewarisi kelas command, yang merupakan pernyataan SQL yang dijalankan pada sumber data. Sebuah aplikasi tidak menciptakan sebuah instance dari interface IDbCommand seacar langsung, tetapi menciptakan sebuah instance dari kelas yang mewarisi IDbCommand.

Kelas yang mewarisi IDbCommand harus mengimplementasikan semua anggota diwariskan, dan biasanya menentukan anggota tambahan untuk menambahkan fungsi khusus operator. Sebagai contoh, interface IDbCommand mendefinisikan metode ExecuteNonQuery metode, maka kelas SqlCommand akan mewarisi pula metode ini.

Kelas interface *command* secara umum terdiri dari property dan metode yang diperlihatkan pada gambar 14.6 dibawah ini:



Gambar 14.6 Kelas interface command

Pada gambar 14.6, terlihat beberapa property dan metode yang masing-masing berfungsi antara lain:

- Perintah text yang mendefinisikan pernyataan SQL atau sebuah store procedure: `string CommandText {get; set;}`
- Obyek koneksi yang mendefinisikan koneksi ke sumber data: `IDbConnection Connection {get; set;}`
- Properti tipe perintah dan setting waktu tenggat: `CommandType CommandType {get; set;}, int CommandTimeout {get; set;}`
- Pembuatan dan pengaksesan parameter: `IDbDataParameter CreateParameter(); IDataParameterCollection Parameters {get;}`
- Eksekusi perintah : `IDataReader ExecuteReader(); IDataReader ExecuteReader(CommandBehavior b); int ExecuteNonQuery(); object ExecuteScalar();`

Contoh program untuk sebuah fungsi yang membaca data kemudian ditampilkan di layar:

```
private static void ReadOrderData(string connectionString)
    {
        string queryString =
        "SELECT OrderID, CustomerID FROM dbo.Orders;";
        using (SqlConnection connection = new SqlConnection(
        connectionString))
        {
            SqlCommand command = new SqlCommand(queryString,
                                    connection);
            connection.Open();
            SqlDataReader reader = command.ExecuteReader();
            try
            {
                while (reader.Read())
                {
                    Console.WriteLine(String.Format("{0}, {1}",
                  reader[0], reader[1]));
                }
            }
            finally { reader.Close(); }
        }
```

Pada interface kelas ini, terdapat suatu metode *ExecuteReader* yang berperan untuk mengeksekusi perintah text yang telah diberikan dan hasilnya dipopulasikan ke sebuah dataReader. Metode *ExecuteReader* memiliki varian atau cara pemanggilan dengan parameter dan tanpa paremeter, yaitu `IDataReader ExecuteReader()` dan `IDataReader ExecuteReader( CommandBehavior behavior );` Sebuah parameter diberikan pada metode ini, merupakan sebuah nilai enumerasi dari nilai konstanta yang ditunjukkan dibawah ini:

```
public enum CommandBehavior {

CloseConnection,      Default,      KeyInfo,      SchemaOnly,
SequentialAccess, SingleResult, SingleRow }
```

Contoh penggalan pemakaian metode IDataReader ExecuteReader() :

```
cmd.CommandText = "SELECT EmployeeID, LastName, FirstName
                    FROM Employees ";
IDataReader reader = cmd.ExecuteReader();
```

Pada interface kelas ini, juga terdapat suatu metode ExecuteNonQuery yang berfungsi untuk mengeksekusi perintah yang ada dalam sebuah *commandtext* . Metode ini menghasilkan nilai integer sebagai nilai dari jumlah baris yang terpengaruh dari eksekusi ini: `int ExecuteNonQuery();`

Secara umum eksekusi non query adalah sebuah perintah berupa update, insert, delete atau create table. Contoh sederhana dari metode ini adalah:

```
cmd.CommandText = "UPDATE Empls SET City = 'Seattle'
                    WHERE iD=8";
int affectedRows = cmd.ExecuteNonQuery();
```

Metode lain pada kelas interface ini adalah *ExecuteScalar* yang berfungsi untuk mengeksekusi perintah text yang berupa nilai aggregasi, sehingga hanya 1 baris saja hasilnya. Contoh sederhana dari metode ini:

```
cmd.CommandText = " SELECT count(*) FROM Employees ";
int count = (int) cmd.ExecuteScalar();
```

14.2.3. Parameter

Suatu parameter sebuah obyek *command* yang mengijinkan adanya parameter sebagai input atau outputnya. Bentuk dari deklarasi pada kelas interface IDbCommand yang mewarisi dari kelas interface IDataParameter: `IDataParameterCollection Parameters {get;}`

Suatu obyek command memiliki parameter dengan atribut antara lain: Name (nama dari parameter), Value (nilai dari parameter), DbDataType (tipe data dari parameter), Direction

(pilihan tujuan dari parameter, sebagi Input , Output , InputOutput atau ReturnValue)

Langkah-langkah memakai parameter:

1. Tentukan sebuah perintah SQL dengan isi didalamnya sebagai parameter dengan notasi tertentu yang bergantung pada jenis providernya (OLE DB memakai notasi '?' dan SQL Server memakai notasi @name) Contoh pemakaian:

   a. Memakai OLE DB

```
OleDbCommand cmd = new OleDbCommand();
cmd.CommandText = "DELETE FROM Empls
WHERE EmployeeID = ?";
```

   b. Memakai SQL Server:

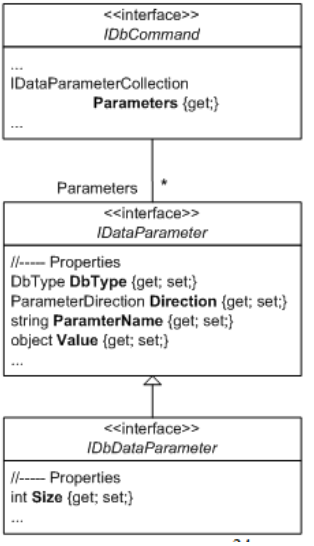
```
SqlCommand cmd = new SqlCommand();
cmd.CommandText = "DELETE FROM Empls
WHERE EmployeeID = @ID";
```

2. Buat parameter dan isi parameter tsb dengan sebuah nilai yang diinginkan, contohnya:

```
cmd.Parameters.Add( new OleDbParameter("@ID",
OleDbType.BigInt));
```

3. Tetapkan nilai parameter tsb dan jalankan perintah sebagai eksekusi non query.

```
cmd.Parameters["@ID"].Value = 1234;
cmd.ExecuteNonQuery();
```



Gambar 14.7 Kelas interface Parameter

14.2.4. Transaksi

Suatu transaksi terdiri dari satu perintah atau sekelompok perintah yang dijalankan dalam sebuah paket. Transaksi memungkinkan untuk menggabungkan beberapa operasi ke dalam satu unit kerja. Jika transaksi mengalami kegagalan pada satu titik dalam transaksi, semua status update dapat digulung kembali ke keadaan pra-transaksi mereka (*rollback)*.

Sebuah transaksi harus sesuai dengan sifat atomicity ACID (atomicity, consistency, isolation, and durability)-, konsistensi, isolasi, dan daya tahan-dalam rangka untuk menjamin konsistensi data. Kebanyakan sistem database relasional, seperti Microsoft SQL Server, menyediakan dukungan terhadap transaksi dengan menyediakan fasilitas penguncian, pencatatan dan fasilitas manajemen transaksi setiap kali aplikasi client melakukan update, insertan atau menghapus operasi.

Transaksi yang melibatkan beberapa sumber daya dapat menurunkan concurrency jika dikunci terlalu lama. Oleh karena itu, transaksi harus dilakukan sesingkat mungkin.

Jika sebuah transaksi melibatkan berbagai tabel dalam database yang sama atau server, transaksi yang tertulis secara eksplisit dalam prosedur yang tersimpan pada database ybs, sering melakukan lebih baik. Hal ini di sebut sebagai *store procedure* sebagai contoh transaksi di SQL Server, pembuatan *store procedure* dengan menggunakan Transact-SQL BEGIN TRANSAKSI, TRANSAKSI COMMIT, dan laporan TRANSAKSI ROLLBACK. Transaksi yang melibatkan manajer sumber daya yang berbeda, seperti transaksi antara SQL Server dan Oracle, memerlukan transaksi terdistribusi.

Kelas interface transaksi di warisi oleh kelas interface command, seperti yang ditunjukkan pada gambar 14.8

Langkah-langkah untuk membuat suatu transaksi secara umum mengikuti pola antara lain:

1. Menentukan koneksi dan membuat obyek transaksi seperti yang dicontohkan dibawah ini:

```
SqlConnection con = new SqlConnection(connStr);
IDbTranaction trans = null;
try {con.Open();
 trans =
 con.BeginTransaction(IsolationLevel.ReadCommitted);
```

2. Membuat obyek *command* dan menetapkan obyek tsb ke obyek *transaction* kemudian meng-eksekusinya, seperti dibawah ini:

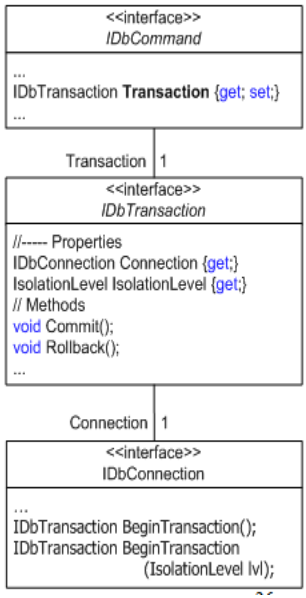
```
IDbCommand cmd1 = con.CreateCommand();
cmd1.CommandText = "DELETE [OrderDetails] WHERE
                    OrderId = 10258";
cmd1.Transaction = trans;
cmd1.ExecuteNonQuery();

IDbCommand cmd2 = con.CreateCommand();
cmd2.CommandText = "DELETE Orders WHERE
```

```
OrderId =     10258";
cmd2.Transaction = trans;
cmd2.ExecuteNonQuery();
```

3. Melakukan (*commit)* atau membatalkan (*rollback*) transaksi tsb, seperti kelanjutan contoh dibawah ini:

```
trans.Commit();
catch (Exception e) {
       if (trans != null)
              trans.Rollback();} finally {
       try {  con.Close();}
}
```



Gambar 14.8 Kelas interface Transaksi

Pada sebuah transaksi terdapat tipe dari tingkat isolasi dari sebuah data, tingkat isolasi ini mempengaruhi terhadap operasi pembacaan dan penulisan data. Tingkat isolasi diberikan dalam sebuah nilai enumerasi antara lain:

```
public enum IsolationLevel {
ReadUncommitted, ReadCommitted, RepeatableRead, Serializable,
... }
```

Tipe masing –masing isolasi dijelaskan pada table 14.2, dibawah ini:

Tabel 14.2 Jenis Isolasi

ReadUncommitted	Mengijinkan membaca data yang lock, mungkin juga membaca data yang belum sempurna
ReadCommitted (Standard)	Membaca data yang belum sempurna tidak diperbolehkan. Hanya data-data yang sudah tersimpan secara permanen ke database yang boleh dibaca
RepeatableRead	Bersifat mirip dengan ReadCommitted namun diperbolehkan membaca secara berulang-ulang
Serializable	Akses ke data secara Serialized, sehingga persoalan perbedaan jumlah data tidak akan terjadi