



2008

.NET Enterprise Application Programming

using patterns and framework

Buku ini membahas tentang penggunaan Patterns dan Framework untuk membantu mengembangkan aplikasi skala Enterprise di platform .NET yang harapannya adalah efisiensi dari segi waktu untuk pengembangan perangkat lunak tanpa harus mengorbankan desain arsitektur yang baik dan elegan.

Ariyanto
.NET Expert
8/17/2008



“Simple thing must be simple, complex thing must be possible”

KATA PENGANTAR

Ketika membaca *masterpiece* Stephen Hawking “*A Brief History of Time*”, penulis sangat tercengang oleh buku karangan ahli Fisika teoritis yang konon dianggap terbesar setelah Einstein tersebut. Betapa tidak, pada kondisinya yang sangat memprihatinkan karena terserang penyakit ALS, Hawking mampu memberikan kontribusi yang sangat berarti bukan saja bagi masyarakat ilmiah pada umumnya, namun masyarakat awam pun dapat menikmati halaman demi halaman penciptaan alam semesta melalui bukunya itu. Bukan bermaksud untuk menyaingi Hawking, penulis terketuk untuk ikut memberikan sedikit sumbangan yang berarti khususnya di bidang Teknologi Informasi, sesuai dengan *background* penulis yang selama ini mengenyam pendidikan di Teknik Informatika dan akhirnya terjun sebagai praktisi dan akademisi dibidang ini selama hampir 5 tahun lebih.

Dari pengalaman penulis, mengembangkan sebuah software yang baik ternyata tidak semudah membalikan telapak tangan. Apalagi jika software yang kita kembangkan termasuk katagori Enterprise Application. Sebetulnya sulit memberikan defenisi aplikasi Enterprise itu apa, bahkan Martin Fowler tidak berani untuk membuat defenisi tentang ini. Beliau hanya memberi contoh apa itu aplikasi enterprise dalam bukunya “*Pattern of Enterprise Application Architecture*”. Menurut beliau aplikasi skala enterprise itu harus memiliki ciri-ciri sebagai berikut : berhubungan dengan database, banyak data yang diolah, banyak pengakses data secara kongkuren, banyak user interface yang terlibat, dan terintegrasi dengan aplikasi lain.

Pada buku ini penulis akan membahas penggunaan Patterns dan Framework untuk membantu mengembangkan aplikasi skala Enterprise di platform .NET. Dengan menggunakan Patterns dan Framework, diharapkan waktu pengembangan software jadi lebih cepat tanpa harus mengorbankan desain arsitektur yang baik dan elegan. Untuk buku ini penulis menggunakan framework yang penulis kembangkan sendiri yaitu : Origami Framework yang dapat didownload di www.codeplex.com/origami.

Ucapan terima kasih tidak lupa penulis sampaikan kepada semua pihak yang telah membantu terbitnya buku ini. Buat Mas Narenda Wicaksono dari Microsoft Indonesia, yang telah berkenan mempublikasikan e-book ini. Tidak lupa saya sampaikan ucapan terimakasih pada istri saya tercinta, Erika Kartawidjaja yang selalu memberi dukungan dan cintanya setiap hari, yang merelakan jatah waktu saya untuk nya demi menyelesaikan buku ini.

Tidak ada karya manusia yang sempurna, hanya kreasi-Nya lah yang tak bercela. Segala kritik, dan saran bisa dialamatkan ke e-mail saya neonerdy@yahoo.com. Jika ada pujian lebih baik dialamatkan ke para master yang bukunya penulis jadikan referensi di daftar pustaka. Seperti kata Issac Newton, “Kita bisa memandang lebih luas alam semesta karena duduk di pundak raksasa”

Bogor, Agustus 2008

Ariyanto

DAFTAR ISI

BAB 1 Pengenalan ADO.NET	1
1.1 .NET Data Provider	2
1.2 Menggunakan ADO.NET	3
1.3 DataSet	9
1.4 Transaction	10
1.5 Stored Procedure	10
BAB 2 ADO.NET Lanjut	12
2.1 Generic Data Access	12
BAB 3 3-Tier Architecture	16
3.1 Business Entity	18
3.1.1 Apa itu Class?	19
3.1.2 Constructor	20
3.1.3 Property	21
3.2 Data Access Object	24
3.3 Object Mapping	32
3.4 Relasi Objek	38
3.4.1 Asosiasi	38
3.4.2 Agregasi	38
3.4.3 Generalisasi	38
3.4.4 Depedency	39
3.4.5 Realisasi	39
BAB 4 Design Pattern	49
4.1 DAO Pattern	49
4.1.1 Apa itu DAO Pattern	50
BAB 5 Origami Framework	57
5.1 Data Access Abstraction	58
5.1.1 Data Access Helper	58
5.1.2 Object Mapping	63
5.1.3 Command Wrapper	65
5.1.4 Query Builder	67
5.2 Dependency Injection	70
5.2.1 Programmatic Injection	72
5.2.2 Declarative Injection	74
5.2.3 Data Access Abstraction Menggunakan Dependency Injection	75
BAB 6 Object Relational Mapping (ORM)	77
6.1 Apa itu Object Persistence?	77
6.2 Arsitektur Origami ORM	78
6.3 Perbandingan ADO.NET dengan ORM	79
6.4 Dasar Mapping	80
6.5 Identitas Objek	86
6.6 Membuat Koneksi ke Database	86
6.7 Operasi CRUD (Create, Read, Update, Delete)	88

6.8 Transaction.....	91
6.9 Query.....	92
6.9.1 Criteria Query	92
6.9.2 Native Query.....	97
6.9.3 Named Native Query	99
6.9.4 Aggregate Query	102
6.10 Stored Procedure	105
6.11 Relationship	107
6.11.1 One to One	116
6.11.2 Many to One	117
6.11.3 One to many	118
6.12 Query pada Relationship.....	119
6.13 Data Binding	121
6.14 Meta Data.....	123
6.15 Object Serialization.....	124
6.16 ORM dan DAO Pattern.....	125
BAB 7 Integrasi Dengan LINQ	132
BAB 8 Unit Testing.....	140
8.1 Tahapan Pengembangan Software	140
8.2 Unit Testing	142
BAB 9 Logging.....	151
9.1 Logging Menggunakan Origami Container	153
9.2 DAO Logging	154
BAB 10 Security	157
10.1 Cryptography	157
BAB 11 Contoh Kasus.....	160
Lampiran.....	181
Meng-Import File Database ke SQL Server Express 2005.....	181
Meng-Import Framework Origami ke Project di Visual C# 2005/2008 Express	181
Daftar Attribute	185
Konvensi dan Best Practice	186
Konfigurasi Untuk Berbagai Macam DBMS.....	187
Biografi Penulis.....	189

BAB 1

Pengenalan ADO.NET

ADO.NET merupakan NET library sebagai bagian dari .NET Framework yang bertanggung jawab untuk memberikan kemudahan dalam mengakses basis data secara universal yang tidak tergantung oleh jenis basis data nya. ADO.NET menyediakan kumpulan class-class yang tergabung dalam beberapa namespace. Namespace adalah pengelompokan secara logic class-class kedalam nama tertentu. Tiap jenis basis data memiliki namespace yang unik yang terdiri dari class-class spesifik

Untuk DBMS MS Access namespace yang digunakan adalah `System.Data.OleDb` dan untuk SQL Server adalah `System.Data.SqlClient`

Kumpulan class-class dalam namespace tersebut :

Fungsi	Namespace	
	<code>System.Data.OleDb</code>	<code>System.Data.SqlClient</code>
Membuka Koneksi	<code>OleDbConnection</code>	<code>SqlConnection</code>
Mengeksekusi perintah SQL	<code>OleDbCommand</code>	<code>SqlCommand</code>
Membaca record secara forward only	<code>OleDbDataReader</code>	<code>SqlDataReader</code>
Penghubung Ke DataSet	<code>OleDbDataAdapter</code>	<code>SqlDataAdapter</code>

Namespace ini harus selalu dipanggil dengan menggunakan keyword `using`
contoh :

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

atau

```
using System.Data.SqlClient
```

Jika namespace tersebut tidak dipanggil maka class-class yang berada dalam namespace yang bersangkutan tidak bisa digunakan Selain MS Access dan SQL Server ADO.NET menyediakan pula library untuk database lainnya dalam sebuah .NET Data Provider.

1.1 .NET Data Provider

ADO.NET menyediakan sekumpulan objek yang bertanggung jawab mengkoneksikan diri ke database dan manipulasi data seperti INSERT, UPDATE, dan DELETE (Connection, Command, DataReader, DataAdapter). ADO.NET terdiri dari beberapa provider sesuai dengan tipe database nya :

- **SQL Server.NET Data Provider (System.Data.SqlClient)**
Provider untuk SQL Server versi 7.0 ke atas. Provider ini dioptimasi khusus untuk SQL Server yang mengakses secara langsung native data transfer protocol SQL Server.
- **Oracle.NET data Provider (System.Data.OracleClient)**
Provider untuk Oracle versi 8.17 ke atas. Memungkinkan mengkoneksikan ke database Oracle dengan menggunakan Oracle client connectivity
- **OleDb.NET Data Provider (System.Data.OleDb)**
Provider untuk OleDb data source. Database yang menggunakan provider ini adalah MS Access, SQL Server 6.5, dan database lain yang menggunakan OleDb
- **ODBC.NET (System.Data.Odbc)**
Provider untuk database yang hanya memiliki driver ODBC

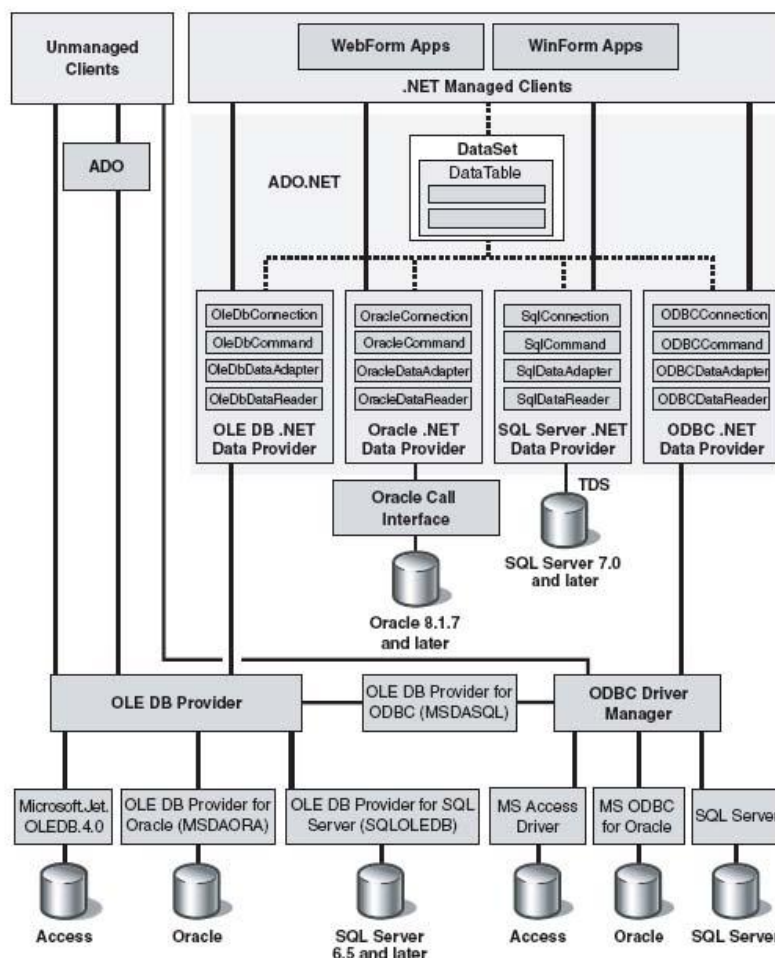


Figure 1.2
Data access stack

Class-class yang terdapat di masing-masing .NET Data Provider

Objek	Contoh	Keterangan
Connection	OleDbConnection, SqlConnection, OracleConnection, OdbcConnection	Membuka koneksi ke database
Command	OleDbCommand, SqlCommand, OracleCommand, OdbcCommand	Mengeksekusi perintah SQL
DataReader	OleDbDataReader, SqlDataReader, OracleDataReader, OdbcDataReader	Membaca data secara read only dan forward only
DataAdapter	OleDbDataAdapter, SqlDataAdapter, OracleDataAdapter, OdbcDataAdapter	Penghubung ke DataSet

1.2 Menggunakan ADO.NET

Ketika kita berhubungan dengan basis data menggunakan ADO.NET “ritual” nya adalah sebagai berikut :

1. Membuka Koneksi

Hal pertama yang harus dilakukan untuk membuat aplikasi database adalah membuka koneksi. Koneksi ini selalu diperlukan untuk menjalankan operasi-operasi manipulasi data seperti INSERT, UPDATE dan DELETE ataupun meng-query record tertentu dengan perintah SELECT. Class yang bertugas untuk membuka koneksi adalah `OleDbConnection` untuk MS Access, dan `SqlConnection` untuk SQL Server.

`OleDbConnection` dan `SqlConnection` membutuhkan property `ConnectionString` yang harus diisi. Selain lewat property `Connection String` bisa ditambahkan juga sebagai parameter constructor class. `Connection String` tiap database berbeda-beda. `Connection String` ini menunjukan provider DBMS yang digunakan beserta letak direktori dan nama database nya baik lokal maupun remote database

Contoh source code untuk buka koneksi :

MS Access :

```
string connStr="Provider=Microsoft.Jet.OleDb.4.0;Data Source=c:\\NWIND.mdb";
OleDbConnection conn=new OleDbConnection(connStr);
conn.Open();
```


atau bisa seperti ini :

```
string connStr="Provider=Microsoft.Jet.OleDb.4.0;Data Source=c:\\NWIND.mdb";
OleDbConnection conn=new OleDbConnection();
conn.ConnectionString=connStr;
conn.Open();
```

SQL Server :

```
string connStr="Data Source=SYSTEMINTERFACE\\SQLEXPRESS;Initial Catalog=NWIND;"
               + "Integrated Security=True";

SqlConnection conn=new SqlConnection(connStr);
conn.Open();
```

Data Source pada ConnStr diisi nama server lokal atau bisa diisi IP Address jika menggunakan server remote.

2. Eksekusi Perintah SQL

Setelah koneksi terbuka, langkah berikutnya adalah memanggil perintah SQL yang akan dieksekusi. SQL (*Structure Query Language*) adalah bahasa yang digunakan untuk berkomunikasi dengan DBMS. Tanpa perintah SQL user tidak dapat melakukan apapun terhadap Database.

Perintah SQL berbeda dengan bahasa pemrograman, SQL sifatnya deklaratif dan lebih mudah dimengerti secara sintaksis karena menggunakan bahasa sehari-hari (bahasa Inggris). SQL tidak memiliki control flow dan deklarasi variabel seperti halnya bahasa pemrograman. Untuk menangani hal tersebut SQL bisa digabungkan dengan bahasa pemrograman tertentu sehingga bisa lebih dinamis.

Secara umum perintah SQL terbagi 3, yaitu :

DDL (Data Definition Language)

Perintah untuk membuat Database baru, dan manipulasi objek-objek terkait yang ada dalam database tersebut. Contoh : CREATE TABLE, DROP TABLE, ALTER TABLE

DML (Data Manipulation Language)

Berhubungan dengan query dan manipulasi table di database. Contoh : SELECT, INSERT, UPDATE, DELETE

DCL (Data Control Language)

Perintah yang berhubungan dengan manajemen user. Perintah DCL ini hanya tersedia untuk database bertipe client-server yang penggunaannya lebih dari 1 user. Contoh : GRANT, REVOKE

Class yang digunakan untuk mengeksekusi perintah SQL adalah `OleDbCommand` pada MS Access atau `SqlCommand` pada SQL Server. `OleDbCommand` dan `SqlCommand` memiliki parameter constructor perintah sql dan Connection.

Contoh :

Class `OleDbCommand` memiliki method utama `ExecuteNonQuery()` dan `ExecuteReader()` method `ExecuteNonQuery()` digunakan untuk mengeksekusi perintah SQL manipulasi data seperti INSERT, UPDATE, dan DELETE. Method `ExecuteReader()` digunakan hanya untuk perintah sql SELECT, return value dari method tersebut adalah objek `OleDbDataReader` atau `SqlDataReader`

INSERT

MS Access :

```
string sql="INSERT INTO Customers (CompanyName,Address) VALUES ("
        + "'XERIS','Sentul Valley')";

OleDbCommand cmd=new OleDbCommand(sql,conn);
cmd.ExecuteNonQuery();
```

atau :

```
string sql="INSERT INTO Customers (CompanyName, Address) VALUES "
        + "('XERIS','Sentul Valley')";

OleDbCommand cmd=new OleDbCommand();
cmd.Connection=conn;
cmd.CommandText=sql;

cmd.ExecuteNonQuery();
```

SQL Server :

```
string sql="INSERT INTO Customers (CompanyName,Address) VALUES "
        + "('XERIS','Sentul Valley')";

SqlCommand cmd=new SqlCommand(sql,conn);
cmd.ExecuteNonQuery();
```

atau :

```
string sql="INSERT INTO Customers (CompanyName,Address) VALUES "
        + "('XERIS','Sentul Valley')";

SqlCommand cmd=new SqlCommand();
cmd.Connection=conn;
```

```
cmd.CommandText=sql;
cmd.ExecuteNonQuery();
```

UPDATE

MS Access :

```
string sql="UPDATE Customers SET CompanyName='XERIS System',Address='BOGOR'"
+ "WHERE CustomerID=1";

OleDbCommand cmd=new OleDbCommand(sql,conn);
cmd.ExecuteNonQuery();
```

SQL Server :

```
string sql="UPDATE Customers SET CompanyName='XERIS System',Address='BOGOR'"
+ "WHERE CustomerID=1";

SqlCommand cmd=new SqlCommand(sql,conn);
cmd.ExecuteNonQuery();
```

DELETE

```
string sql="DELETE Customers WHERE CustomerID='XERIS'";
OleDbCommand cmd=new OleDbCommand(sql,conn);
cmd.ExecuteNonQuery();
```

SQL Server :

```
string sql="DELETE Customers WHERE CustomerID='XERIS'";
SqlCommand cmd=new SqlCommand(sql,conn);
cmd.ExecuteNonQuery();
```

3. Menampilkan Data

Untuk menampilkan data di ADO.NET class yang digunakan adalah `OleDbDataReader` dan `SqlDataReader`. `DataReader` ini sifatnya forward-only artinya hanya bisa baca maju ke depan, tidak bisa `Move previous` atau `Move Next` seperti pada ADO classic.

MS Access

```
string sql="SELECT * FROM Customers";
OleDbCommand cmd=new OleDbCommand(sql,conn);
```

```
OleDbDataReader rdr=cmd.ExecuteReader();

while (rdr.Read())
{
    Console.WriteLine(Convert.ToInt32(rdr["CustomerID"]));
    Console.WriteLine(Convert.ToString(rdr["CompanyName"]));
}

rdr.Close();
cmd.Dispose();
```

SQL Server

```
string sql="SELECT * FROM Customers";
SqlCommand cmd=new SqlCommand(sql,conn);
SqlDataReader rdr=cmd.ExecuteReader();

while (rdr.Read())
{
    Console.WriteLine(Convert.ToInt32(rdr["CustomerID"]));
    Console.WriteLine(Convert.ToString(rdr["CompanyName"]));
}

rdr.Close();
cmd.Dispose();
```

Pemanggilan method `ExecuteReader()` menghasilkan semua record tabel Customers dan disimpan di objek `OleDbDataReader/SqlDataReader`. Untuk menampilkan record di object tersebut gunakan statement looping `while`. Jika data tersebut ada maka tampilkan di Console dengan perintah `Console.WriteLine()`. `Convert` berfungsi untuk mengubah objek ke tipe yang bersesuaian dengan tipe field di table nya

4. Pencarian Data

Salah satu operasi yang paling banyak dilakukan di database adalah operasi pencarian/seraching. Untungnya DBMS menyediakan kemampuan pencarian tersebut di SQL dengan hanya menambahkan clausa `WHERE`

Contoh :

Jika Anda ingin mendapatkan data Customer yang alamatnya di Bogor cukup seperti ini

```
SELECT * FROM Customers WHERE Address='Bogor'
```

Pencarian dapat dilakukan lebih luas lagi dengan menggunakan operator `LIKE`

```
SELECT * FROM Customers WHERE ContactName LIKE '%dewi%'
```

Jika diketahui datanya sebagai berikut

No	ContactName	Address	Email
1	Sandra Dewi	Jakarta	sandradewi@seleb.com
2	Dewi Sandra	Jakarta	dewisandra@seleb.com
3	Dewi Persik	Jakarta	dewipersik@seleb.com
4	Dewi-Dewi	Jakarta	dewidewi@seleb.com
5	Cinta Laura	Jakarta	cintalaura@seleb.com

query `SELECT * FROM Customers WHERE ContactName LIKE '%dewi%'` (% didepan dan dibelakang) akan menghasilkan record no 1,2,3, dan 4 perintah sql tersebut mencari semua data di tabel customer yang mengandung kata 'dewi' baik nama depan, belakang, maupun tengah

query `SELECT * FROM Customers WHERE ContactName LIKE '%dewi'(% didepan)` akan mencari semua data di tabel Customers yang nama belakangnya mengandung kata 'dewi'. Record yang dihasilkan adalah no 1 dan 4

query `SELECT * FROM Customers WHERE ContactName LIKE 'dewi%'(% dibelakang)` akan mencari semua data di tabel Customers yang nama depannya mengandung kata 'dewi', data yang dihasilkan adalah no 2,3, dan 4.

Lalu bagaimana dengan 'Cinta Laura', bagaimana cara menampilkannya?(jangan khawatir cin,kamu pasti bisa tampil :-)) untuk menampilkan cinta laura bisa 3 cara

```
SELECT * FROM Customers WHERE ContactName ='Cinta Laura'
SELECT * FROM Customers WHERE ContactName LIKE '%cinta%'
```

atau

```
SELECT * FROM Customers WHERE ContactName NOT LIKE '%dewi%'
```

hmmm...cara terakhir sepertinya dipaksakan sekali :-)

Contoh Program :

MS Access

```
string cari="dewi";

string sql="SELECT * FROM Customers WHERE ContactName LIKE '%" + cari + "%'";
OleDbCommand cmd=new OleDbCommand(sql,conn);
OleDbDataReader rdr=cmd.ExecuteReader();

while (rdr.Read())
{
    Console.WriteLine(Convert.ToInt32(rdr["CustomerID"]));
    Console.WriteLine(Convert.ToString(rdr["Nama"]));
}

rdr.Close();
cmd.Close();
```

SQL Server

```
string cari="dewi";

string sql="SELECT * FROM Customers WHERE ContactName LIKE '%" + cari + "%'";
SqlCommand cmd=new SqlCommand(sql,conn);
SqlDataReader rdr=cmd.ExecuteReader();

while (rdr.Read())
```

```

{
    Console.WriteLine(Convert.ToInt32(rdr["CustomerID"]));
    Console.WriteLine(Convert.ToString(rdr["Nama"]));
}

rdr.Close();
cmd.Dispose();

```

1.3 DataSet

DataSet adalah tabel virtual yang tersimpan di memory. DataSet merupakan fitur unggulan ADO.NET yang menyediakan pengaksesan data bisa dilakukan secara *disconnected*, tanpa harus selalu terhubung ke database.

Keuntungan menggunakan DataSet :

1. **Flexibility**, DataSet mengandung koleksi dari data dan dapat merepresentasikan relasi yang kompleks
2. **Serialization**, DataSet mendukung serialisasi yang biasanya digunakan untuk distributed application
3. **Data Binding**, DataSet dapat di ikatkan ke komponen-komponen yang bersifat “data aware” baik pada WinForm (GUI) atau WebForm (ASP.NET)
4. **Sorting dan Filtering**, DataSet mendukung sorting dan filtering data dengan menggunakan DataView Object.
5. **Interchangeability**, DataSet dapat dibaca dan diubah dalam format XML sehingga data bisa diakses walaupun koneksi sudah terputus (*disconnected application*) .
6. **Optimistic Concurrency**, Ketika melakukan pengupdate-an data DataSet dapat digunakan bersama DataAdapter yang memungkinkan *concurrency check* dilakukan dengan mudah
7. **Extensibility**, Schema dari DataSet bisa diubah secara runtime

Contoh penggunaan DataSet

```

string sql = "SELECT * FROM Customers";

DataSet ds = new DataSet();
SqlDataAdapter adapter = new SqlDataAdapter(sql, conn);
adapter.Fill(ds);

foreach (DataRow row in ds.Tables[0].Rows)
{
    Console.WriteLine(row["CustomerID"]);
    Console.WriteLine(row["CompanyName"]);
}

```

1.4 Transaction

Transaksi didefinisikan sebagai himpunan satu atau lebih pernyataan yang dieksekusi sebagai satu unit, dengan demikian dalam suatu transaksi himpunan pernyataan harus dilaksanakan atau tidak sama sekali. Contoh jika kita ingin menghapus record yang memiliki hubungan master-detail. Proses penghapusan record di tabel master harus disertai dengan record yang berelasi di tabel detail, jika proses penghapusan record pada tabel master gagal proses penghapusan harus dibatalkan seluruhnya agar integritas data tetap terjaga.

Contoh Code

```
string connStr="Data Source=SYSTEMINTERFACE\\SQLEXPRESS;Initial Catalog=NWIND;"
               + "Integrated Security=True";

SqlConnection conn = new SqlConnection(connStr);
conn.Open();

SqlTransaction tx = conn.BeginTransaction();
SqlCommand cmd = new SqlCommand();
cmd.Connection = conn;
cmd.Transaction = tx;

try
{
    cmd.CommandText = "DELETE Orders WHERE OrderID=1";
    cmd.ExecuteNonQuery();
    cmd.CommandText = "DELETE OrderDetails WHERE OrderID=1";
    cmd.ExecuteNonQuery();

    tx.Commit();
}
catch (SqlException ex)
{
    Console.WriteLine(ex.Message.ToString());
    tx.Rollback();
}
```

1.5 Stored Procedure

Stored Procedure adalah procedure yang terletak di DBMS yang dapat dipanggil oleh aplikasi baik secara local ataupun lewat jaringan. Stored Procedure merupakan salah satu fitur utama DBMS bertipe client-server seperti SQL Server dan Oracle. Berikut ini manfaat menggunakan Stored Procedure :

1. Diletakan di server, stored procedure biasanya digunakan untuk menuliskan business logic. Jika terjadi perubahan cukup merubahnya di server.
2. Memiliki keamanan yang lebih baik. Karena diakses di server, aplikasi yang menggunakan stored procedure akan terhindar dari upaya hacking seperti SQL Injection.
3. Membuat program yang lebih modular
4. Dapat menurunkan trafik jaringan

Contoh Stored Procedure

```
CREATE PROCEDURE [dbo].[spInsertCustomer]
    @CustomerID varchar(5)
    @CompanyName varchar(50),
    @Address varchar(200)
AS
BEGIN
    INSERT INTO Customers (CustomerID,CompanyName,Address) VALUES
        (@CustomerID,@CompanyName,@Address)
END
```

Cara mengaksesnya :

```
SqlCommand cmd=new SqlCommand(sql,conn);

cmd.CommandText="spInsertCustomer";
cmd.CommandType=CommandType.StoredProcedure;

DbParameter param1 = cmd.CreateParameter();
param1.ParameterName ="@CustomerID";
param1.Value="XERIS";

DbParameter param2 = cmd.CreateParameter();
param2.ParameterName ="@CompanyName";
param2.Value="XERIS System Interface";

DbParameter param3 = cmd.CreateParameter();
param3.ParameterName ="@Address";
param3.Value="Sentul Valley";

cmd.Parameters.Add(param1);
cmd.Parameters.Add(param2);
cmd.Parameters.Add(param3);

cmd.ExecuteNonQuery();
```


BAB 2

ADO.NET Lanjut

2.1 Generic Data Access

ADO.NET menyediakan beberapa namespace berbeda untuk mengakses database. Seperti yang telah saya jelaskan di Pengenalan ADO.NET, setiap mengubah database kita harus mengubah juga namespacenya misal jika menggunakan database MS Access , maka namespace yang digunakan adalah `System.Data.OleDb`, dan SQL Server : `System.Data.OracleClient`.

Celakanya, bukan hanya namespace yang diubah, class class didalam namespace tersebut juga harus berubah.

Contoh :

<code>OleDbConnection</code>	→	<code>SqlConnection</code>
<code>OleDbCommand</code>	→	<code>SqlCommand</code>
<code>OleDbDataReader</code>	→	<code>OleDbDataReader</code>

Lalu dimanakah permasalahannya?bukankah masing-masing class di namespace tersebut memiliki cara akses yang sama. Dari sudut pandang *Software Engineering*, jika ingin membuat aplikasi yang *scalable* dan dinamis, kita harus memikirkan desain arsitektur yang "tahan banting". Perubahan adalah salah satu hal yang tidak bisa dihindari dalam *Software Development* karenanya kita harus mempertimbangkan berbagai aspek yang kemungkinan bisa terjadi dikemudian hari. Karena fokus kita kali ini di database, saya hanya akan membahas bagaimana cara mensiasati jika dikemudian hari terjadi pergantian DBMS karena alasan tertentu. Salah satu solusi untuk mengatasi perubahan DBMS adalah dengan membuat generic data access agar program tidak tergantung dengan database tertentu. Sehingga jika terjadi pergantian DBMS tidak akan ada perubahan signifikan.

.NET Framework menyediakan namespace `System.Data.Common` untuk pembuatan generic data access. Namespace tersebut memiliki class-class berikut.

- **DbProviderFactory**
Pembuatan instance provider data source spesifik
- **DbConnection**
Membuka koneksi
- **DbCommand**
Eksekusi perintah SQL
- **DbDataReader**
Menampilkan data
- **DbDataAdapter**

Penghubung ke DataSet

Saya hanya akan membahas class-class utama saja dalam namespace tersebut. Class utama dalam namespace tersebut adalah `DbProviderFactory`. Class ini bertanggungjawab terhadap pembuatan instance dari provider data source yang diinginkan.

Contoh :

MS Access

```
string driver = "System.Data.OleDb";
string connStr = "Provider=Microsoft.Jet.OleDb.4.0;Data Source=c:\\NWIND.mdb";

DbProviderFactory factory = DbProviderFactories.GetFactory(driver);
DbConnection conn = factory.CreateConnection();
conn.ConnectionString = connStr;
conn.Open();
```

Data source pada contoh diatas menggunakan MS Access, sehingga namespace yang digunakan adalah `System.Data.OleDb` dan Connection string nya untuk MS Access jika ingin menggantinya ke SQL Server maka yang perlu diubah cukup driver dan connection string nya.

SQL Server

```
string driver = "System.Data.SqlClient";
string connStr = "Data Source= SYSTEMINTERFACE\\SQLEXPRESS;"
               + "Initial Catalog=NWIND;Integrated Security=True";

DbProviderFactory factory = DbProviderFactories.GetFactory(driver);
DbConnection conn = factory.CreateConnection();
conn.ConnectionString = connStr;
conn.Open();
```

Yang menarik disini driver, dan connStr diperlakukan sebagai string, sehingga informasi ini bisa kita simpan dalam bentuk konfigurasi di XML. Dengan demikian jika terjadi perubahan implementasi database program tidak usah di compile ulang, cukup dengan mengubah konfigurasi XML nya. Bayangkan sebuah aplikasi client-server skala enterprise yang client nya bisa jadi terdiri dari banyak sekali komputer, compile ulang dan mendistribusikan kembali program tersebut ke client. Hmm..jujur saja saya tidak akan mau melakukan ini !. Jika kita simpan dalam bentuk XML, file konfigurasi ini bisa disimpan di server. Jika terjadi perubahan cukup mengedit file XML tersebut 1 kali, dan client tidak usah disentuh sama sekali

INSERT

```
DbCommand cmd = conn.CreateCommand();
cmd.Connection = conn;
cmd.CommandText = "INSERT INTO Customers (CustomerID,CompanyName) VALUES "
               + "(@CustId,@Name)";

DbParameter param1 = cmd.CreateParameter();
```

```

param1.ParameterName = "@CustId";
param1.Value = "XERIS";

DbParameter param2 = cmd.CreateParameter();
param2.ParameterName = "@Name";
param2.Value = "XERIS System Interface";

cmd.Parameters.Add(param1);
cmd.Parameters.Add(param2);

cmd.ExecuteNonQuery();

```

UPDATE

```

DbCommand cmd = conn.CreateCommand();
cmd.Connection = conn;
cmd.CommandText = "UPDATE Customers SET CompanyName=@Name "
    + "WHERE CustomerID=@CustId";

DbParameter param1 = cmd.CreateParameter();
param1.ParameterName = "@CustId";
param1.Value = "XERIS";

DbParameter param2 = cmd.CreateParameter();
param2.ParameterName = "@Name";
param2.Value = "XERIS System Interface";

cmd.Parameters.Add(param1);
cmd.Parameters.Add(param2);

cmd.ExecuteNonQuery();

```

DELETE

```

DbCommand cmd = conn.CreateCommand();
cmd.Connection = conn;
cmd.CommandText = "DELETE FROM Customers WHERE CustomerID=@CustId";

DbParameter param1 = cmd.CreateParameter();
param1.ParameterName = "@CustId";
param1.Value = "XERIS";

cmd.Parameters.Add(param1);
cmd.ExecuteNonQuery();

```

SELECT

```

DbCommand cmd = conn.CreateCommand();
cmd.Connection = conn;
cmd.CommandText = "SELECT * FROM Customers WHERE CompanyName LIKE @Name";

DbParameter param1 = cmd.CreateParameter();
param1.ParameterName = "@CustId";

```

```

param1.Value = "XERIS";
cmd.Parameters.Add(param1);

IDataReader rdr = cmd.ExecuteReader();

while (rdr.Read())
{
    Console.WriteLine(rdr["CustomerID"]);
    Console.WriteLine(rdr["CompanyName"]);
}

rdr.Close();

```

Lalu bagaimana kalau konfigurasi data source nya diletakkan di XML? .NET Framework menyediakan akses ke konfigurasi XML lewat namespace `System.Configuration`. Class yang digunakan adalah `ConfigurationManager`

perhatikan contoh berikut :

```

string driver = ConfigurationManager.AppSettings["Driver"];
string connStr = ConfigurationManager.AppSettings["ConnectionString"];

```

Sedangkan konfigurasi XML disimpan di file `App.config`

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="Driver" value="System.Data.SqlClient"/>
    <add key="ConnectionString" value="Data Source=SYSTEMINTERFACE\\SQLEXPRESS;"
      + "Initial Catalog=NWIND;Integrated Security=True"/>
  </appSettings>
</configuration>

```

Agar dapat mengakses konfigurasi XML di `App.config` tambahkanlah library `System.configuration` menggunakan IDE Microsoft Visual C# 2005 Express Edition Anda melalui menu **Project->Add Reference**.

BAB 3

3-Tier Architecture

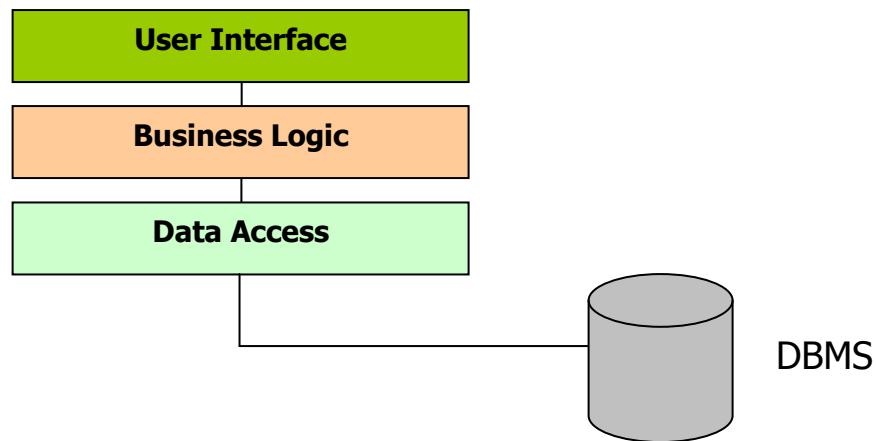
Dalam menghadapi kompleksitas kita perlu menggunakan politik Belanda waktu menjajah Indonesia "*Devide et Impera*", pecah belah lalu kuasai. Ketika membangun aplikasi yang kompleks, ada baiknya kita "memecah belah" aplikasi menjadi 3 layer. Pemisahan ini bisa jadi hanya secara logik (software), tidak harus berkorespondensi secara fisik (hardware). Masih ingat arsitektur TCP/IP yang dibuat berlapis-lapis? kira-kira apa tujuannya? pemisahan arsitektur aplikasi menjadi beberapa layer dimaksudkan untuk meminimalkan terjadinya *ripple effect*. Efek ini terjadi jika sebuah modul sangat tergantung dengan modul yang lain, sehingga jika terjadi perubahan disatu sisi, sisi lain harus ikut diubah. Istilah kerennya adalah *tighty couple*.

Pada pendekatan 3-tier architecture aplikasi ini dipecah menjadi 3 layer yaitu : Presentation, Business Logic dan Data Access layer. Pemecahan ini tidak harus berkorespondensi secara fisik (hardware), bisa jadi hanya secara logik (software).

Presentation layer bertanggung jawab dengan tampilan (user interface), **Business Logic** dengan logika business/domain permasalahan dan **Data Access** bertanggung jawab bagaimana mekanisme akses ke basis datanya. Dengan pemisahan ini aplikasi tidak tergantung dengan user interface nya apa (Console, WinForm, Web/ASP.NET) atau pilihan DBMS (SQL Server, Oracle, MySQL), sehingga apabila terjadi perubahan dikemudian hari karena suatu hal, developer tidak harus menulis ulang program dari awal.

Perbedaan 1-Tier dengan 3-Tier

Classic (1-Tier)	3-Tier
Semua kode (SQL dan C#) diletakkan disatu tempat, yaitu di bagian user interface.	Kode dipecah menjadi 3 layer sesuai dengan fungsi dan tanggung jawabnya
Terjadi pengulangan penulisan program atau syntax SQL, sehingga ada banyak duplikasi kode	Method-method yang sudah dibuat tinggal dipanggil, sehingga class dapat di <i>reusable</i> (penggunaan ulang kembali)
Struktur program tidak teratur	Program lebih modular dan mudah dibaca
Jika terjadi perubahan user interface, maka program harus ditulis ulang	Tidak usah menulis ulang keseluruhan program. Class-class di layer business logic dan Data Access dapat digunakan kembali



3.1 Business Entity

Dalam rekayasa sistem berorientasi objek, segala sesuatunya adalah objek, bukan saja benda atau orang, tapi sebuah proses seperti transaksi pun bisa dianggap sebagai objek. Business Entity adalah objek-objek yang terdapat dalam sebuah problem domain/domain permasalahan disebuah sistem tertentu. Business entity ini dihasilkan dari proses analisis sebuah requirement software. Requirement adalah daftar kebutuhan perangkat lunak dari semua stakeholder yang terlibat dalam pengembangan sistem. Requirement baru menggambarkan "apa" yang ada di sistem, belum mencakup "bagaimana" hal tersebut nantinya dilakukan. Sebagai contoh jika problem domain adalah "Purchasing", maka Business entity yang terlibat adalah Customer, Supplier, Product, dan Order.

Berdasarkan *guidelines* Microsoft Pattern & Practices Group ada 5 cara untuk merepresentasikan Business Entity yaitu : XML, DataSet, Typed DataSet, Business Entity Object, dan CRUD Business Entity Object .

Metode	Keterangan
XML	Format data terbuka yang bisa diintegrasikan dengan beragam aplikasi lain. XML Business entity dapat direpresentasikan dalam bentuk XML Document Object Model (DOM)
DataSet	Cache tabel dalam memori
Typed DataSet	Class yang diturunkan dari ADO.NET yang menyediakan strong method, event, dan property untuk mengakses tabel dan kolom di DataSet
Busines Entity Object	Entity class yang merepresentasikan business entity. Class ini berisi enkapsulasi field, property, dan method.
CRUD Business Entity Object	Entity class yang memiliki kemampuan CRUD (Create, Read, Update, Delete)

Menurut saya pilihan ideal untuk merepresentasikan business entity adalah dengan menggunakan sebuah entity class yang bisa jadi berkorespondensi dengan tabel di Database. Contoh Business entity Customer diimplementasikan dalam class Customer dan bisa berhubungan dengan tabel Customers di database Purchasing. Perlu digaris bawahi business object tidak selalu berhubungan

dengan tabel didatabase, tergantung apakah diperlukan proses penyimpanan dimedia tertentu seperti database atau file. Terkadang proses persistensi ini hanya terjadi di memori.

Contoh Entity Class

Class Customer

```
public class Customer
{
    //deklarasi Field

    private string customerId;
    private string companyName;
    private string contactName;
    private string address;
    private string phone;
}
```

3.1.1 Apa itu Class?

Salah satu teknik yang diperkenalkan OOP adalah Encapsulation atau pengkapsulan. Teknik ini menjamin pembungkusan data beserta operasi-operasinya dalam suatu modul yang disebut class. Saya sangat setuju dengan pendapat **Bertrand Meyer**, yang mengatakan bahwa class adalah modul sekaligus sebuah tipe data. Istilah modul diambil dari paradigma pemrograman terstruktur yang bertujuan mengelompokkan procedure-procedure kedalam kelompok-kelompok tertentu sesuai dengan fungsinya. Class dapat juga dipandang sebagai sebuah blue print/cetak biru yang membentuk sebuah objek. Bisa saya analogikan kalau class adalah cetakan kue maka objek (instance dari class) adalah kue nya (hmmmm ...yummmmy...). Mari kita lihat struktur class Customers.

Setelah deklarasi class ada deklarasi field. Field adalah sebuah variabel yang menjadi atribut suatu class. Field ini memiliki access modifier private, public, dan protected.

private = Hanya dikenali di class tersebut
public = Bisa dikenali dimana saja
protected = Hanya dikenali di class tersebut berikut class turunnanya

setelah Access Modifier variabel terdapat tipe data dari field. Tipe data terbagi dua yaitu primitive type dan composite type. Tipe primitif adalah tipe konvensional yang sudah umum dikenal seperti : int,string,byte dan sebagainya. Sedangkan tipe composite adalah tipe kompleks seperti array dan objek.

3.1.2 Constructor

Sebuah class dapat memiliki 0-n constructor. Constructor ini sifatnya optional, boleh ada boleh juga tidak. Constructor sebetulnya adalah sebuah method special yang akan selalu dieksekusi ketika class diinstantiasi. Penulisan constructor harus sama dengan nama class nya . Jika di class tersebut tidak memiliki constructor maka compiler akan membuatkan constructor default yang tidak memiliki implementasi apapun

```
public class Customer
{
    private string customerId;
    private string companyName;
    private string contactName;
    private string address;
    private string phone;

    //Constructor Default

    public Customer()
    {
    }

    public class Customer
    {
        private string customerId;
        private string companyName;
        private string contactName;
        private string address;
        private string phone;

        //Constructor Default

        public Customer()
        {
        }

        //Constructor Overloading

        public Customer(string customerId, string companyName,
            string contactName, string address, string phone)
        {
            this.customerId = customerId;
            this.companyName = companyName;
            this.contactName = contactName;
            this.address = address;
            this.phone = phone;
        }
    }

    public Customer(string customerId, string companyName,
        string contactName, string address, string phone)
    {
        this.customerId = customerId;
        this.companyName = companyName;
        this.contactName = contactName;
        this.address = address;
    }
}
```

```
        this.phone = phone;
    }
}
```

Class Customer memiliki 2 buah constructor, yang pertama constructor default tanpa parameter dan implementasi, dan constructor kedua memiliki 5 parameter. Keyword `this` menunjukkan object aktif pada saat itu. Jadi `this.customerId` adalah variabel field sedangkan `customerId` setelah sama dengan adalah variabel parameter constructor

Kita bisa memiliki banyak constructor dalam satu class, dengan parameter dan implementasi yang berbeda-beda istilah dalam OOP disebut polymorphism. Dalam polymorphism ini dikenal istilah *overriding* dan *overloading*. Jika constructor kita memiliki parameter yang berbeda disebut overloading, jika memiliki parameter yang sama namun implementasi berbeda disebut overriding. Selain constructor polymorphism juga bisa dikenakan kepada sebuah method.

3.1.3 Property

Sebuah field dalam class memiliki access modifier tertentu, bisa private, public, atau protected. Kita harus bijaksana dalam menggunakan access modifier field ini, jika tidak berhati-hati bisa menyebabkan desain class kita rentan terhadap perubahan dikemudian hari. Salah satu point yang hendak dicapai dalam Object Oriented Desain adalah meminimalkan terjadinya *ripple effect*. Efek ini bisa terjadi jika perubahan class yang satu mengakibatkan perubahan di class lainnya, istilah lainnya adalah "*tightly-coupled*". Demi Encapsulation, variabel/data tidak boleh diekspose langsung, karena itu access modifier nya harus private. Lalu bagaimana cara saya mengakses variabel tersebut dari class lain. Bukankah kalau access modifier nya private hanya dikenal di class tersebut? Variabel hanya boleh diakses lewat method **accessor** dan **mutator** nya. What the #\$%@# istilah apa lagi itu accessor dan mutator? makin memusingkan saja OOP ini!

Kita lihat contoh berikut :

```
public class Customer
{
    //Field
    private string customerId;

    //Accessor

    public string GetCustomerId()
    {
        return customerId;
    }

    //Mutator

    public void SetCustomerId(string customerId)
    {
        this.customerId = customerId;
    }
}
```

Kita punya sebuah field bernama `customerId` dengan tipe `string` dan access modifier `private`. Untuk mengakses nilai variabel `customerId` ini harus lewat method `GetCustomerId()` terlebih dahulu. Jika ingin mengubah nilai `customerId` harus melalui method `SetCustomerId()` dengan parameter `customerId`. Method yang pertama disebut **Accessor** karena fungsinya untuk mengakses nilai sebuah field, sedangkan yang kedua disebut **mutator** karena fungsinya mengubah nilai.

Contoh Penggunaanya :

```
Customer cust = new Customer();
cust.SetCustomerId("XERIS"); //mengubah variabel customerId
Console.WriteLine(cust.GetCustomerId()); //mengakses variabel customerId
```

Di .NET ada cara efisien untuk memadukan **Accessor** dan **mutator** ini dalam satu kesatuan dengan menggunakan sebuah method tunggal yang disebut **Property**.

Contoh Property :

```
public class Customer
{
    //Field

    private string customerId;
    private string companyName;

    //Property

    public string CustomerId
    {
        get { return customerId; }
        set { customerId = value; }
    }

    public string CompanyName
    {
        get { return companyName; }
        set { companyName = value; }
    }
}
```

pada property `CustomerId`, method **Accessor** diwakili oleh `get` dan **mutator** oleh `set`. Lebih simple bukan :-)

Cara Penggunaannya :

```
Customer cust = new Customer();

//memodifikasi nilai variabel customerId

cust.CustomerId = "XERIS";
```

```
//menampilkan nilai variabel customerId  
  
Console.WriteLine(cust.CustomerId);
```

Code Lengkap :

```
public class Customer  
{  
    private string customerId;  
    private string companyName;  
    private string contactName;  
    private string address;  
    private string phone;  
  
    //Constructor Default  
  
    public Customer()  
    {  
  
    }  
  
    //Constructor Overloading  
  
    public Customer(string customerId, string companyName,  
        string contactName, string address, string phone)  
    {  
        this.customerId = customerId;  
        this.companyName = companyName;  
        this.contactName = contactName;  
        this.address = address;  
        this.phone = phone;  
  
    }  
  
    public string CustomerId  
    {  
        get { return customerId; }  
        set { customerId = value; }  
    }  
  
    public string CompanyName  
    {  
        get { return companyName; }  
        set { companyName = value; }  
    }  
  
    public string ContactName  
    {  
        get { return contactName; }  
        set { contactName = value; }  
    }  
  
    public string Address  
    {  
        get { return address; }  
        set { address = value; }  
    }  
  
    public string phone  
    {
```

```

        get { return phone; }
        set { phone = value; }
    }
}

```

3.2 Data Access Object

Data Access Object (DAO) adalah sebuah objek yang bertanggung jawab untuk melakukan segala hal yang berkaitan dengan database, mulai dari buka koneksi, query sampai manipulasi data. Umumnya DAO ini berisi operasi CRUD (Create, Read, Update, Delete) dimana :

Create = Membuat record baru (SQL INSERT)

Read = Membaca data (SQL SELECT)

Update = Update data (SQL Update)

Delete = Menghapus data (SQL Delete)

Untuk menandakan sebuah class adalah layer DAO gunakan akhiran "Dao" disetiap class yang dibuat (ini sebetulnya bukan keharusan, hanya kesepakatan saja). Contoh jika class business entity nya Customer maka DAO bernama CustomerDao

Contoh class CustomerDao

```

using System;
using System.Data;
using System.Data.SqlClient;

namespace DAO
{
    public class CustomerDao
    {
        public void Create(string customerId, string companyName)
        {
            string connStr = "Data Source=SYSTEMINTERFACE\\SQLEXPRESS;"
                + "Initial Catalog=NWIND;Integrated Security=true";

            SqlConnection conn = new SqlConnection(connStr);
            conn.Open();

            string sql = "INSERT INTO Customers (CustomerID,CompanyName) "
                + "VALUES ('" + customerId + "','" + companyName + "')";

            SqlCommand cmd = new SqlCommand(sql, conn);
            cmd.ExecuteNonQuery();
        }

        public void Read()
        {
            string connStr = "Data Source=SYSTEMINTERFACE\\SQLEXPRESS;"
                + "Initial Catalog=NWIND;Integrated Security=true";

            SqlConnection conn = new SqlConnection(connStr);
            conn.Open();
        }
    }
}

```

```

        string sql = "SELECT * FROM Customers";

        SqlCommand cmd = new SqlCommand(sql, conn);
        SqlDataReader rdr = cmd.ExecuteReader();

        while (rdr.Read())
        {
            Console.WriteLine(rdr["CustomerId"]);
            Console.WriteLine(rdr["CompanyName"]);
        }
    }

    public void Update(string customerId, string companyName)
    {
        string connStr = "Data Source=SYSTEMINTERFACE\\SQLEXPRESS;"
            + "Initial Catalog=NWIND;Integrated Security=true";

        SqlConnection conn = new SqlConnection(connStr);
        conn.Open();

        string sql = "UPDATE Customers SET CompanyName='" + companyName
            + " WHERE CustomerId='" + customerId + "'";

        SqlCommand cmd = new SqlCommand(sql, conn);
        cmd.ExecuteNonQuery();
    }

    public void Delete(string customerId)
    {
        string connStr = "Data Source=SYSTEMINTERFACE\\SQLEXPRESS;"
            + "Initial Catalog=NWIND;Integrated Security=true";

        SqlConnection conn = new SqlConnection(connStr);
        conn.Open();

        string sql = "DELETE FROM Customers WHERE CustomerId='"
            + customerId + "'";
        SqlCommand cmd = new SqlCommand(sql, conn);
        cmd.ExecuteNonQuery();
    }
}

```

Kalau kita perhatikan ada beberapa bagian kode yang dipanggil berulang-ulang. Setiap method (Create, Read, Update, Delete) ada bagian untuk buka koneksi, tentu saja hal ini tidak efisien karena salah satu tujuan OOP adalah reusability, penggunaan ulang kembali.

Mengikuti nasihat **Martin Fowler**, ketika menemukan ada blok kode yg berulang 3 kali maka harus di refactoring. Refactoring adalah pengubahan struktur kode tanpa menambah fungsionalitas tertentu. Pengubahan hanya bertujuan untuk efisiensi dan efektifitas penulisan kode dengan cara menghilangkan atau menata kembali duplikasi kode yang mungkin terjadi.

Ingat bagi programmer berpengalaman kode yang ditulis tidak hanya harus menyelesaikan masalah (*solve the problem*) namun penyelesaian tersebut haruslah indah (*elegant*)

Bagi saya blok kode tersebut seperti bait-bait puisi, yang ketika membaca nya saja sudah dapat menggugah perasaan yang paling dalam karena nilai estetikanya. Struktur kode yang bagus juga menyebabkan kode menjadi mudah dibaca . Tentu saja hal ini sangat bermanfaat untuk maintenance dikemudian hari, terlebih lagi jika kita bekerja dalam tim.

Refactoring 1

```
using System;
using System.Data;
using System.Data.SqlClient;

namespace DAO
{
    public class CustomerDao
    {
        private SqlConnection conn;

        public CustomerDao()
        {
            string connStr = "Data Source=SYSTEMINTERFACE\\SQLEXPRESS;"
                + "Initial Catalog=NWIND;Integrated Security=true";

            conn = new SqlConnection(connStr);
            conn.Open();
        }

        public void Create(string customerId, string companyName)
        {
            string sql = "INSERT INTO Customers (CustomerID,CompanyName) "
                + "VALUES ('" + customerId + "', '" + companyName + "')";

            SqlCommand cmd = new SqlCommand(sql, conn);
            cmd.ExecuteNonQuery();
        }

        public void Read()
        {
            string sql = "SELECT * FROM Customers";

            SqlCommand cmd = new SqlCommand(sql, conn);
            SqlDataReader rdr = cmd.ExecuteReader();

            while (rdr.Read())
            {
                Console.WriteLine(rdr["CustomerID"]);
                Console.WriteLine(rdr["CompanyName"]);
            }
        }

        public void Update(string customerId, string companyName)
        {
            string sql = "UPDATE Customers SET CompanyName='" + companyName
                + " WHERE CustomerId='" + customerId + "'";
        }
    }
}
```

```

        SqlCommand cmd = new SqlCommand(sql, conn);
        cmd.ExecuteNonQuery();

    }

    public void Delete(string customerId)
    {
        string sql = "DELETE FROM Customers WHERE CustomerId='"
            + customerId + "'";
        SqlCommand cmd = new SqlCommand(sql, conn);
        cmd.ExecuteNonQuery();
    }
}

```

Kali ini koneksi cukup kita tuliskan sekali pada constructor class CustomerDao deklarasi variabel conn dikeluarkan dari method menjadi field dari class CustomerDao. Dengan demikian variabel conn dapat diakses diseluruh method yang dideklarasikan pada class customerDao. Duplikasi kode sudah dihilangkan, Kali ini kode kita jauh lebih ringkas bukan :-)

Karena kode untuk buka koneksi diletakkan di constructor, maka koneksi ke database akan dibuka otomatis ketika class CustomerDao di instantiasi. Berikutnya coba perhatikan method Read(), isi dari object rdr langsung ditampilkan ke console dengan menggunakan perintah Console.WriteLine(). Hmm..lalu dimana permasalahannya?apakah perlu di refactoring juga?

Jawabnya ya. Desain class yang baik harus meminimalisir ketergantungan antar class. Usahakan class-class yang saling berhubungan bersifat *lousley-couple*. Perintah Console.WriteLine() pada class CustomerDao jelas-jelas membuat class tersebut tergantung dengan user interface (dalam hal ini user interface nya berbasis Console) Bagaimana suatu saat jika user interface nya diganti dengan GUI atau web, tentu harus kita lakukan perubahan lagi.

Dalam Object Oriented Design dikenal prinsip "*Separation of Concern*" atau ada juga konsep "*Single Responsibility Principle*". Salah satu penerapan kedua konsep ini adalah teknik layering yang sedang kita bahas. masing-masing layer memiliki tanggung jawab yang unik (*single responsibility*) yang memiliki concern tertentu yang spesifik.

Dalam hal ini Data Access Object hanya memiliki concern dan tanggung jawab terhadap pengaksesan data, tidak ada hubungannya dengan menampilkan ke layar. Tanggung jawab tersebut (menampilkan ke layar) diserahkan pada objek di layer Presentation/User Interface

Refactoring 2 :

```

public SqlDataReader Read()
{
    string sql = "SELECT * FROM Customers";

    SqlCommand cmd = new SqlCommand(sql, conn);
    SqlDataReader rdr = cmd.ExecuteReader();

    return rdr;
}

```


Cara Mengakses CustomerDao

Layer Presentation :

```
CustomerDao custDao = new CustomerDao();
SqlDataReader rdr = custDao.Read();
while (rdr.Read())
{
    Console.WriteLine(rdr["CustomerId"]);
    Console.WriteLine(rdr["CompanyName"]);
}
```

That's it. apakah ada yang perlu diperbaiki lagi? sepertinya sudah perfect? Hmm..no code perfect, jangan berpuas diri dulu. Coba sekarang buat lagi sebuah Data Access Object, namakan saja SupplierDao

```
using System;
using System.Data;
using System.Data.SqlClient;

namespace DAO
{
    public class SupplierDao
    {
        private SqlConnection conn;

        public SupplierDao()
        {
            string connStr = "Data Source=SYSTEMINTERFACE\\SQLEXPRESS;"
                + "Initial Catalog=NWIND;Integrated Security=true";

            conn = new SqlConnection(connStr);
            conn.Open();
        }

        public void Create()
        {
            // ....
        }

        public SqlDataReader Read()
        {
            // .....
        }

        public void Update()
        {
            // .....
        }

        public void Delete()
        {
            // .....
        }
    }
}
```

Disetiap constructor Data Access Object ada bagian yang sama, yaitu kode untuk buka koneksi yang berisi connection string dan object `SqlConnection`. Ini juga duplikasi kode tapi letaknya sudah antar class. Bayangkan kalau kita punya banyak DAO seperti ini, waktu terbuang percuma untuk menulis hal yang sama, belum lagi kalo connection string nya di edit. Misalnya letak atau nama database nya diganti. hmmm..sepertinya ini buruk..

Kalau kita menuruti nasihat Martin Fowler, maka blok kode seperti ini akan di refactor menjadi class, "*Extract method to class*". Buat sendiri class untuk menangani buka koneksi yang kemudian dipanggil disetiap DAO

```
public class ConnectionService
{
    public static SqlConnection GetConnection()
    {
        string connStr = "Data Source=SYSTEMINTERFACE\\SQLEXPRESS;"
            + "Initial Catalog=NWIND;Integrated Security=true";

        SqlConnection conn = new SqlConnection(connStr);
        conn.Open();

        return conn;
    }
}
```

CustomerDao

```
public class CustomerDao
{
    private SqlConnection conn;

    public CustomerDao()
    {
        this.conn=ConnectionService.GetConnection();
    }
}
```

SupplierDao

```
public class SupplierDao
{
    private SqlConnection conn;

    public SupplierDao()
    {
        this.conn=ConnectionService.GetConnection();
    }
}
```

Sentuhan terakhir tambahkan blok `try..catch` disetiap method untuk exception handling (penanganan kesalahan)

Source Code Lengkap :

CustomerDao

```
using System;
using System.Data;
using System.Data.SqlClient;

namespace DAO
{
    public class CustomerDao
    {
        private SqlConnection conn;

        public CustomerDao()
        {
            this.conn = ConnectionService.GetConnection();
        }

        public void Create(string customerId, string companyName)
        {
            try
            {
                string sql = "INSERT INTO Customers (CustomerID,CompanyName) "
                    + "VALUES ('" + customerId + "','" + companyName + "')";

                SqlCommand cmd = new SqlCommand(sql, conn);
                cmd.ExecuteNonQuery();
            }
            catch (SqlException sqllex)
            {
                throw new Exception(sqllex.Message.ToString());
            }
        }

        public SqlDataReader Read()
        {
            SqlDataReader rdr = null;
            try
            {
                string sql = "SELECT * FROM Customers";
                SqlCommand cmd = new SqlCommand(sql, conn);
                rdr = cmd.ExecuteReader();
            }
            catch (SqlException sqllex)
            {
                throw new Exception(sqllex.Message.ToString());
            }

            return rdr;
        }
    }
}
```

```

        public void Update(string customerId, string companyName)
        {
            string sql = "UPDATE Customers SET CompanyName='" + companyName
                + " WHERE CustomerId='" + customerId + "'";

            SqlCommand cmd = new SqlCommand(sql, conn);
            cmd.ExecuteNonQuery();

        }

        public void Delete(string customerId)
        {
            try
            {
                string sql = "DELETE FROM Customers WHERE CustomerId='"
                    + customerId + "'";
                SqlCommand cmd = new SqlCommand(sql, conn);
                cmd.ExecuteNonQuery();
            }
            catch (SqlException sqllex)
            {
                throw new Exception(sqllex.Message.ToString());
            }
        }
    }
}

```

ConnectionService

```

using System;
using System.Data;
using System.Data.SqlClient;

namespace DAO
{
    public class ConnectionService
    {
        public static SqlConnection GetConnection()
        {
            SqlConnection conn = null;
            try
            {
                string connStr = "Data Source=SYSTEMINTERFACE\\SQLEXPRESS;"
                    + "Initial Catalog=NWIND;Integrated Security=true";

                conn = new SqlConnection(connStr);
                conn.Open();
            }
            catch (SqlException sqllex)
            {
                throw new Exception(sqllex.Message.ToString());
            }
            return conn;
        }
    }
}

```

Main

```
using System;
using System.Data;
using System.Data.SqlClient;

namespace DAO
{
    class Program
    {
        static void Main(string[] args)
        {
            CustomerDao custDao = new CustomerDao();
            SqlDataReader rdr = custDao.Read();
            while (rdr.Read())
            {
                Console.WriteLine(rdr["CustomerId"]);
                Console.WriteLine(rdr["CompanyName"]);
            }

            Console.ReadLine();
        }
    }
}
```

3.3 Object Mapping

Rekayasa Sistem Berorientasi Objek menuntut untuk sejenak merubah cara pandang kita terhadap sebuah sistem. Dalam dunia objek (object universe) segala sesuatu adalah objek. Demi konsistensi, Tools yang digunakan dalam mengembangkan system dalam paradigma object oriented pun sebaiknya ada embel-embel "object" Pemrograman menggunakan OOP (*Object Oriented Programming*), analisis menggunakan OOAD (*Object Oriented Analysis and Design*), dan database (seharusnya) menggunakan OODB (*Object Oriented Database*).

Mengapa kita harus "dipaksa" menggunakan teknologi berorientasi objek? bukankah pendekatan sebelumnya (terstruktur) sudah terbukti selama bertahun-tahun? Hmm ... pertama tanpa mengurangi rasa hormat saya terhadap ahli-ahli komputer yang berkontribusi terhadap paradigma terstruktur, menurut saya paradigma tersebut sudah ketinggalan zaman, tidak mencukupi lagi jika dihadapkan pada problematika software development terkini yang makin kompleks.

Kedua saya merasa "tidak dipaksa" menggunakan paradigma berorientasi objek. Saya dengan senang hati menggunakannya karena bukan hanya menggumbar janji lewat jargon-jargon nya, tetapi sudah terbukti (proven) dan dapat dipertanggung jawabkan hasilnya. OOP sudah umum digunakan sekarang, semua bahasa pemrograman modern mendeklarasikan diri sebagai object oriented. Begitu juga OOAD, UML (*Unified Modelling Language*) menjadi standar de facto industri yang didukung oleh perusahaan-perusahaan sekaliber IBM, Microsoft, Oracle, SUN

Microsystem, dan banyak perusahaan lainnya yang tergabung dalam OMG (*Object Management Group*).

Untuk OODB ada pengecualian, walaupun riset dan produk nya sudah bermunculan, DBMS populer yang beredar dipasaran kebanyakan adalah Relational DBMS. Oracle, SQL Server, DB2, MySQL, PostgreSQL semuanya termasuk RDBMS. Konsep RDBMS tidak begitu mirip dengan dunia objek karena berbasis aljabar relasi. Data disimpan dalam sebuah tabel 2 dimensi yang saling berelasi. Hubungan antara tabel dibatasi oleh aturan normalisasi yang sangat ketat, berbeda dengan dunia objek yang memiliki hubungan lebih variatif mulai dari asosiasi, generalisasi, independensi, komposisi, dan agregasi. Intinya antara dunia objek dan RDBMS memiliki jurang pemisah. Para ahli menyebut fenomena ini "*Impedence Mismatch*". Lebih lanjut mengenai jargon tersebut bisa dilihat Wikipedia. Jadi apakah antara OOP dan RDBMS tidak berjodoh?

Jawabnya bisa iya bisa tidak. Keterbatasan RDBMS ini bisa diatasi dengan pendekatan Object (Relational) Mapping. Teknik ini akan mengkonversi row/baris dari tabel di RDBMS menjadi objek. Konversi bisa dilakukan secara manual atau menggunakan tool ORM (*Object Relational Mapping*) Framework.

Perhatikan class CustomerDao

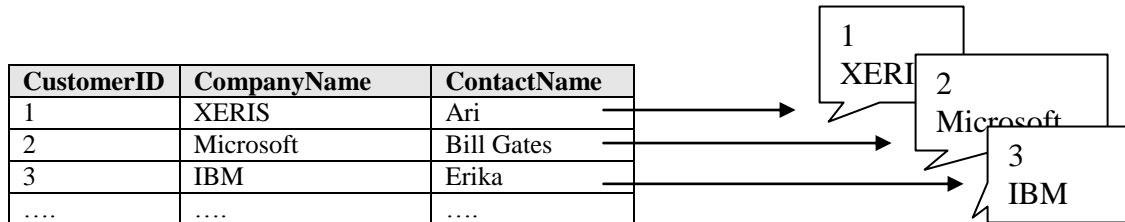
```
using System;
using System.Data;
using System.Data.SqlClient;

namespace DAO
{
    public class CustomerDao
    {
        private SqlConnection conn;

        public CustomerDao()
        {
            this.conn = ConnectionService.GetConnection();
        }

        public SqlDataReader Read()
        {
            SqlDataReader rdr = null;
            try
            {
                string sql = "SELECT * FROM Customers";
                SqlCommand cmd = new SqlCommand(sql, conn);
                rdr = cmd.ExecuteReader();
            }
            catch (SqlException sqllex)
            {
                throw new Exception(sqllex.Message.ToString());
            }
            return rdr;
        }
    }
}
```

Method `Read()` memiliki return value `rdr` yang bertipe `SqlDataReader`. Bisa dikatakan method tersebut mengembalikan record/row dari tabel bukannya objek `Customer`, padahal kita sudah buat `Business Object` nya. Agar code kita lebih bernuansa OOP, row ini akan dipetakan menjadi objek, satu row akan diwakili oleh satu objek. Perhatikan ilustrasi berikut :



Method `Read()` dipisah jadi 2 method yaitu `GetById()` dan `GetAll()`

```
public Customer GetById(string customerId)
{
    Customer customer = null;
    try
    {
        string sql = "SELECT * FROM Customers WHERE CustomerId='"
            + customerId + "'";

        SqlCommand cmd = new SqlCommand(sql, conn);
        SqlDataReader rdr = cmd.ExecuteReader();

        if (rdr.Read())
        {
            Customer customer = new Customer();
            customer.CustomerId = rdr["CustomerId"].ToString();
            customer.CompanyName = rdr["CompanyName"].ToString();
            customer.ContactName = rdr["ContactName"].ToString();
        }
    }
    catch (SqlException sqllex)
    {
        throw new Exception(sqllex.Message.ToString());
    }
    return customer;
}
```

Method `GetById()` digunakan untuk mencari data `Customer` berdasarkan `CustomerId` tertentu. Jika row ada akan dimapping ke objek supplier. Isi Field ditabel `Customers` akan dimapping ke property class `Customer`. Dengan demikian method `GetById()` return valuenya objek `customer`. Jika ingin mendapatkan semua record dari tabel `Customers`, gunakan method `GetAll()`

```

public List<Customer> GetById(string customerId)
{
    List<Customer> customer = new List<Customer>();
    try
    {
        string sql = "SELECT * FROM Customers";

        SqlCommand cmd = new SqlCommand(sql, conn);
        SqlDataReader rdr = cmd.ExecuteReader();

        while (rdr.Read())
        {
            Customer customer = new Customer();
            customer.CustomerId = rdr["CustomerId"].ToString();
            customer.CompanyName = rdr["CompanyName"].ToString();
            customer.ContactName = rdr["ContactName"].ToString();

            list.Add(customer);
        }
    }
    catch (SqlException sqlex)
    {
        throw new Exception(sqlex.Message.ToString());
    }

    return list;
}

```

Query "SELECT * FROM Customers" kan mengambil seluruh record dalam tabel Customers. Karena terdiri dari banyak record sehingga dari proses mapping akan menghasilkan banyak objek sesuai dengan jumlah recordnya. Jika kita berhubungan dengan banyak objek, maka kumpulan objek ini harus diwadahi dalam container/tempat penyimpanannya objek. Koleksi dari objek tersebut dapat disimpan di tipe data Array atau class khusus untuk menangani collection seperti List. Class List dapat digunakan jika kita memanggil namespace System.Collection. sedangkan jika ingin menggunakan tipe Generic (List<T>) bisa dipanggil namespace System.Collection.Generic.

Generic adalah fitur baru yang diperkenalkan di .NET Framework 2.0. Dengan menggunakan generic data type yang digunakan lebih aman (type safe), karena kesalahan bisa diketahui pada saat design time bukan pada saat runtime. Generic juga menghilangkan proses Casting yang biasanya dilakukan jika kita berhubungan dengan tipe yang tidak sesuai. Pada method GetAll() return value nya adalah sebuah List generic yang isi nya koleksi object class Customer yang dihasilkan dari proses mapping dari record tabel Customers

Berikut ini modifikasi method GetAll() untuk pencarian Customer berdasarkan CustomerName

```

public List<Customer> FindByName(string companyName)
{
    List<Customer> customer = new List<Customer>();
    try
    {
        string sql = "SELECT * FROM Customers WHERE CompanyName LIKE '%"
            + companyName + "'";
    }
}

```



```

        SqlCommand cmd = new SqlCommand(sql, conn);
        SqlDataReader rdr = cmd.ExecuteReader();

        while (rdr.Read())
        {
            Customer customer = new Customer();
            customer.CustomerId = rdr["CustomerId"].ToString();
            customer.CompanyName = rdr["CompanyName"].ToString();
            customer.ContactName = rdr["ContactName"].ToString();

            list.Add(customer);
        }
    }
    catch (SqlException sqllex)
    {
        throw new Exception(sqllex.Message.ToString());
    }

    return list;
}

```

Oke cukup dengan bagian Read nya, bagaimana dengan Create, Update, dan Delete. Apakah bisa kita perlakukan dengan pendekatan object mapping juga? Hmm.. mungkin lebih tepatnya object dependency. Perhatikan contoh source code berikut :

```

public void Create(string customerId, string companyName)
{
    try
    {
        string sql = "INSERT INTO Customers (CustomerId,CompanyName) "
            + "VALUES ('" + customerId + "','" + companyName + "')";

        SqlCommand cmd = new SqlCommand(sql, conn);
        cmd.ExecuteNonQuery();
    }
    catch (SqlException sqllex)
    {
        throw new Exception(sqllex.Message.ToString());
    }
}

```

Method Create yang digunakan untuk menambah row baru ke tabel Customer memiliki 2 parameter tipe primitive yaitu string. Karena kita sudah memiliki class Customer sebagai business object maka sebaiknya gunakan parameter objek. Sehingga method Create memiliki dependency ke class Customer

```

public void Create(Customer customer)
{
    try
    {
        string sql = "INSERT INTO Customers (CustomerId,CompanyName) "
            + "VALUES ('" + customer.CustomerId + "','"
            + customer.CompanyName + "')";
    }
}

```

```

        SqlCommand cmd = new SqlCommand(sql, conn);
        cmd.ExecuteNonQuery();
    }
    catch (SqlException sqllex)
    {
        throw new Exception(sqllex.Message.ToString());
    }
}

```

Update dan Delete

```

public void Update(Customer customer)
{
    try
    {
        string sql = "UPDATE Customers SET CompanyName='" + customer.CompanyName
            + " WHERE CustomerId='" + customer.CustomerId + "'";

        SqlCommand cmd = new SqlCommand(sql, conn);
        cmd.ExecuteNonQuery();
    }
    catch (SqlException sqllex)
    {
        throw new Exception(sqllex.Message.ToString());
    }
}

```

```

public void Delete(Customer customer)
{
    try
    {
        string sql = "DELETE FROM Customers WHERE CustomerId='"
            + customer.CustomerId + "'";

        SqlCommand cmd = new SqlCommand(sql, conn);
        cmd.ExecuteNonQuery();
    }
    catch (SqlException sqllex)
    {
        throw new Exception(sqllex.Message.ToString());
    }
}

```

Cara Akses

```

Customer customer = new Customer();

customer.CustomerId = "XERS";
customer.CompanyName = "XERIS System Interface";
customer.ContactName = "Ari";

CustomerDao custDao = new CustomerDao();

```

```
custDao.Save(customer)

List<Customer> list = custDao.GetAll();
foreach (Customer cust in list)
{
    Console.WriteLine(cust.CustomerId + "-" + cust.CompanyName);
}
```

3.4 Relasi Objek

Relational DBMS memiliki tabel-tabel yang saling beralasi. Jika tabel-tabel tersebut tidak berelasi maka tidak usah menggunakan RDBMS, kembali lagi saja ke jaman teks file atau mungkin sedikit lebih canggih menggunakan XML.RDBMS dibuat untuk mengatasi kelemahan tipe database sebelumnya. Pertanyaanya bagaimanakah caranya memetakan tabel yang beralasi? Apakah class bisa kita relasikan satu sama lain seperti halnya tabel? jawabnya tentu saja bisa

RDBMS yang berbasis aljabar relasi memiliki tipe relasi yang disebut Asosiasi. Hubungan ini terbagi tiga yaitu hubungan one-to-one, one-to-many, many-to-one dan many-to-many. Sebelum merelasikan tabel harus dilakukan proses normalisasi terlebih dahulu untuk menjamin konsistensi, integritas data dan meminimalisir redudansi. Tidak seperti RDBMS yang hanya memiliki 1 jenis relasi, class memiliki 5 macam jenis relasi yang mungkin yaitu : Asosiasi, Agregasi, Generalisasi, Depedensi, dan Realisasi .

3.4.1 Asosiasi

Hubungan antara class yang ada. Asosiasi memungkinkan sebuah class mengetahui Property dan Method class lain yang saling berelasi. Syaratnya Property dan Method harus memiliki access modifier/visibility public.

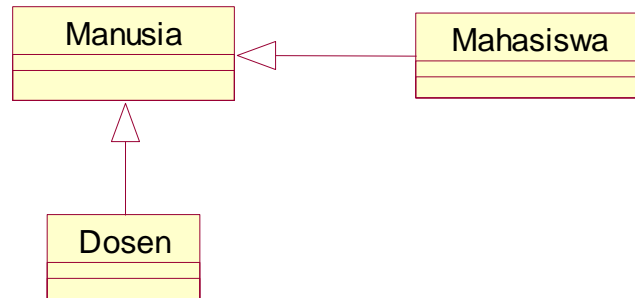
3.4.2 Agregasi

Relasi antara "keseluruhan" dengan "bagian"

3.4.3 Generalisasi

Relasi pewarisan antara dua class. Class induk disebut super class sedangkan class turunan disebut sub class. Superclass mewarisi seluruh property dan method ke subclass, namun tidak berlaku sebaliknya. Subclass bisa jadi memiliki property dan method spesifik yang tidak dimiliki oleh superclassnya. Generalisasi dalam OOP disebut juga Inheritance. Dengan menggunakan Inheritance class yang kita buat akan menjadi lebih simple karena tidak harus menuliskan semua property dan method pada class anak, cukup "diturunkan" dari class induk

Perhatikan contoh berikut



Class Manusia

```
public class Manusia
{
    private string nama;
    private int berat;
    private int tinggi;

    public string Nama
    {
        get { return nama; }
        set { nama = value; }
    }

    public int Berat
    {
        get { return berat; }
        set { berat = value; }
    }

    public int Tinggi
    {
        get { return tinggi; }
        set { tinggi = value; }
    }
}
```

Class Mahasiswa

```
public class Mahasiswa : Manusia
{
    private string nim;

    public string Nim
    {
        get { return Nim; }
        set { Nim = value; }
    }
}
```

Class Dosen

```
public class Dosen : Manusia
{
    private string nip;

    public string Nip
    {
        get { return nip; }
        set { nip = value; }
    }
}
```

Instantiasi

```
Manusia m=new Manusia();
m>Nama="Sandra";
m.Berat=55;
m.Tinggi=175;

Student s=new Student();
s.Nim="JXXX";
s>Nama="Dewi";

Dosen d1=new Dosen();
d1.Nip="XYZ";
d1>Nama="Ari";

Dosen d2=new Dosen();
d2.Nip="ABC";
d2>Nama="Erika";
```

Perhatikan class Student dan class Dosen. Mengapa kedua class ini dapat mengakses property Nama? padahal kalau kita perhatikan di kedua class tersebut tidak ada property Nama. Hanya ada property Nim di class Mahasiswa dan property Nip di class Dosen. Jawabanya karena kedua class ini "Inherit" ke class Person. Semua property dan method di class Person otomatis turun ke Mahasiswa dan Dosen. Jadi lebih simple bukan.

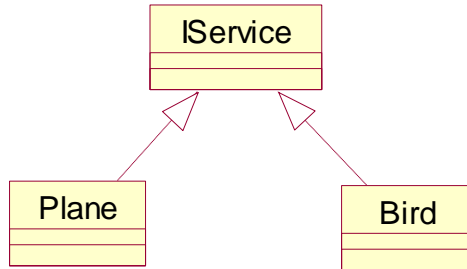
3.4.4 Depedency

Relasi yang menunjukkan sebuah class mengacu ke class lainnya. Depedency adalah sebuah relasi yang bersifat tightly-coupled. Perubahan pada class yang diacu bisa jadi menyebabkan perubahan di class pengguna

3.4.5 Realisasi

Relasi antara interface dengan class yang menjadi implemetasi dari interface tersebut. "Programming to interface not to implemantation", begitu kata para developer berpengalaman.

Programming to interface merupakan best practice yang patut diikuti dalam menulis program. Interface memisahkan apa(what) dengan bagaimana(how) nantinya hal tersebut implementasikan. Dengan menggunakan interface, struktur kode kita menjadi *loosely-coupled*, karena memungkinkan secara dinamis mengganti implementasi.



Perhatikan contoh berikut :

```
public interface IService
{
    void Fly();
}
```

```
public class Plane : IService
{
    public void Fly()
    {
        Console.WriteLine("Plane ready take off..");
    }
}
```

```
public class Bird : IService
{
    public void Fly()
    {
        Console.WriteLine("Bird fly on the sky");
    }
}
```

```
IService plane=new Plane();
plane.Fly();

IService bird=new Bird();
bird.Fly();
```

Kedua class diatas mengimplementasikan interface IService yang memiliki deklarasi method Fly(). Sebuah interface hanya berisi deklarasi method atau property yang nantinya baru diimplementasikan di class yang menggunakan interface tersebut. Interface bisa dipandang

sebagai kontrak yang harus dipenuhi oleh class yang mengimplementasikan, jika method di interface tersebut tidak ditulis akan muncul pesan error. Pemahaman dasar mengenai interface sangat diperlukan, karena nantinya banyak dipakai dalam Design Pattern GoF (Gang of Four).

Oke, cukup dengan teorinya. Saya akan langsung saja tunjukkan bagaimana menggunakan semua yang diteorikan tadi untuk merelasikan objek-objek yang mewakili tabel-tabel yang saling berelasi. Berikut saya berikan contoh penerjemahan tabel Products ke class Product

Class Product

```
public class Product
{
    private int productId;
    private string name;
    private int categoryId;
    private int supplierId;
    private int stock;

    public int ProductId
    {
        get { return productId; }
        set { productId = value; }
    }

    public string Name
    {
        get { return name; }
        set { name = value; }
    }

    public int CategoryId
    {
        get { return categoryId; }
        set { categoryId = value; }
    }

    public int SupplierId
    {
        get { return supplierId; }
        set { supplierId = value; }
    }

    public int Stock
    {
        get { return stock; }
        set { stock = value; }
    }
}
```

Perhatikan class Product, penerjemah dari tabel Products benar-benar sama persis. Column di tabel diterjemahkan ke field/variabel di class Product, termasuk foreign key CATEGORY_ID dan SUPPLIER_ID. Demi desain yang lebih baik, saya akan melakukan refactoring class Customer.

field categoryId dan supplierId yang bertipe int akan diganti menjadi tipe objeknya yaitu class Supplier dan Category.

Class Product

```
using System;

namespace ObjectMapping
{
    public class Product
    {
        private int productID;
        private string name;
        private Category category = new Category();
        private Supplier supplier = new Supplier();
        private int stock;

        public int ProductID
        {
            get { return productID; }
            set { productID = value; }
        }

        public string Name
        {
            get { return name; }
            set { name = value; }
        }

        public Category Category
        {
            get { return category; }
            set { category = value; }
        }

        public Supplier Supplier
        {
            get { return supplier; }
            set { supplier = value; }
        }

        public int Stock
        {
            get { return stock; }
            set { stock = value; }
        }
    }
}
```

Class Category

```
using System;

namespace ObjectMapping
{
    public class Category
```



```

{
    private int categoryID;
    private string categoryName;

    public int CategoryID
    {
        get { return categoryID; }
        set { categoryID = value; }
    }

    public string CategoryName
    {
        get { return categoryName; }
        set { categoryName = value; }
    }
}

```

Class Supplier

```

using System;

namespace ObjectMapping
{
    public class Supplier
    {
        private int supplierID;
        private string companyName;

        public int SupplierID
        {
            get { return supplierID; }
            set { supplierID = value; }
        }

        public string CompanyName
        {
            get { return companyName; }
            set { companyName = value; }
        }
    }
}

```

Dengan mengubah tipenya menjadi objek kita bisa mengakses semua property objek category dan supplier jika diperlukan.

Contoh :

```
Product p=new Product();
p.Name="Remote";
p.Category.Name="Electronic";
p.Supplier.CompanyName="Samsung";

Console.WriteLine(p.Name + "-" + p.Category.Name + "-"
    + p.Supplier.CompanyName);
```

Relasi antara tabel di Products dengan Categories dan Suppliers diterjemahkan menjadi asosiasi antara class Product dengan class Category dan Supplier. lalu bagaimana jika kita punya relasi One-to-many? Bagaimana cara menterjemahkannya ke class? Gunakanlah **object collection**!

Data Access

ProductDao

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;

namespace ObjectMapping
{
    public class ProductDao : IProductDao
    {
        private SqlConnection conn;
        private SqlCommand cmd;
        private SqlDataReader rdr;
        private string sql;

        public ProductDao()
        {
            string connStr = "Data Source=SYSTEMINTERFACE\\SQLEXPRESS;"
                + "Initial Catalog=NWIND;Integrated Security=True";

            conn = new SqlConnection(connStr);
            conn.Open();
        }

        public Product Get(int id)
        {
            string sql = "SELECT dbo.Products.ProductID,"
                + "dbo.Products.ProductName,dbo.Suppliers.CompanyName,"
                + "dbo.Categories.CategoryName,dbo.Products.UnitsInStock "
                + "FROM dbo.Products INNER JOIN "
                + "dbo.Suppliers ON dbo.Products.SupplierID = "
                + "dbo.Suppliers.SupplierID "
                + "INNER JOIN dbo.Categories ON dbo.Products.CategoryID = "
                + "dbo.Categories.CategoryID WHERE ProductID=" + id;
```

```

        cmd = new SqlCommand(sql, conn);
        rdr=cmd.ExecuteReader();

        Product p = new Product();

        if (rdr.Read())
        {
            p.ProductID = Convert.ToInt32(rdr["ProductID"]);
            p.Name = rdr["ProductName"].ToString();
            p.Category.CategoryName = rdr["CategoryName"].ToString();
            p.Supplier.CompanyName = rdr["CompanyName"].ToString();

        }
        rdr.Close();

        return p;
    }

    public List<Product> Find()
    {
        List<Product> list = new List<Product>();

        string sql = "SELECT dbo.Products.ProductID, "
            + "dbo.Products.ProductName,dbo.Suppliers.CompanyName, "
            + "dbo.Categories.CategoryName,dbo.Products.UnitsInStock "
            + "FROM dbo.Products INNER JOIN "
            + "dbo.Suppliers ON dbo.Products.SupplierID = "
            + "dbo.Suppliers.SupplierID "
            + "INNER JOIN dbo.Categories ON dbo.Products.CategoryID = "
            + "dbo.Categories.CategoryID";

        cmd = new SqlCommand(sql, conn);
        rdr = cmd.ExecuteReader();

        while (rdr.Read())
        {
            Product p = new Product();

            p.ProductID = Convert.ToInt32(rdr["ProductID"]);
            p.Name = rdr["ProductName"].ToString();
            p.Category.CategoryName = rdr["CategoryName"].ToString();
            p.Supplier.CompanyName = rdr["CompanyName"].ToString();

            list.Add(p);
        }

        return list;
    }

    public void Save(Product p)
    {
        string sql = "INSERT INTO Products "
            + "(ProductName,CategoryID,SupplierID,UnitsInStock) "
            + " VALUES ('" + p.Name + "','" + p.Category.CategoryID + "','"
            + p.Supplier.SupplierID + "','" + p.Stock + ")";

        SqlCommand cmd = new SqlCommand(sql, conn);
        cmd.ExecuteNonQuery();
    }
}

```

Agar desain class lebih bagus lagi kita praktekan "*Programming to interface, not to implementation*" dengan membuat interface IProductDao

```
using System;
using System.Collections.Generic;

namespace ObjectMapping
{
    interface IProductDao
    {
        Product Get(int id);
        List<Product> Find();
        void Save(Product p);
    }
}
```

Cara akses :

```
using System;
using System.Collections.Generic;

namespace ObjectMapping
{
    class Program
    {
        static void Main(string[] args)
        {
            IProductDao productDao = new ProductDao();

            //Save

            Product product = new Product();
            product.Name = "Cumi-Cumi";
            product.Category.CategoryID = 28;
            product.Supplier.SupplierID = 1;
            product.Stock = 20;

            productDao.Save(product);

            //View

            List<Product> list = productDao.Find();
            foreach (Product p in list)
            {
                Console.WriteLine("Product Name : " + p.Name);
                Console.WriteLine("Category      : " + p.Category.CategoryName);
                Console.WriteLine("Supplier       : " + p.Supplier.CompanyName);
            }
            Console.ReadLine();
        }
    }
}
```

Nah kita sudah mempraktekan Generalisasi, Dependency, Asosiasi, dan Aggregasi pada class ProductDao.

Relasi	Code
Generalisasi	<pre>public class ProductDao : IProductDao { }</pre>
Depedency	<pre>public void Save(Product p) { }</pre>
Asosiasi	<pre>public class Product { private Category category; private Supplier supplier; }</pre>
Aggregasi	<pre>public Product Get(int id) { }</pre>

BAB 4

Design Pattern

Ketika membangun sebuah aplikasi skala besar, terkadang kita sering tidak bisa menghindari mengulang-ulang penulisan source code. Jika terjadi duplikasi seperti ini code harus segera di refactoring. Istilah refactoring diperkenalkan oleh Martin Fowler yang berarti mengubah struktur code tanpa mengubah feature atau menambahkan fungsi-fungsi baru. Refactoring menyebabkan penulisan program menjadi lebih efisien karena tidak ada duplikasi atau code yang bertele-tele.

Disini saya tidak akan menjelaskan refactoring secara panjang lebar, justru saya kan menceritakan tentang Design Pattern. Hmm jargon apa lagi itu Design Pattern, tidak cukupkah dgn jargon-jargon OOP seperti encapsulation, inheritance, polymorphism, overloading, overriding, message passing, constructor, bla..bla..bla.... Terus terang sebagai orang yang berlatar belakang teknis saya senang sekali dengan istilah-istilah tersebut. Begitu 'sexy' terdengarnya. sama seperti ketika Mullan Jameela mengucapkan 'aw..aw...aw' di lagu Mahluk paling sexy nya :-). Tapi senang saja tentu tidak cukup, kita perlu mengetahui makna dan filosofi dibaliknya

4.1 DAO Pattern

Hmmm..daripada saya merinding membayangkan Mulan lebih baik kita kembali ke Design pattern. Menurut Wikipedia, Design Pattern adalah solusi berulang yang diterapkan untuk memecahkan masalah dalam software desain. Solusi berulang disini artinya solusi dari permasalahan tersebut sudah ditemukan oleh orang lain dan dirumuskan dalam sebuah pola/pattern. Sehingga jika kita punya masalah 'x' gunakan pola 'y'. Ini konsep reusability dalam paradigma berorientasi objek, "*dont reinvent the wheel*", daripada membuat dari awal gunakan konsep atau solusi yang sudah terbukti efektif (proven).

Design pattern berhubungan dengan konsep refactoring dalam mengubah desain program menjadi lebih elegan. Jika kita berbicara Design Pattern, tidak terlepas dari GoF (*Gang of Four*) Pattern-pattern ini sudah begitu melegenda sehingga pasti muncul di pattern-pattern lain yang lebih kompleks seperti Core J2EE pattern, Pattern Of Enterprise Architecture, Microsoft Pattern & Practice, atau Integration Pattern.

4.1.1 Apa itu DAO Pattern

Kali ini kita akan mencoba DAO Pattern. DAO Pattern adalah bagian dari Core J2EE Pattern Sun Microsystems yang diimplementasikan dalam bahasa Java (tentu saja) Saya menggunakan solusi dari dunia Java bukan berarti dunia .NET tidak tersedia katalog pattern semacam itu. Microsoft sendiri memiliki Pattern & Practice yang menghasilkan pattern2 composite sejenis J2EE Core pattern. Disini saya ingin menunjukkan bahwa belajar sesuatu itu bisa lintas bahasa pemrograman, yang perlu dipahami adalah konsep dasarnya Implementasi bisa diterapkan di platform/bahasa apapun, tentu saja untuk konteks ini yang sudah mendukung paradigma OOP.

Perhatikan contoh source code berikut :

Layer Business Logic

```
using System;

namespace DAOPattern
{
    public class Customer
    {
        private string customerID;
        private string companyName;
        private string contactName;
        private string phone;

        public string CustomerID
        {
            get { return customerID; }
            set { customerID = value; }
        }

        public string CompanyName
        {
            get { return companyName; }
            set { companyName = value; }
        }

        public string ContactName
        {
            get { return contactName; }
            set { contactName = value; }
        }

        public string Phone
        {
            get { return phone; }
            set { phone = value; }
        }
    }
}
```

Business logik bisa jadi hanya berisi entitas dari problem domain (domain permasalahan) implementasinya dalam bentuk class yang hanya bersisi property dari entitas tersebut

Data Access

```
using System;
using System.Collections.Generic;

namespace DAOPattern
{
    public interface ICustomerDAO
    {
        List<Customer> Find();
        List<Customer> FindByName(string name);
        Customer GetById(string id);
        void Save(Customer entity);
        void Update(Customer entity);
        void Delete(Customer entity);
    }
}
```

Programing to interface not to implementation. Pemisahan interface (what) dengan implementasi (how)

CustomerDao

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Common;

namespace DAOPattern
{
    public class CustomerDAO : ICustomerDAO
    {
        private DbConnection conn;
        private DbCommand cmd;
        private DbDataReader rdr;
        private string table = "Customers";

        public CustomerDAO(DataSource ds)
        {
            this.conn = ConnectionFactory.Build(ds);
        }

        public List<Customer> Find()
        {
            List<Customer> list = new List<Customer>();
            try
            {
                cmd = conn.CreateCommand();
                cmd.Connection = conn;
                cmd.CommandText = "SELECT * FROM " + table;
                rdr = cmd.ExecuteReader();
                while (rdr.Read())
                {

```



```

        Customer cust = new Customer();
        cust.CustomerID = Convert.ToString(rdr["CustomerID"]);
        cust.CompanyName = Convert.ToString(rdr["CompanyName"]);
        list.Add(cust);
    }
    rdr.Close();
}
catch (DbException ex)
{
    throw new Exception(ex.Message.ToString());
}
return list;
}

public List<Customer> FindByName(string name)
{
    List<Customer> list = new List<Customer>();
    try
    {
        cmd = conn.CreateCommand();
        cmd.Connection = conn;
        cmd.CommandText = "SELECT * FROM " + table
            + " WHERE CompanyName LIKE '%" + name + "%'";
        rdr = cmd.ExecuteReader();
        while (rdr.Read())
        {
            Customer cust = new Customer();
            cust.CustomerID = Convert.ToString(rdr["CustomerID"]);
            cust.CompanyName = Convert.ToString(rdr["CompanyName"]);

            list.Add(cust);
        }
        rdr.Close();
    }
    catch (DbException ex)
    {
        throw new Exception(ex.Message.ToString());
    }
    return list;
}

public Customer GetById(string id)
{
    Customer cust = null;
    try
    {
        cmd = conn.CreateCommand();
        cmd.Connection = conn;
        cmd.CommandText = "SELECT * FROM " + table
            + " WHERE CustomerID='" + id + "'";
        rdr = cmd.ExecuteReader();

        if (rdr.Read())
        {
            cust = new Customer();
            cust.CustomerID = Convert.ToString(rdr["CustomerID"]);
            cust.CompanyName = Convert.ToString(rdr["CompanyName"]);
        }
        rdr.Close();
    }
    catch (DbException ex)

```

```

        {
            throw new Exception(ex.Message.ToString());
        }
        return cust;
    }

    public void Save(Customer cust)
    {
        try
        {
            cmd = conn.CreateCommand();
            cmd.Connection = conn;
            cmd.CommandText = "INSERT INTO " + table
                + "(CustomerID,CompanyName) VALUES "
                + "(" + cust.CustomerID + ",'"
                + cust.CompanyName + "'" + ")";
            cmd.ExecuteNonQuery();

        }
        catch (DbException ex)
        {
            throw new Exception(ex.Message.ToString());
        }
    }

    public void Update(Customer cust)
    {
        try
        {
            cmd = conn.CreateCommand();
            cmd.Connection = conn;
            cmd.CommandText = "UPDATE " + table + " SET CompanyName='"
                + cust.CompanyName + "'"
                + "WHERE CustomerID='" + cust.CustomerID + "'";
            cmd.ExecuteNonQuery();

        }
        catch (DbException ex)
        {
            throw new Exception(ex.Message.ToString());
        }
    }

    public void Delete(Customer cust)
    {
        try
        {
            cmd = conn.CreateCommand();
            cmd.Connection = conn;
            cmd.CommandText = "DELETE FROM " + table
                + " WHERE CustomerID='" + cust.CustomerID + "'";
            cmd.ExecuteNonQuery();

        }
        catch (DbException ex)
        {
            throw new Exception(ex.Message.ToString());
        }
    }
}

```

DAOFactory

```
using System;

namespace DAOPattern
{
    public class DAOFactory
    {
        public static ICustomerDAO GetCustomerDAOInstance (DataSource ds)
        {
            return new CustomerDAO(ds);
        }
    }
}
```

ConnectionFactory

```
using System;
using System.Data;
using System.Data.Common;

namespace DAOPattern
{
    public class ConnectionFactory
    {
        private static DbConnection conn;

        public static DbConnection Build(DataSource ds)
        {
            try
            {
                DbProviderFactory factory =
                    DbProviderFactories.GetFactory(ds.Driver);
                conn = factory.CreateConnection();
                conn.ConnectionString = ds.ConnectionString;
                conn.Open();
            }
            catch (DbException ex)
            {
                throw new Exception(ex.Message.ToString());
            }
            return conn;
        }
    }
}
```

DataSource

```
using System;

namespace DAOPattern
{
    public class DataSource
    {
        private string driver;
        private string connectionString;

        public DataSource()
        {
        }

        public DataSource(string driver, string connectionString)
        {
            this.driver = driver;
            this.connectionString = connectionString;
        }

        public string Driver
        {
            get { return driver; }
            set { driver = value; }
        }

        public string ConnectionString
        {
            get { return connectionString; }
            set { connectionString = value; }
        }
    }
}
```

Presentation Layer

```
static void Main(string[] args)
{
    DataSource ds = new DataSource();

    ds.Driver = ConfigurationManager.AppSettings["Driver"];
    ds.ConnectionString = ConfigurationManager.AppSettings["ConnectionString"];

    ICustomerDAO custDao = DAOFactory.GetCustomerDAOInstance(ds);

    //Save

    Customer cust = new Customer();

    cust.CustomerID = "XERIS";
    cust.CompanyName = "Xeris System Interface";
    custDao.Save(cust);

    //View

    List<Customer> customers = custDao.Find();
}
```

```
foreach (Customer cust in customers)
{
    Console.WriteLine(cust.CompanyName);
}

Console.ReadLine();

}
```

XML Configuration

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="Driver" value="System.Data.SqlClient"/>
    <add key="ConnectionString" value="Data Source=SYSTEMINTERFACE\\SQLEXPRESS;"
      "Initial Catalog=NWIND;Integrated Security=True"/>
  </appSettings>
</configuration>
```

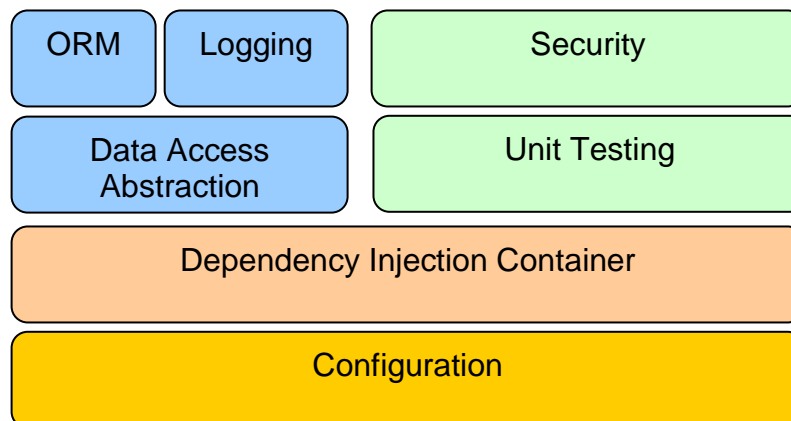
BAB 5

Origami Framework

Menurut Booch, framework adalah pola arsitektur yang menyediakan suatu template yang dapat diperluas untuk aplikasi di dalam suatu domain. Framework dapat digambarkan sebagai mikro arsitektur yang meliputi sekumpulan mekanisme yang bekerjasama untuk memecahkan suatu masalah yang umum pada suatu domain.

Origami adalah sebuah Application Framework yang bertujuan untuk memudahkan programmer/developer dalam mengembangkan aplikasi skala Enterprise. Pada versi pertamanya Origami merupakan akronim dari *Object Relational Gateway Microarchitecture*, memposisikan diri sebagai ORM Framework. Namun pada versinya yang kedua Origami telah bertransformasi menjadi sebuah *full stack application framework* yang dilengkapi dengan fitur-fitur terkini sebuah framework modern- terdiri dari modul Configuration, Microkernel (Dependency Injection Container), Data Access Abstraction, ORM (*Object Relational Mapping*), Logging, dan Unit Testing. Origami dikembangkan oleh XERIS System Interface untuk menyediakan solusi Enterprise Application Framework yang ringan (*lightweight*) di platform .NET. Framework lain yang populer untuk pengembangan Enterprise Application di platform .NET adalah Microsoft EnterpriseLib dan Spring.NET.

Origami Framework dapat didownload di www.codeplex.com/origami. Berikut ini adalah gambar dari arsitektur Origami Framework :



5.1 Data Access Abstraction

Origami menyediakan fitur Data Access Abstraction yang memudahkan penggunaan ADO.NET dengan cara membuat wrapper (pembungkus) method-method pengaksesan data seperti :

`ExecuteQuery()`, `ExecuteNonQuery()`, `ExecuteReader()`, `ExecuteScalar()`, `ExecuteDataSet()`, dan `UpdateDataSet()`.

Fitur dari Origami Data Access Abstraction :

- Menyederhanakan API pengaksesan data via ADO.NET
- Data Access Helper, Mengatur koneksi dan mengenkapsulasi method-method :
`ExecuteScalar()`, `ExecuteQuery()`, `ExecuteNonQuery()`, `ExecuteDataSet()`, `UpdateDataSet()`
- Mapping row ke objek
Mengatasi masalah *impedance-mismatch* antara dunia objek dengan relational database dengan menyediakan method-method : `ExecuteList()`, dan `ExecuteObject()`
- Generic Data Access
Menyediakan abstraksi untuk berbagai macam tipe DBMS
- Pengaksesan stored procedure
- Menyediakan wrapper untuk objek Command
- Mendukung programmatic transaction
- Konfigurasi Data Source dapat diletakkan di XML

5.1.1 Data Access Helper

Method	Keterangan
<code>ExecuteReader()</code>	Method yang digunakan untuk mengeksekusi perintah SQL SELECT Parameter : <code>Sql</code> Contoh : <code>ExecuteReader("SELECT * FROM Employees")</code> Return value : <code>IDataReader</code>
<code>ExecuteNonQuery()</code>	Method yang digunakan untuk mengeksekusi perintah SQL INSERT, UPDATE, dan DELETE Parameter : <code>Sql</code> Contoh : <code>ExecuteNonQuery("DELETE FROM Employees WHERE EmployeeID=1")</code> Tidak memiliki return value
<code>ExecuteDataSet()</code>	Method yang digunakan untuk mengambil row dari database (SQL SELECT) untuk selanjutnya disimpan ke <code>DataSet</code> Parameter : <code>Sql</code> Contoh : <code>ExecuteReader("SELECT * FROM Employees")</code>

	Return value : DataSet
ExecuteScalar()	Method yang mengembalikan nilai single value
ExecuteObject()	Method yang digunakan untuk mengambil row dari database (SQL SELECT) dengan return value object
ExecuteList()	Method yang digunakan untuk mengambil row dari database (SQL SELECT) dengan return value object collection (List<T>)
UpdateDataSet()	Melakukan perubahan data yang disimpan di dataset

Contoh :

```
DataSource dataSource = new DataSource();
dataSource.Provider="System.Data.SqlClient";
dataSource.ConnectionString="Data Source=SYSTEMINTERFACE\\SQLEXPRESS;Initial "
    + "Catalog=NWIND;Integrated Security=True";

IEntityManager em=new EntityManager(dataSource);
```

ExecuteReader()

```
string sql = "SELECT * FROM Employees";
IDataReader rdr = em.ExecuteReader(sql);
while (rdr.Read())
{
    Console.WriteLine(rdr["FirstName"]);
    Console.WriteLine(rdr["LastName"]);
    Console.WriteLine(rdr["Title"]);
}
```

ExecuteNonQuery()

```
string sql = "INSERT INTO Employees (FirstName,LastName,Title) VALUES "
    + "('Bill','Gates','Software Architect')";

em.ExecuteNonQuery(sql);
```

ExecuteDataSet()

```
string sql = "SELECT * FROM Employees";
DataSet ds = em.ExecuteDataSet(sql);

foreach (TableRow row in ds.Tables[0].Rows)
```



```

{
    Console.WriteLine(row["FirstName"]);
    Console.WriteLine(row["LastName"]);
    Console.WriteLine(row["Title"]);
}

```

Saya akan menugabah source code pada DAO Pattern pada bab sebelumnya dengan menggunakan Data Access Abstraction.

Class Customer

```

using System;

namespace OrigamiDemo.Entity
{
    public class Customer
    {
        private int customerID;
        private string companyName;
        private string contactName;
        private string address;
        private string city;
        private string postalCode;
        private string country;
        private string phone;
        private string fax;
        private string email;

        //property class
        //.....
    }
}

```

Interface ICustomerDao

```

using OrigamiDemo.Entity;
using System.Collections.Generic;

namespace OrigamiDemo.Data
{
    public interface ICustomerDao
    {
        Customer GetById(string customerId);
        List<Customer> Find();
        List<Customer> FindByName(string companyName);
        int Save(Customer customer);
        int Update(Customer customer);
        int Delete(Customer customer);
    }
}

```

CustomerDao

```
using System;
using System.Collections.Generic;
using Origami.Data.Provider;
using Origami.Data;
using Origami.Configuration;
using OrigamiDemo.Entity;

namespace OrigamiDemo.Data
{
    public class CustomerDao : ICustomerDao
    {
        private IEntityManager em;
        private string table = "Customers";
        private DataSource ds;

        public CustomerDao(DataSource dataSource)
        {
            this.em = new EntityManager(dataSource);
        }

        public Customer GetById(string customerId)
        {
            string sql = "SELECT * FROM " + table + " WHERE CustomerId='"
                + customerId + "'";
            Customer customer = em.ExecuteObject<Customer>(
                sql, new CustomerMapper());
            return customer;
        }

        public List<Customer> Find()
        {
            string sql = "SELECT * FROM " + table;
            List<Customer> customers = em.ExecuteList<Customer>(
                sql, new CustomerMapper());

            return customers;
        }

        public List<Customer> FindByName(string companyName)
        {
            string sql = "SELECT * FROM " + table + " WHERE CompanyName LIKE '%"
                + companyName + "%'";
            List<Customer> customers = em.ExecuteList<Customer>(
                sql, new CustomerMapper());

            return customers;
        }

        public int Save(Customer customer)
        {
            string sql = "INSERT INTO " + table
                + " (CustomerId,CompanyName,ContactName) "
                + "VALUES (@CustID,@CompanyName,@ContactName)";

            DbCommandWrapper cmd = em.CreateCommand(
                CommandWrapperType.Text, sql);
        }
    }
}
```

```

        cmd.SetParameter("@CustId", customer.CustomerId);
        cmd.SetParameter("@CompanyName", customer.CompanyName);
        cmd.SetParameter("@ContactName", customer.ContactName);

        return em.ExecuteNonQuery(cmd);
    }

    public int Update(Customer customer)
    {
        string sql = "UPDATE " + table + " SET "
            + "CompanyName=@CompanyName,ContactName=@ContactName "
            + "WHERE CustomerID=@CustId";

        DbCommandWrapper cmd = em.CreateCommand(
            CommandType.Text, sql);

        cmd.SetParameter("@CustId", customer.CustomerId);
        cmd.SetParameter("@CompanyName", customer.CompanyName);
        cmd.SetParameter("@ContactName", customer.ContactName);

        return em.ExecuteNonQuery(cmd);
    }

    public int Delete(Customer customer)
    {
        string sql = "DELETE FROM " + table + " WHERE CustomerID=@CustId";

        DbCommandWrapper cmd = em.CreateCommand(
            CommandType.Text, sql);

        cmd.SetParameter("@CustId", customer.CustomerId);
        return em.ExecuteNonQuery(cmd);
    }
}

```

CustomerMapper

```

using System;
using System.Data;
using Origami.Data;
using OrigamiDemo.Entity;

namespace OrigamiDemo.Data
{
    public class CustomerMapper : IRowMapper<Customer>
    {
        public Customer Map(IDataReader rdr)
        {
            Customer customer = new Customer();
            customer.CustomerId = rdr["CustomerId"].ToString();
            customer.CompanyName = rdr["CompanyName"].ToString();
            customer.ContactName = rdr["ContactName"].ToString();
            return customer;
        }
    }
}

```

Mari kita perhatikan satu persatu class yang sudah dibuat. Perhatikan class CustomerDao. Class ini merupakan implementasi dari interface ICustomerDao. Interface ICustomerDao hanya berisi deklarasi method-method manipulasi data, bagaimana hal itu nantinya diimplementasikan diserahkan ke class yang mengimplementasikannya (class kongkret).

Pemisahan interface dan class kongkret adalah disiplin pemrograman yang sangat baik. Client/class pengguna hanya mengakses interface bukan class kongkret, sehingga jika terjadi penggantian class implementasi, client tidak usah tahu sehingga tidak perlu ada perubahan code.

Class CustomerDao memiliki satu buah constructor yang memiliki parameter objek dataSource. Objek dataSource merupakan instantiasi class DataSource yang dilakukan dari program utama

```
public CustomerDao(DataSource dataSource)
{
    this.em = new EntityManager(dataSource);
}
```

5.1.2 Object Mapping

Berikutnya perhatikan method Finder di class CustomerDao. Kita memiliki 3 buah method finder yaitu : GetById(), Find(), dan FindByName(). Masing-masing memiliki return value object dan object collection.

Method Finder	Return Value	Keterangan
GetById(string customerId)	Customer	Menampilkan Customer berdasarkan CustomerId
Find()	List<Customer>	Menampilkan semua data Customer
FindByName(string companyName)	Customer	Menampilkan Customer berdasarkan nama

Semua method Finder memiliki return value objek bukan DataReader, hal ini dimungkinkan berkat fitur object mapping di Origami Data Access Abstraction yaitu method ExecuteObject() dan ExecuteList(). Kedua method ini memiliki parameter sql dan interface IRowMapper<T> . Method tersebut merupakan member dari class EntityManager

Method Object Mapping	Keterangan
ExecuteObject<T>(sql, IRowMapper<T>)	Mengubah row menjadi objek
ExecuteList<T>() (sql, IRowMapper<T>)	Mengubah kumpulan row menjadi object collection

Perhatikan source code berikut :

```
public int ExecuteNonQueryParameter(DbCommandWrapper cmdWrapper, bool
isStoredProc, string sql,
    bool isTransaction, Transaction tx)
{
```

```

int result = 0;
try
{
    DbCommand cmd = conn.CreateCommand();

    if (isTransaction)
    {
        cmd.Transaction = tx.GetTransaction;
    }

    if (isStoredProc)
    {
        cmd.CommandType = CommandType.StoredProcedure;
    }
    else
    {
        cmd.CommandType = CommandType.Text;
    }

    cmd.CommandText = sql;

    foreach (ParameterClause param in cmdWrapper.Parameters)
    {
        DbParameter sqlParam = cmd.CreateParameter();
        sqlParam.ParameterName = param.Name;
        sqlParam.Value = param.Argument;
        cmd.Parameters.Add(sqlParam);
    }

    result=cmd.ExecuteNonQuery();
    cmd.Dispose();
}
catch (DbException ex)
{
    throw new Exception(ex.Message.ToString());
}

return result;
}

```

Class kongkret dari interface `IRowMapper<T>` adalah class `CustomerMapper` yang isinya mapping row ke objek, yaitu dengan cara mengkonversi `DataReader` menjadi objek customer.

```

public class CustomerMapper : IRowMapper<Customer>
{
    public Customer Map(IDataReader rdr)
    {
        Customer customer = new Customer();
        customer.CustomerId = rdr["CustomerId"].ToString();
        customer.CompanyName = rdr["CompanyName"].ToString();
        customer.ContactName = rdr["ContactName"].ToString();
        return customer;
    }
}

```

5.1.3 Command Wrapper

Origami menyediakan wrapper untuk objek Command sehingga penggunaanya bisa lebih sederhana daripada menggunakan `OleDbCommand` atau `SqlCommand`. Dengan menggunakan Command Wrapper perintah sql akan dieksekusi menggunakan Prepared Statement yang lebih efisien untuk pemanggilan sql yang berulang-ulang. Command Wrapper mendukung juga pemanggilan Stored Procedure.

Perhatikan contoh berikut :

```
public int Save(Customer customer)
{
    string sql = "INSERT INTO " + table
        + " (CustomerId,CompanyName,ContactName) "
        + "VALUES (@CustID,@CompanyName,@ContactName) ";

    DbCommandWrapper cmd = em.CreateCommand(CommandWrapperType.Text, sql);

    cmd.SetParameter("@CustId", customer.CustomerId);
    cmd.SetParameter("@CompanyName", customer.CompanyName);
    cmd.SetParameter("@ContactName", customer.ContactName);

    return em.ExecuteNonQuery(cmd);
}
```

Jika menggunakan Stored Procedure :

```
public int Save(Customer customer)
{
    string spName = "spCustomerInsert";

    DbCommandWrapper cmd = em.CreateCommand(
        CommandWrapperType.StoredProcedure, sql);

    cmd.SetParameter("@CustId", customer.CustomerId);
    cmd.SetParameter("@CompanyName", customer.CompanyName);
    cmd.SetParameter("@ContactName", customer.ContactName);

    return em.ExecuteNonQuery(cmd);
}
```

Class DAOFactory

```
using System;
using Origami.Data.Provider;
using OrigamiDemo.Data;

namespace OrigamiDemo.Data
{
    public class DaoFactory
    {
        public static ICustomerDao GetCustomerDaoInstance (DataSource ds)
        {
            return new CustomerDao(ds);
        }
    }
}
```

Program Utama

```
using System;
using System.Collections;
using System.Collections.Generic;
using Origami.Data;
using Origami.Data.Provider;
using OrigamiDemo.Entity;
using OrigamiDemo.Data;

namespace OrigamiDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            DataSource dataSource=new DataSource();

            dataSource.Provider="System.Data.SqlClient";
            dataSource.ConnectionString =
                "Data Source=SYSTEMINTERFACE\\SQLEXPRESS;Initial "
                + "Catalog=NWIND;Integrated Security=True";

            Customer customer=new Customer();
            customer.CustomerId="XERIS";
            customer.CompanyName="XERIS System Interface";
            customer.ContactName="Ariyanto";

            ICustomerDao custDao=DaoFactory.GetCustomerDaoInstance(dataSource);
            custDao.Save(customer);

            List<Customer> list=custDao.Find();
            foreach(Customer cust in list)
            {
                Console.WriteLine(cust.CustomerId);
                Console.WriteLine(cust.CompanyName);
                Console.WriteLine(cust.ContactName);
            }
        }
    }
}
```

5.1.4 Query Builder

Fitur Query Builder disediakan untuk memberi alternatif penulisan sql pada pengaksesan ADO.NET dengan cara lama. Melalui QueryBuilder, perintah sql dienkapsulasi menjadi objek, sehingga pengguna tidak usah mengetahui detail perintah sql didalamnya. Perintah sql di *generate* secara otomatis melalui method **BuildQuery()**. API QueryBuilder terdapat pada namespace Origami.Data.Query

Cara penggunaanya sama seperti ketika menggunakan Criteria API.

Method	Keterangan	Contoh
SelectTable()	Memilih tabel tertentu	SelectTable("Customers")
AddCriteria()	Menambahkan kriteria tertentu Parameter : Criteria	AddCriteria(Criteria.Like("Name", "%XERIS"))
AddJoin()	Membuat relasi dengan menambahkan table join (Inner Join,Outer Join,Left Join,Right Join) Parameter : Join	AddJoin(Join.InnerJoin("Suppliers","Products","SupplierID","SupplierID"))
AddAggregate()	Membuat aggregate query (MIN,MAX,SUM,AVG,STDDEV) Parameter : Aggregate	AddAggregate(Aggregate.Max("Qty"))
AddGroup()	Menambahkan kolom group pada aggregate query Parameter : column	AddGroup("CategoryID");
AddOrder()	Menambahkan sorting(ASC,DESC) Parameter : column,OrderBy	AddOrder("PoductName",OrderBy.ASCENDING)
BuildQuery()	Generate query Retrun value : string	BuildQuery()

Method `BuildQuery()` menghasilkan perintah sql yang akan dieksekusi. Selanjutnya hasil dari `BuildQuery()` ini menjadi parameter `ExecuteReader()` atau `ExecuteDataSet()`

AddCriteria()

```
QueryBuilder qb = new QueryBuilder();

qb.SelectTable("Customers");
qb.AddCriteria(Criteria.Equal("CustomerID", "ALFKI"));

string sql = qb.BuildQuery();
```

Generate Query : `SELECT * FROM Customers WHERE CustomerID='ALFKI'`

```
QueryBuilder qb = new QueryBuilder();

qb.SelectTable("Employees");
qb.AddCriteria(Criteria.Equal("FirstName", "Andrew"));
qb.AddCriteria(Criteria.Equal("LastName", "Fuller"));

string sql = qb.BuildQuery();
```

Generate Query : `SELECT * FROM Employees WHERE FirstName='Andrew' AND LastName='Fuller'`

AddOrder()

```
QueryBuilder qb = new QueryBuilder();

qb.SelectTable("Suppliers");
qb.AddOrder("CompanyName", OrderBy.ASCENDING);
qb.AddCriteria(Criteria.Like("CompanyName", "%ltd%"));

string sql = qb.BuildQuery();
```

Generate Query : `SELECT * FROM Suppliers WHERE CompanyName LIKE '%ltd%' ORDER BY CompanyName ASC`

AddJoin()

```
QueryBuilder qb = new QueryBuilder();

qb.SelectTable("Products");
qb.AddJoin(Join.InnerJoin("Suppliers", "Products",
    "SupplierID", "SupplierID"));
qb.AddJoin(Join.InnerJoin("Categories", "Products",
    "CategoryID", "CategoryID"));

string sql = qb.BuildQuery();
```

Generate Query :

```
SELECT * FROM Products INNER JOIN Suppliers ON Products.SupplierID =  
Suppliers.SupplierID INNER JOIN Categories ON Products.CategoryID =  
Categories.CategoryID
```

AddAggregate()

```
QueryBuilder qb = new QueryBuilder();  
qb.SelectTable("Products");  
qb.AddAggregate(Aggregate.Sum("Price"));  
  
string sql = qb.BuildQuery();
```

Generate Query : SELECT SUM(Price) AS Price FROM Products

AddGroup()

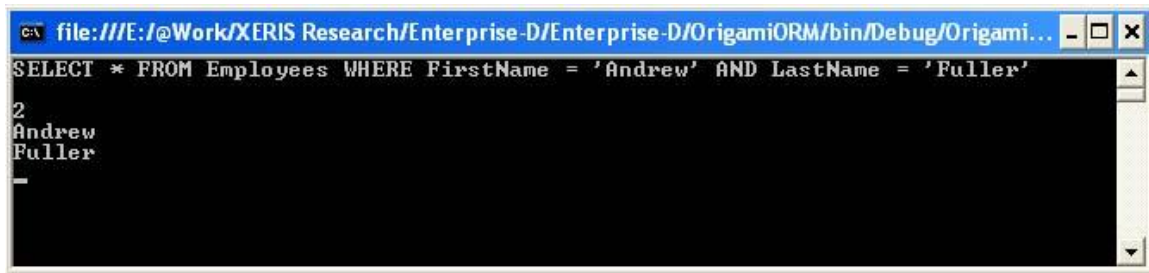
```
QueryBuilder qb = new QueryBuilder();  
qb.SelectTable("Products");  
qb.AddAggregate(Aggregate.Max("Price"));  
qb.AddGroup("CategoryID");  
  
string sql = qb.BuildQuery();
```

Generate Query : SELECT MAX(Price) AS Price FROM Products GROUP BY CategoryID

Eksekusi SQL

```
QueryBuilder qb = new QueryBuilder();  
  
qb.SelectTable("Employees");  
qb.AddCriteria(Criteria.Equal("FirstName", "Andrew"));  
qb.AddCriteria(Criteria.Equal("LastName", "Fuller"));  
  
string sql = qb.BuildQuery();  
Console.WriteLine(sql);  
Console.WriteLine();  
  
IDataReader rdr = em.ExecuteReader(sql);  
while (rdr.Read())  
{  
    Console.WriteLine(rdr["EmployeeID"]);  
    Console.WriteLine(rdr["FirstName"]);  
    Console.WriteLine(rdr["LastName"]);  
}  
rdr.Close();
```

Output :



```
file:///E:/@Work/XERIS Research/Enterprise-D/Enterprise-D/OrigamiORM/bin/Debug/Origami...
SELECT * FROM Employees WHERE FirstName = 'Andrew' AND LastName = 'Fuller'
2
Andrew
Fuller
-
```

Catatan :

QueryBuilder hanya berlaku untuk SQL SELECT, tidak untuk SQL INSERT, UPDATE, dan DELETE

5.2 Dependency Injection

Selain mengakses class `DataSource` secara langsung, Origami juga mendukung konfigurasi lewat XML . Hal ini dimungkinkan berkat fitur Origami Container yang merupakan implementasi Dependency Injection/IOC Container di Origami. Framework lain yang memiliki kemampuan ini antara lain adalah Spring.NET dan Unity.

Fitur Origami Container

- Pembuatan objek dilakukan melalui Container
- Mengatur object dependency secara runtime
- Mendukung Dependency Lookup
- Mendukung Constructor Injection dan Setter Injection
- Konfigurasi objek dalam container diletakkan di file XML
- Injection dapat dilakukan secara programmatic ataupun deklaratif dengan menggunakan *custome attribute*

Contoh konfigurasi :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="objects" type="Origami.Configuration.ConfigurationHandler,
Origami" />
  </configSections>
  <objects>
    <object id="dataSource" type="Origami.Data.Provider.DataSource, Origami">
      <property name="Provider" value="System.Data.SqlClient" />
      <property name="ConnectionString" value="Data
Source=SYSTEMINTERFSCF\SQLEXPRESS;Initial Catalog=NWIND;
Integrated Security=True" />
    </object>
  </objects>
</configuration>
```

Dengan menggunakan Origami Container, objek tidak usah di-instantiasi secara langsung di kode. Container-lah yang bertanggung jawab untuk melakukan hal tersebut, kita cukup mendaftarkan objek ke container dengan cara mendeklarasikan objek apa yang dibutuhkan beserta property nya pada file App.config. Selanjutnya “bukan sihir, bukan sulap”, objek tersebut langsung bisa kita pakai di aplikasi bersangkutan. Menariknya, jika ada perubahan pada pemanggilan objek atau pengaturan property nya, cukup di edit di konfigurasi XML nya tanpa harus menyentuh source code sama sekali.

Dalam Object Oriented Principle, Dependency Injection/IoC merupakan penerapan prinsip “*Dependency Inversion*” yang menjamin kode bisa loosely-coupled untuk mencegah saling ketergantungan dengan kode yang lain. Origami Microkernel mendukung *Dependency Lookup*, *Constructor Injection*, dan *Setter Injection*. Kita akan bahas dibagian lain ketiga istilah teknis ini.

Instantiasi dataSource lewat Container

```
IOrigamiContainer container = new OrigamiContainer();  
DataSource dataSource = Container.GetObject<DataSource>("dataSource");
```

Cara kerja :

1. Container disiapkan dengan pemanggilan class kongkret dari interface IOrigamiContainer. Semua objek yang ingin diletakkan di Container harus dideklarasikan di App.config
2. Setiap objek ditandai oleh id tertentu. Objek dengan id “dataSource” merujuk ke class DataSource di namespace Origami.Data.Provider. Jika ditemukan, otomatis objek tersebut akan diinstantiasi
3. Jika objek tersebut memiliki setting-an property, maka Container akan meng-inject value dari property yang bersangkutan

```
ICustomerDao custDao = DaoFactory.GetCustomerDaoInstance(dataSource);
```

Jika kita perhatikan class CustomerDao pada contoh Data Access Abstraction, CustomerDao memiliki dependency/ketergantungan terhadap class DataSource. Ketergantungan tampak di constructor class CustomerDao

```
public class CustomerDao : ICustomerDao  
{  
    private IEntityManager em;  
    private string table = "Customers";  
    private DataSource ds;  
  
    public CustomerDao(DataSource dataSource)  
    {  
        this.em = new EntityManager(dataSource);  
    }  
}
```

Origami Container dapat mengelola ketergantungan class dengan menyediakan fitur constructor injection dan setter injection. Untuk mengakses fitur microkernel ini diperlukan class Container pada namespace `Origami.Container`

```
IOrigamiContainer container = new OrigmiContainer();
```

Interface IContainer memiliki method-method berikut :

Method	Keterangan
<code>GetObject<T>(string objected)</code>	Mengambil objek yang sudah terigistrasi di container
<code>GetObject<T,E>(IDependencyInjection dependencyInjection)</code>	Mengambil objek dan melakukan injection secara programmatic objek yang menjadi dependency nya, baik dengan menggunakan constructor injection atau setter injection
<code>GetObject<T,E>(string objectId)</code>	Declarative constructor injection
<code>GetObject<T,E>(string objectId, string propertyName)</code>	Declarative setter injection

5.2.1 Programmatic Injection

Origami mendukung dua cara dalam menangani ketergantungan, yaitu lewat programmatic injection dan declarative injection. Programatic injection melakukan injection langsung pada source code tanpa menggunakan custom attribute. Berikut contohnya :

Constructor injection

```
IOrigamiContainer container = new OrigmiContainer();

ICustomerDao custDao=container.GetObject<CustomerDao, DataSource>(
    new ConstructorInjection("custDao", "dataSource")
);
```

Konfigurasi App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="objects" type="Origami.Configuration.ConfigurationHandler,
Origami" />
  </configSections>
  <objects>
    <object id="dataSource" type="Origami.Data.Provider.DataSource,Origami">
      <property name="Provider" value="System.Data.SqlClient" />
      <property name="ConnectionString" value="Data
Source=SYSTEMINTERFSCE\SQLEXPRESS;Initial Catalog=NWIND;
```

```

        Integrated Security=True" />
    </object>
    <object id="custDao" type="OrigamiDemo.Data.CustomerDao,OrigamiDemo">
    </objects>
</configuration>

```

Constructor injection adalah teknik dari DI (*Dependency Injection*) yang melewati dependency object melalui constructor

```
ConstructorInjection(string objectId, string dependencyId)
```

Container akan mencari apakah objek dengan id “custDao” sudah terregistrasi. Jika sudah container menginstantiasi class yang merujuk ke object id tersebut yaitu `CustomerDao()` yang berada di namespace `OrigamiDemo.Data`. Ketika di instantiasi ternyata constructor `CustomerDAO()` memiliki dependency yaitu objek `dataSource` yang bertipe `DataSource`. Container secara otomatis akan menginject objek tersebut berdasarkan referensi objek id “dataSource” di container.

Setter Injection

Setter injection melewati dependency object melalui “setter” di property

CustomerDao

```

public class CustomerDao
{
    private IEntityManager em;
    private string table = "Customers";
    private DataSource ds;

    public DataSource DataSource
    {
        get { return ds; }
        set
        {
            ds = value;
            this.em = new EntityManager(ds);
        }
    }
}

```

Main

```

IOrigamiContainer container = new OrigmiContainer();
ICustomerDao custDao = container.GetObject<ICustomerDao, DataSource>(
    new SetterInjection("custDao", "dataSource", "DataSource")
);

```

Class SetterInjection :

```
SetterInjection(string objectId, string dependencyId, string propertyName)
```

objectId = object id di container

dependencyId = object id yang menjadi dependency

propertyName = Property yang hendak di Inject

5.2.2 Declarative Injection

Declarative injection memungkinkan injection dilakukan melalui custom attribute. Origami Microkernel menggunakan attribute “[Dependency]” untuk menyuntikkan ketergantungan. Penggunaan declarative injection lebih simple daripada programmatic injection karena code yang ditulis menjadi lebih sedikit. Perhatikan contoh berikut :

CustomerDao

```
using Origami.Container;

public class CustomerDao
{
    private IEntityManager em;
    private string table = "Customers";
    private DataSource ds;

    [Dependency(Name="dataSource")]
    public CustomerDao(DataSource dataSource)
    {
        this.em = new EntityManager(dataSource);
    }
}
```

Attribute “Dependency” diisi “dataSource” yang merupakan object id yang sudah didefinisikan di App.config. Ingat “dataSource” ini merujuk ke class Origami.Data.Connection.DataSource. Karena dependency diletakkan di constructor, maka injection yang akan dilakukan adalah constructor injection.

Main

```
IOrigamiContainer container = new OrigamiContainer();
ICustomerDao custDao = container.GetObject<ICustomerDao, DataSource>("custDao");
```

5.2.3 Data Access Abstraction Menggunakan Dependency Injection

Setelah memahami konsep dan penerapan Dependency Injection, saya akan menunjukkan penggunaannya di Data Access Abstraction yang sudah dibuat sebelumnya. Ubahlah CustomerDao dan Main pada contoh sebelumnya menjadi seperti berikut :

```
using System;
using System.Collections.Generic;
using Origami.Data.Provider;
using Origami.Data;
using Origami.Container;
using OrigamiDemo.Entity;

namespace OrigamiDemo.Data
{
    public class CustomerDao : ICustomerDao
    {
        private IEntityManager em;
        private string table = "Customers";
        private DataSource ds;

        [Dependency(Name="dataSource")]
        public CustomerDao(DataSource dataSource)
        {
            this.em = new EntityManager(dataSource);
        }

        //Method GetById

        //Method Find

        //Method FindByName

        //Method Save

        //Method Update

        //Method Delete
    }
}
```

Program Utama

```
using System;
using System.Collections;
using System.Collections.Generic;
using Origami.Data;
using Origami.Data.Provider;
using Origami.Container;
using OrigamiDemo.Entity;
using OrigamiDemo.Data;

namespace OrigamiDemo
{
}
```



```

class Program
{
    static void Main(string[] args)
    {
        //simpan data

        Customer customer=new Customer();
        customer.CustomerId="XERIS";
        customer.CompanyName="XERIS System Interface";
        customer.ContactName="Ariyanto";

        IOrigamiContainer container = new OrigmiContainer();
        ICustomerDao custDao = container.GetObject<ICustomerDao,
            DataSource>("custDao");

        custDao.Save(customer);

        //tampilkan data

        List<Customer> list=custDao.Find();
        foreach(Customer cust in list)
        {
            Console.WriteLine(cust.CustomerId);
            Console.WriteLine(cust.CompanyName);
            Console.WriteLine(cust.ContactName);
        }
    }
}

```

Dibandingkan dengan kode sebelumnya, program utama (Main) sekarang menjadi lebih singkat. Bagian untuk memanggil `DataSource` sudah dihilangkan, dan `CustomerDao` tidak lagi memanggil `DaoFactory`. Kedua class ini diinstantiasi secara otomatis oleh container.

BAB 6

Object Relational Mapping (ORM)

ORM adalah framework yang memetakan tabel pada relational database ke objek dengan menggunakan informasi pada metadata. Metadata dapat berupa file XML atau *custom attribute* pada *entity class*. Kian hari ORM tools makin populer karena dapat meningkatkan produktivitas developer. Ketika kita membangun sebuah aplikasi database dengan menggunakan OOP, maka harus dibuatkan sebuah persisten objek yang memodelkan tabel, view dan relationship yang terdapat di database tersebut.

6.1 Apa itu Object Persistence?

Hampir semua aplikasi membutuhkan persistensi data. Jika tidak, ketika aliran listrik dimatikan maka data tersebut akan hilang. Penyimpanan ini bisa dilakukan di file, registry, atau di database. Pada aplikasi yang menggunakan database, persistensi berarti penyimpanan data ke tabel. Relational Database menyediakan representasi terstruktur dari persisten data yang memungkinkan manipulasi, sortir, pencarian, dan agregasi data. Ketika memodelkan persistence object, makin besar aplikasi yang kita buat, maka makin rumit pemodelan yang dilakukan. Pemodelan dengan "tangan kosong" bukanlah ide yang baik. Selain banyak waktu yang terbuang karena banyak pekerjaan yang di lakukan berulang-ulang, bisa jadi banyak kesalahan yang tidak bisa dihindari.

Untuk mengatasi "*hand coding*" ini bisa digunakan 2 alternatif. Pertama dengan menggunakan Code Generator. Persistence objek bisa digenerate berdasarkan informasi schema dari database. Source code yang dihasilkan kemudian bisa di edit, disesuaikan dan diintegrasikan dengan kode lain. Tools seperti ini banyak tersedia di Internet, contoh nya CodeSmith dan CodeWeaver. Kelemahan dari code generator adalah kita masih harus memaintenance source code dari persistent objek nya Pendekatan yang kedua yaitu dengan menggunakan ORM. ORM sebenarnya adalah sebuah code generator yang meng generate SQL CRUD (Create, Read, Update, Delete) berdasar informasi meta data pada entity class atau dari konfigurasi XML.

Perbedaanya dengan code generator biasa, ORM meng-generate code secara runtime, sehingga code yang ditulis tetap ramping dan lebih singkat. Dibandingkan code generator, pendekatan ORM bisa jadi lebih efektif dan produktif, tetapi tetap ada trade-off yang harus ditanggung yaitu eksekusi program cenderung lebih lambat daripada tanpa menggunakan ORM. Hal ini diakibatkan adanya proses mapping terlebih dahulu terhadap tabel di basis data. Proses ini mengubah "record/tupel" menjadi objek atau objek collection. Namun dengan perkembangan kecepatan processor dan kapasitas memory saat ini trade-off tersebut dapat diatasi.

Solusi ORM minimal harus mengandung 4 hal berikut :

- a. API untuk melakukan operasi CRUD pada objek di persistent class
- b. Menyediakan mekanisme untuk melakukan query terhadap objek
- c. Fasilitas untuk spesifikasi metadata untuk melakukan mapping
- d. Mendukung operasi transactional object

Berikut adalah alasan mengapa menggunakan ORM :

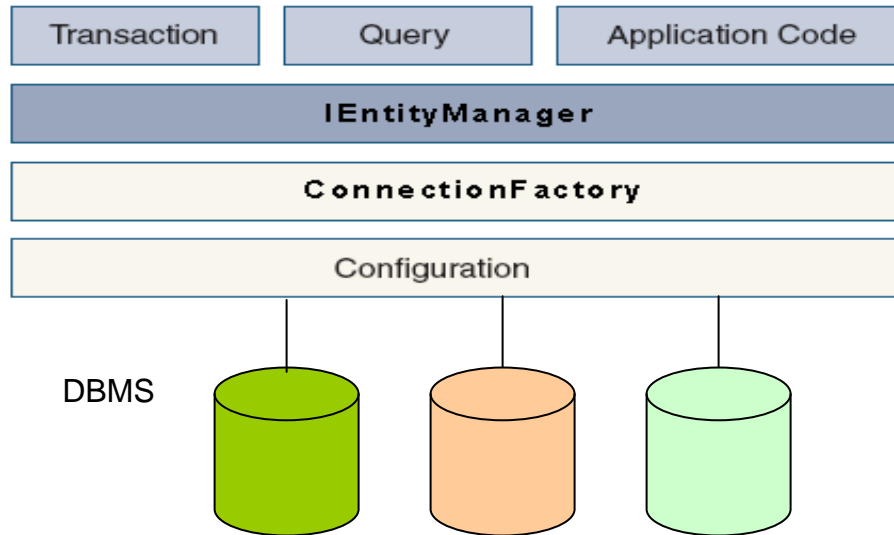
- a. Productivity : Mengurangi *hand code* dan menghilangkan layer pengaksesan data (DAO/*Data Access Object*)
- b. Maintainability : Kode lebih mudah di maintenance, karena tidak perlu lagi menuliskan perintah SQL bersama-sama dengan kode utama, bahkan tidak perlu menuliskan SQL sama sekali.
- c. Performance : Kode lebih efisien sehingga bisa dioptimasi lebih lanjut
- d. Vendor Independence : Tidak tergantung penyedia DBMS. Jika terjadi perubahan database cukup diubah konfigurasi koneksi saja
- e. Mengatasi Paradigm Mismatch : Penyimpanan data dalam RDBMS dan objek sama sekali berbeda . ORM menjadi layer perantara RDBMS dengan pemrograman berbasis objek.

Fitur Origami ORM

- a. Melakukan operasi CRUD (Create, Read, Update, Delete) secara otomatis
- b. Mendukung metadata di entity class dengan menggunakan custom attribute
- c. Mendukung DBMS yang beraneka ragam
- d. Object query
Query dengan pendekatan object oriented.
- e. Query builder
Men-generate query INNER JOIN secara runtime
- f. Native query
Mendukung native SQL DBMS yang bersangkutan
- g. Mendukung relasi : One-to-one, one-to-many, dan many-to-one
- h. Transformasi objek ke format lain seperti text, xml, dan excel
- i. Menampilkan metadata sebuah tabel secara runtime
- j. Mendukung stored procedure
- k. Mendukung classic ADO.NET
- l. Mendukung programatic transaction

6.2 Arsitektur Origami ORM

Karena alasan kemudahan dan performance, Origami tidak menggunakan XML dalam mendefinisikan mapping entity nya. Origami menggunakan custom attribute yang ditulis pada entity class. Dengan memanfaatkan fitur reflection pada .NET informasi metadata berupa atribut ini digunakan untuk membangkitkan perintah-perintah SQL yang bersesuaian. Hasil dari eksekusi perintah SQL yang berupa row/tupel dari tabel di DBMS akan dimapping ke objek dan disimpan dalam sebuah collection. Arsitektur Origami dapat dilihat digambar berikut :



Configuration	: Pengaturan Data Source di Container
ConnectionFactory	: Connection manager
IEntityManager	: Interface yang mengatur operasi-operasi dasar pada database seperti CRUD (<i>Create, Read, Update, Delete</i>), query, transaction, stored procedure dan lain-lain
Transaction	: Pengaturan Transaksi (<i>commit, rollback</i>)
Query	: Pencarian berdasarkan kriteria tertentu
Application Code	: Kode aplikasi

Origami mendukung berbagai macam tipe DBMS (*Database Management System*) seperti : SQL Server, MySQL, PostgreSQL, Oracle, DB2, Informix, Paradox, AS 400 dan database lainnya. Untuk database diluar produk Microsoft biasanya membutuhkan library tambahan atau sebuah .NET Connector. Jika tidak tersedia, bisa menggunakan OLEDB (`System.Data.OleDb`) atau ODBC (`System.Data.Odbc`).

6.3 Perbandingan ADO.NET dengan ORM

Akses database menggunakan ADO.NET :

```
string connStr = "Data Source=SYSTEMINTERFACE\\SQLEXPRESS;"
                + "Initial Catalog=NWIND;Integrated Security=True";

SqlConnection conn = new SqlConnection(connStr);
conn.open();

string sql = "SELECT * FROM Categories";
SqlCommand cmd = new SqlCommand(sql, conn);
SqlDataReader rdr = cmd.ExecuteReader();

while (rdr.Read())
```

```
{
    Console.WriteLine(rdr["CategoryName"] + "-" + rdr["Description"]);
}
```

Jika menggunakan ORM (Origami) :

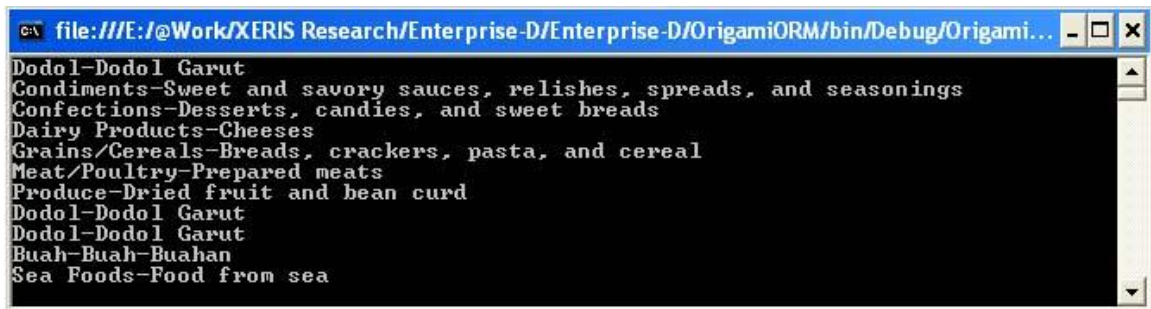
```
DataSource dataSource = new DataSource();

dataSource.Provider = "System.Data.SqlClient";
dataSource.ConnectionString = "Data Source=SYSTEMINTERFACE\\SQLEXPRESS;"
    + "Initial Catalog=NWIND;Integrated Security=True";

IEntityManager em = ConnectionFactory.Build(dataSource);

List<Category> categories = em.Get<Category>();
foreach (Category category in categories)
{
    Console.WriteLine(category.CategoryName + "-" + category.Description);
}
```

Output :



Pada contoh diatas, solusi ADO.NET menggunakan pendekatan manipulasi *row* pada database. Class yang digunakan adalah `SqlConnection`, `SqlCommand` dan `SqlDataReader`. Pendekatan Origami terasa lebih simple karena hanya menggunakan class `DataSource` dan sebuah interface `IEntityManager`. ORM melakukan manipulasi terhadap *object collection*, bukan lagi *row* karena *row* (record) pada RDBMS telah di mapping ke objek

6.4 Dasar Mapping

Mapping dilakukan pada sebuah Entity class. Entity class ini representasi entitas di domain permasalahan/*business domain*, biasanya berisi property dari entitas.

```

using System;

namespace OrigamiORM.Entity
{
    public class Category
    {
        private int categoryID;
        private string categoryName;
        private string description;

        public int CategoryID
        {
            get { return categoryID; }
            set { categoryID = value; }
        }


        public string CategoryName
        {
            get { return categoryName; }
            set { categoryName = value; }
        }

        public string Description
        {
            get { return description; }
            set { description = value; }
        }
    }
}


```

Entity class diatas selanjutnya dimapping ke tabel dan kolom database bersangkutan dengan menggunakan *customer attribute*. Mapping ini menggunakan namespace `Origami.Data.Mapping`.


Tabel Categories

Table - dbo.Categories*		Summary	
Column Name	Data Type	Allow Nulls	
 CategoryID	int	<input type="checkbox"/>	
CategoryName	varchar(15)	<input checked="" type="checkbox"/>	
Description	text	<input checked="" type="checkbox"/>	

Tabel Suppliers

Table - dbo.Suppliers			
Summary			
	Column Name	Data Type	Allow Nulls
	SupplierID	int	<input type="checkbox"/>
	CompanyName	varchar(40)	<input checked="" type="checkbox"/>
	ContactName	varchar(30)	<input checked="" type="checkbox"/>
	Address	varchar(60)	<input checked="" type="checkbox"/>
	City	varchar(15)	<input checked="" type="checkbox"/>
	Country	varchar(15)	<input checked="" type="checkbox"/>
	Phone	varchar(24)	<input checked="" type="checkbox"/>

Tabel Products

Table - dbo.Products			
Summary			
	Column Name	Data Type	Allow Nulls
	ProductID	int	<input type="checkbox"/>
	ProductName	varchar(40)	<input checked="" type="checkbox"/>
	SupplierID	int	<input checked="" type="checkbox"/>
	CategoryID	int	<input checked="" type="checkbox"/>
	QuantityPerUnit	varchar(20)	<input checked="" type="checkbox"/>
	UnitPrice	money	<input checked="" type="checkbox"/>
	UnitsInStock	int	<input checked="" type="checkbox"/>

Class Category

```
using System;
using Origami.Data.Mapping;

namespace OrigamiORM.Entity
{
    [Table(Name="Categories")]
    public class Category
    {
        private int categoryID;
        private string categoryName;
        private string description;

        [Id(Name="CategoryId", IsIdentity=true)]
        public int CategoryID
        {
            get { return categoryID; }
            set { categoryID = value; }
        }
    }
}
```

```

        [Column(Name="CategoryName")]
        public string CategoryName
        {
            get { return categoryName; }
            set { categoryName = value; }
        }

        [Column(Name="Description")]
        public string Description
        {
            get { return description; }
            set { description = value; }
        }
    }
}

```

Class Supplier

```

using System;
using Origami.Data.Mapping;

namespace OrigamiORM.Entity
{
    [Table(Name = "Suppliers")]
    public class Supplier
    {
        private int supplierId;
        private string companyName;
        private string contactName;
        private string address;
        private string city;
        private string country;
        private string phone;

        [Id(Name = "SupplierId", IsIdentity=true)]
        public int SupplierID
        {
            get { return supplierId; }
            set { supplierId = value; }
        }

        [Column(Name = "CompanyName")]
        public string CompanyName
        {
            get { return companyName; }
            set { companyName = value; }
        }

        [Column(Name = "ContactName")]
        public string ContactName
        {
            get { return contactName; }
            set { contactName = value; }
        }

        [Column(Name = "Address")]
        public string Address
        {
            get { return address; }

```



```

        set { address = value; }
    }

    [Column(Name = "City")]
    public string City
    {
        get { return city; }
        set { city = value; }
    }

    [Column(Name = "Country")]
    public string Country
    {
        get { return country; }
        set { country = value; }
    }

    [Column(Name = "Phone")]
    public string Phone
    {
        get { return phone; }
        set { phone = value; }
    }
}

```

Class Product

```

using System;
using Origami.Data.Mapping;

namespace OrigamiORM.Entity
{
    [Table(Name = "Products")]
    public class Product
    {
        private int productID;
        private string productName;
        private int categoryID;
        private Category category;
        private int supplierID;
        private Supplier supplier;
        private decimal unitPrice;
        private int unitInStock;

        [Id(Name = "ProductID", IsIdentity=true)]
        public int ProductID
        {
            get { return productID; }
            set { productID = value; }
        }

        [Column(Name = "ProductName")]
        public string ProductName
        {
            get { return productName; }
            set { productName = value; }
        }
    }
}

```

```

[Column(Name="CategoryID")]
public int CategoryID
{
    get { return categoryID; }
    set { categoryID = value; }
}

[Association(EntityRef = typeof(Category),Key="CategoryID")]
public Category Category
{
    get { return category; }
    set { category = value; }
}

[Column(Name="SupplierID")]
public int SupplierID
{
    get { return supplierID; }
    set { supplierID = value; }
}

[Association(EntityRef = typeof(Supplier),Key="SupplierID")]
public Supplier Supplier
{
    get { return supplier; }
    set { supplier = value; }
}

[Column(Name = "UnitPrice")]
public decimal UnitPrice
{
    get { return unitPrice; }
    set { unitPrice = value; }
}

[Column(Name = "UnitsInStock")]
public int UnitInStock
{
    get { return unitInStock; }
    set { unitInStock = value; }
}
}
}

```

Berikut penjelasan attribute nya :

Attribute	Keterangan
[Table]	Menspesifikasikan nama table di database
[Id]	<p>Menspesifikasikan kolom yang menjadi primary key</p> <p>Parameter :</p> <p>Name : Nama kolom</p> <p>IsIncrement : ID degenerate otomatis</p> <p>IsIdentity : ID adalah identity</p>

[Column]	Menspesifikasikan kolom dari tabel bersangkutan Parameter : Name : Nama Kolom
[Association]	Menspesifikasikan relasi antara class Parameter : EntityRef : Class yang berelasi Key : Kolom yang menjadi foreign key

Attribut diberikan pada deklarasi property class, bukan pada atribut nya. Nama kolom tabel boleh berbeda dengan nama property

6.5 Identitas Objek

Identitas objek di mapping ke kolom yang menjadi primary key. Origami memiliki generator ID dengan tipe INCREMENT, dan IDENTITY.

Berikut adalah penjelasannya :

INCREMENT = ID di generate secara otomatis oleh ORM
IDENTITY = Kolom yang menjadi ID memiliki property auto number di DBMS

6.6 Membuat Koneksi ke Database

Origami didesain tidak terikat database tertentu (*database independent*), karena itu perlu diatur terlebih dahulu konfigurasi koneksinya. Berikut pengaturannya :

```
DataSource dataSource = new DataSource();

dataSource.Provider = "System.Data.SqlClient";
dataSource.ConnectionString = "Data Source=SYSTEMINTERFACE\\SQLEXPRESS;"
+ "Initial Catalog=NWIND;Integrated Security=True";
```

Pengaturan Konfigurasi koneksi terdapat pada class Configuration di namespace Origami. Configuration, property nya adalah sebagai berikut :

Provider : Mengeset provider database yang digunakan
ConnectionString : Mengeset connection string dari database bersangkutan

Berikut adalah beberapa provider untuk database populer :

Nama Database	Provider Namespace	Connection String
SQL Server	System.Data.SqlClient	Data Source=myServerAddress;Initial Catalog=myDataBase;User Id=myUsername;Password=myPassword;
Oracle 9i	System.Data.OracleClient	Data Source=MyOracleDB;User Id=myUsername;Password=myPassword;Integrated Security=no;
MySQL	System.Data.MySqlClient	Server=myServerAddress;Database=myDataBase;Uid=myUsername;Pwd=myPassword;
PostgreSQL	System.Data.OleDb	User ID=root;Password=myPassword;Host=localhost;Port=5432;Database=myDataBase;Pooling=true;Min Pool Size=0;Max Pool Size=100;Connection Lifetime=0;
IBM DB2 (OLE DB)	System.Data.OleDb	Provider=DB2OLEDB;Network Transport Library=TCPIP;Network Address=XXX.XXX.XXX.XXX;Initial Catalog=MyCtlg;Package Collection=MyPkgCol;Default Schema=Schema;User ID=myUsername;Password=myPassword;
IBM Informix	IBM.Data.Informix.IfxConnection	Database=myDataBase;Host=192.168.10.10;Server=db_engine_tcp;Service=1492;Protocol=onsoctcp;UID=myUsername;Password=myPassword;
AS 400/iSeries (OLE DB)	System.Data.OleDb	Provider=IBMDA400;Data Source=MY_SYSTEM_NAME;User Id=myUsername;Password=myPassword;
Paradox (OLE DB)	System.Data.OleDb	Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\myDb;Extended Properties=Paradox 5.x;

Kunjungi <http://connectionstrings.com> untuk memperoleh informasi mengenai berbagai macam tipe koneksi pada platform .NET.

Selain mengakses class `DataSource` secara langsung di kode program, `Data Source` bisa juga di konfigurasi lewat XML . Hal ini dimungkinkan berkat fitur *Oigami Container* yang merupakan implementasi *Dependency Injection/IoC Container* di *Origami*.

Contoh konfigurasi :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="objects" type="Origami.Configuration.ConfigurationHandler,
Origami" />
  </configSections>
  <objects>
    <object id="dataSource" type="Origami.Data.Provider.DataSource,Origami">
      <property name="Provider" value="System.Data.SqlClient" />
      <property name="ConnectionString" value="Data
Source=SYSTEMINTERFACE\SQLEXPRESS;Initial Catalog=NWIND;
Integrated Security=True" />
    </object>
  </objects>
</configuration>
```

Instantiasi dataSource lewat Container

```
IOrigamiContainer container = new OrigamiContainer();
DataSource dataSource = container.GetObject<DataSource>("dataSource");
```

Untuk membuka koneksi ke database, gunakan static class ConnectionFactory

```
IEntityManager em = ConnectionFactory.Build(dataSource);
```

selanjutnya, operasi ke database dilakukan lewat interface IEntityManager yang terdapat pada namespace Origami.Data.

6.7 Operasi CRUD (Create, Read, Update, Delete)

Pada aplikasi database operasi CRUD (Create,Read,Update,Delete) merupakan operasi standar yang harus ada. Create adalah operasi penambahan row baru ke tabel (SQL INSERT), Read adalah operasi pembacaan row dari tabel (SQL SELECT), Update adalah operasi untuk mengubah row di tabel (SQL UPDATE), dan Delete adalah operasi untuk menghapus row dari tabel (SQL DELETE). Ada perubahan paradigma dalam penggunaan ORM dibanding ADO.NET. Jika pada ADO.NET kita mengakses row, pada ORM yang kita manipulasi adalah objek, karena row telah di mapping ke objek.

Menambah objek :

```
Category category = new Category();
category.CategoryName = "Sea Foods";
category.Description = "Food from sea";
em.Save(category);
```

Mengubah objek :

```
Category category = new Category();
category.CategoryID = 1;
category.CategoryName = "Sea Foods and Drinks";
category.Description = "Foods and drinks from sea";

em.Update(category);
```

Perhatian : Untuk operasi Save dan Update semua property yang diisi harus lengkap walaupun nilainya kosong.

Menghapus objek :

```
Category category = new Category();
category.CategoryID = 1;
em.Delete(category);
```

Mengakses objek :

```
Category category = em.Get<Category>(2);
Console.WriteLine(category.CategoryName);
Console.WriteLine(category.Description);
```

Kode diatas adalah menampilkan data dengan Category ID=1, jika ingin menampilkan semua data Category :

```
List<Category> categories = em.Get<Category>();
foreach (Category category in categories)
{
    Console.WriteLine(category.CategoryName);
    Console.WriteLine(category.Description);
}
```

Jika sebuah entity memiliki relasi aggregate class seperti pada class Product (aggregate class : Category dan Supplier), manipulasi objeknya adalah sebagai berikut :

Create

```
Product product = new Product();

product.ProductName = "Chicken Fries Combo";
product.CategoryID = 1;
product.SupplierID = 2;
```

```
product.UnitPrice = 40;
product.UnitInStock = 100;

em.Save(product);
```

Read

```
List<Product> products = em.Get<Product>();
Console.WriteLine(products.Count + " Products Found \n");

foreach (Product p in products)
{
    Console.WriteLine("Product : " + p.ProductName);
    Console.WriteLine("Supplier : " + p.Supplier.CompanyName);
    Console.WriteLine("Category : " + p.Category.CategoryName);
    Console.WriteLine();
}
```

Output



```
C:\file:///E:/@Work/XERIS Research/Enterprise-D/Enterprise-D/OrigamiORM/bin/Debug/Origami...
69 Products Found

Product : Chicken Legs
Supplier : Grandma Kelly's Homestead
Category : Condiments

Product : Chang
Supplier : Exotic Liquids
Category : Dodol

Product : Aniseed Syrup
Supplier : Exotic Liquids
Category : Condiments

Product : Chef Anton's Cajun Seasoning
Supplier : New Orleans Cajun Delights
Category : Condiments
```

Salah satu kelebihan desain berbasis objek adalah kita bisa menampilkan sekaligus property dari objek-objek lain yang berelasi mulai dari class induknya, kemampuan ini disebut *object graph navigation*. Contoh, untuk menampilkan nama Supplier cukup ditulis `p.Supplier.CompanyName`. Jika ingin menampilkan alamat dari Supplier : `p.Supplier.Address`.

Product berdasarkan ID :

```
Product product = em.Get<Product>(1);
Console.WriteLine("Product Name : " + product.ProductName);
Console.WriteLine("Supplier : " + product.Supplier.CompanyName);
Console.WriteLine("Category : " + product.Category.CategoryName);
```

Output



Update

```
Product product = new Product();  
  
product.ProductID = 1;  
product.ProductName = "Chicken Legs";  
product.CategoryID = 2;  
product.SupplierID = 3;  
product.UnitPrice = 40;  
product.UnitInStock = 100;  
  
em.Update(product);
```

Delete

```
Product product = new Product();  
product.ProductID = 1;  
em.Delete(product);
```

6.8 Transaction

Transaksi didefinisikan sebagai himpunan satu atau lebih pernyataan yang dieksekusi sebagai satu unit, dengan demikian dalam suatu transaksi himpunan pernyataan harus dilaksanakan atau tidak sama sekali

Pada implementasinya ketika kita ingin mengeksekusi perintah SQL secara bersamaan, jika ada satu perintah yang gagal semua perintah SQL tersebut harus dibatalkan (*rollback*), jika sukses semua dieksekusi (*commit*).

```
Transaction tx = null;  
try  
{  
    tx = em.BeginTransaction();  
  
    Category category = new Category();  
    category.CategoryName = "Sea Foods";  
    category.Description = "Food from sea";  
    em.Save(category);  
  
    Supplier supplier = new Supplier();
```



```

supplier.CompanyName = "Field Corn Ltd.";
supplier.Address = "Toronto";

em.Save(supplier, tx);

tx.Commit();

}
catch (Exception ex)
{
    ex.Message.ToString();
    tx.Rollback();
}

```

6.9 Query

Setelah objek berhasil disimpan, operasi yang paling banyak dilakukan selanjutnya adalah menampilkan objek tersebut sesuai dengan kriteria tertentu yang diminta user. Origami menyediakan 3 cara untuk melakukan query : melalui criteria query, native query, dan OQL (*Object Query Language*) .

Masing-masing pendekatan memiliki kelebihan dan kekurangan. Penerapannya disesuaikan dengan context query yang diminta. Khusus untuk OQL, Origami bisa memanfaatkan fitur LINQ (*Language Integrated Query*). Pembahasan mengenai integrasi Origami dengan LINQ bisa dibaca di Bab 7. Untuk melakukan query harus menyertakan namespace `Origami.Data.Query` terlebih dahulu..

6.9.1 Criteria Query

```

ObjectQuery q = em.CreateQuery<Category>();
q.AddCriteria(Criteria.Equal("CategoryName", "Dairy Products"));

List<Category> categories = q.List<Category>();
foreach (Category category in categories)
{
    Console.WriteLine(category.CategoryName);
    Console.WriteLine(category.Description);
}

```



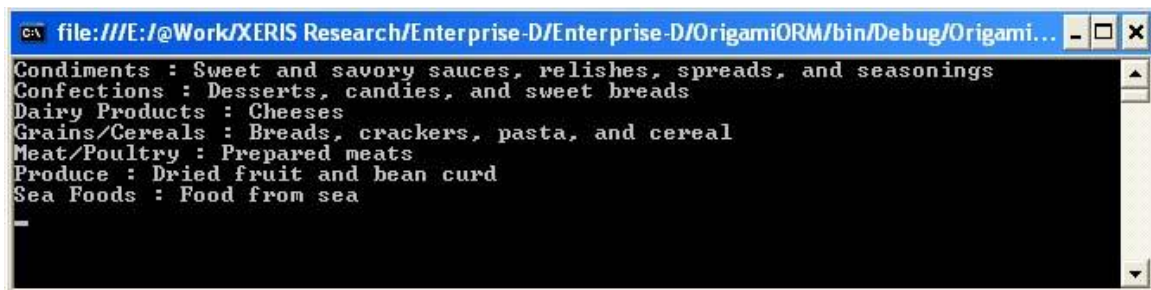
Criteria yang ditambahkan bisa lebih dari satu

Sorting :

```
ObjectQuery q = em.CreateQuery<Category>();
q.AddOrder("CategoryName", OrderBy.ASCENDING);

List<Category> categories = q.List<Category>();
foreach (Category category in categories)
{
    Console.WriteLine(category.CategoryName + "-" + category.Description);
}
```

Order yang ditambahkan bisa lebih dari satu



```
file:///E:/@Work/XERIS Research/Enterprise-D/Enterprise-D/OrigamiORM/bin/Debug/Origami...
Condiments : Sweet and savory sauces, relishes, spreads, and seasonings
Confections : Desserts, candies, and sweet breads
Dairy Products : Cheeses
Grains/Cereals : Breads, crackers, pasta, and cereal
Meat/Poultry : Prepared meats
Produce : Dried fruit and bean curd
Sea Foods : Food from sea
```

Searching :

```
ObjectQuery q = em.CreateQuery<Category>();
q.AddCriteria(Criteria.Like("CategoryName", "%Food%"));

List<Category> categories = q.List<Category>();
foreach (Category category in categories)
{
    Console.WriteLine(category.CategoryName);
    Console.WriteLine(category.Description);
}
```



```
file:///E:/@Work/XERIS Research/Enterprise-D/Enterprise-D/OrigamiORM/bin/Debug/Origami...
Sea Foods
Food from sea
```

Program diatas secara berurutan menghasilkan perintah SQL :

```
SELECT * FROM Categories WHERE CategoryName ='Dairy Products'
SELECT * FROM Categories ORDER BY CategoryName
SELECT * FROM Categories WHERE CategoryName LIKE '%Food%'
```

Method **AddCriteria()** dari class Query memiliki parameter static class Criteria. Static class Criteria sendiri memiliki method-method sebagai berikut .

Parameter :

Criteria	<p>Static class yang berisi method-method untuk operator comparison query. Method yang tersedia .:</p> <pre>Equal<T> (=) NotEqual<T> (<>) Like (LIKE) NotLike (NOT LIKE) GreaterThan<T> (>) GreaterEqualThan<T> (>=) LessThan<T> (<) LessEqualThan<T> (<=) Between<T>(BETWEEN) NotBetween<T>(NOT BETWEEN) In<T>(In) NotIn<T>(NOT IN) IsNull (IS NULL) IsNotNull (IS NOT NULL)</pre>
-----------------	---

Sedangkan method `AddOrder()` memiliki parameter : property name, dan order by (memiliki tipe enumerasi dengan nilai `ASCENDING` dan `DESCENDING`).

Criteria : LessThan

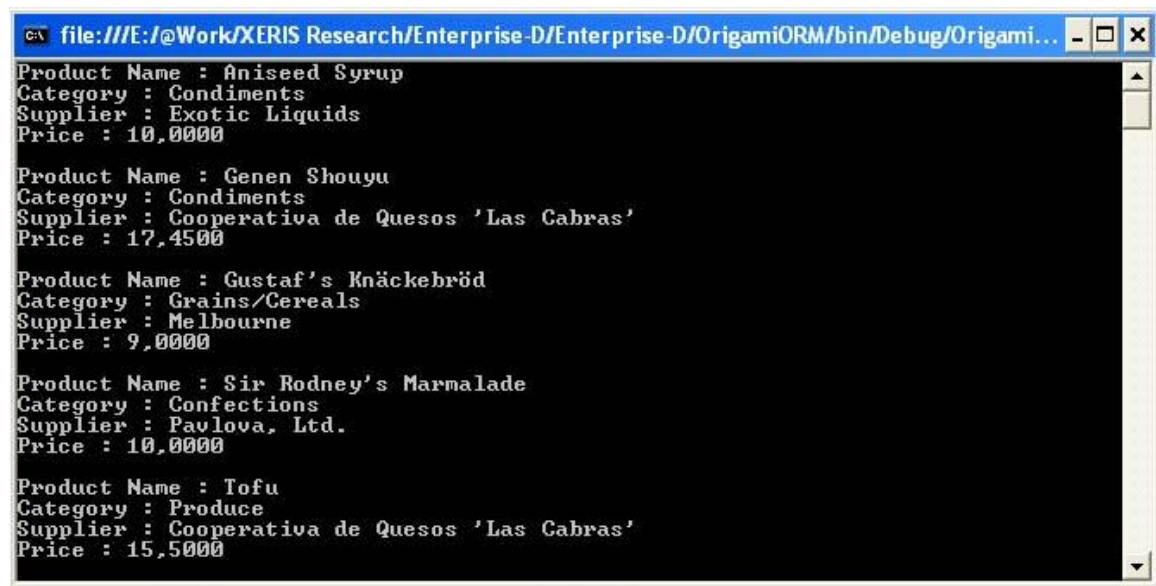
```
ObjectQuery q = em.CreateQuery<Product>();
q.AddCriteria(Criteria.LessThan("UnitPrice", 20));
q.AddOrder("ProductName", OrderBy.ASCENDING);

List<Product> products = q.List<Product>();

foreach (Product p in products)
{
    Console.WriteLine("Product Name : " + p.ProductName);
    Console.WriteLine("Category : " + p.Category.CategoryName);
    Console.WriteLine("Supplier : " + p.Supplier.CompanyName);
    Console.WriteLine("Price: " + p.UnitPrice);
    Console.WriteLine();
}
```

Query diatas menghasilkan data produk dengan `UnitPrice < 20` dan di sorting berdasarkan `ProductName`. Detail query yang degenerate seperti ini

```
... WHERE UnitPrice < 20 ORDER BY ProductName ASC
```

A screenshot of a Windows console window. The title bar reads "file:///E:/@Work/XERIS Research/Enterprise-D/Enterprise-D/OrigamiORM/bin/Debug/Origami...". The console output lists five products with their details: Product Name, Category, Supplier, and Price. The products are Aniseed Syrup, Genen Shouyu, Gustaf's Knäckebröd, Sir Rodney's Marmalade, and Tofu.

```
Product Name : Aniseed Syrup
Category : Condiments
Supplier : Exotic Liquids
Price : 10,0000

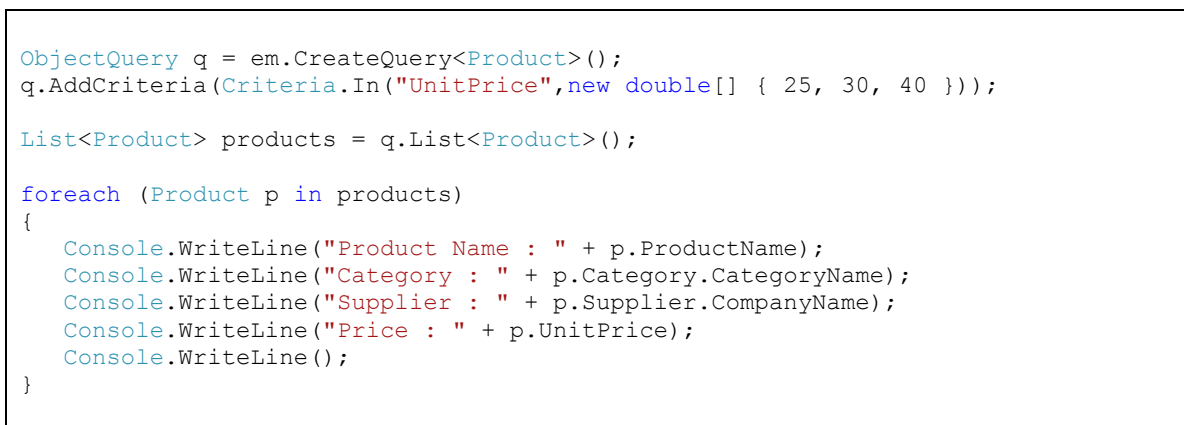
Product Name : Genen Shouyu
Category : Condiments
Supplier : Cooperativa de Quesos 'Las Cabras'
Price : 17,4500

Product Name : Gustaf's Knäckebröd
Category : Grains/Cereals
Supplier : Melbourne
Price : 9,0000

Product Name : Sir Rodney's Marmalade
Category : Confections
Supplier : Pavlova, Ltd.
Price : 10,0000

Product Name : Tofu
Category : Produce
Supplier : Cooperativa de Quesos 'Las Cabras'
Price : 15,5000
```

Criteria : In

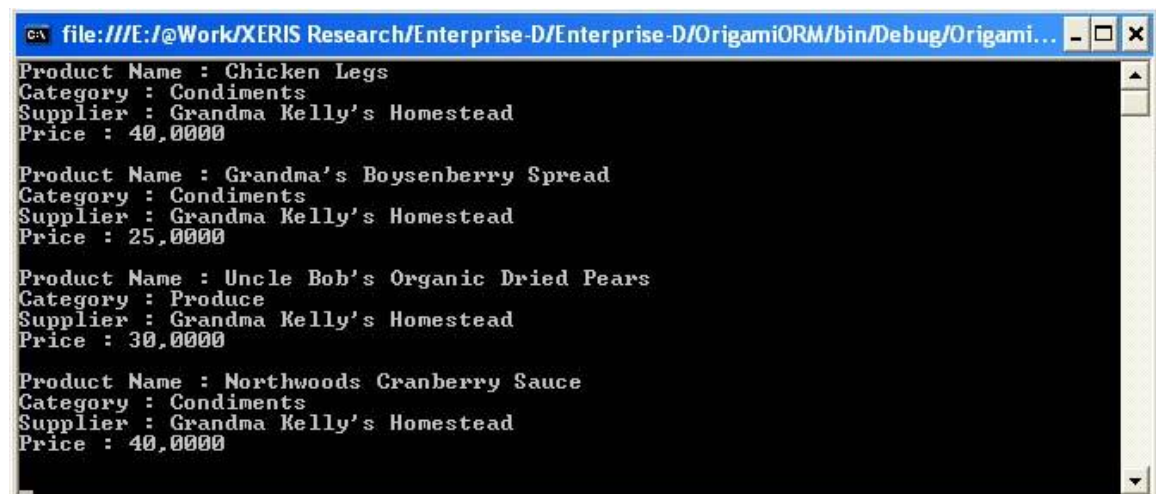
A screenshot of a code editor showing C# code that uses Entity Framework's ObjectQuery to query products based on unit price. The code includes comments in Spanish and uses LINQ to filter products where the unit price is in the range of 25 to 40.

```
ObjectQuery q = em.CreateQuery<Product>();
q.AddCriteria(Criteria.In("UnitPrice", new double[] { 25, 30, 40 }));

List<Product> products = q.List<Product>();

foreach (Product p in products)
{
    Console.WriteLine("Product Name : " + p.ProductName);
    Console.WriteLine("Category : " + p.Category.CategoryName);
    Console.WriteLine("Supplier : " + p.Supplier.CompanyName);
    Console.WriteLine("Price : " + p.UnitPrice);
    Console.WriteLine();
}
```

Generate Query : ... WHERE UnitPrice IN (25,30,40)

A screenshot of a Windows console window showing the results of a query. The title bar is the same as the first screenshot. The console output lists four products that match the criteria: Chicken Legs, Grandma's Boysenberry Spread, Uncle Bob's Organic Dried Pears, and Northwoods Cranberry Sauce.

```
Product Name : Chicken Legs
Category : Condiments
Supplier : Grandma Kelly's Homestead
Price : 40,0000

Product Name : Grandma's Boysenberry Spread
Category : Condiments
Supplier : Grandma Kelly's Homestead
Price : 25,0000

Product Name : Uncle Bob's Organic Dried Pears
Category : Produce
Supplier : Grandma Kelly's Homestead
Price : 30,0000


Product Name : Northwoods Cranberry Sauce
Category : Condiments
Supplier : Grandma Kelly's Homestead
Price : 40,0000
```

Criteria : Between

```
ObjectQuery q = em.CreateQuery<Product>();
q.AddCriteria(Criteria.Between("UnitPrice", 18, 40));

List<Product> products = q.List<Product>();
foreach (Product p in products)
{
    Console.WriteLine("Product Name : " + p.ProductName);
    Console.WriteLine("Category : " + p.Category.CategoryName);
    Console.WriteLine("Supplier : " + p.Supplier.CompanyName);
    Console.WriteLine("Price : " + p.UnitPrice);
    Console.WriteLine();
}
```

Generate Query : ... WHERE UnitPrice BETWEEN 18 AND 40



```
file:///E:/@Work/XERIS Research/Enterprise-D/Enterprise-D/OrigamiORM/bin/Debug/Origami...
Product Name : Chicken Legs
Category : Condiments
Supplier : Grandma Kelly's Homestead
Price : 40,0000

Product Name : Chef Anton's Cajun Seasoning
Category : Condiments
Supplier : New Orleans Cajun Delights
Price : 22,0000

Product Name : Chef Anton's Gumbo Mix
Category : Condiments
Supplier : New Orleans Cajun Delights
Price : 21,0000

Product Name : Grandma's Boysenberry Spread
Category : Condiments
Supplier : Grandma Kelly's Homestead
Price : 25,0000

Product Name : Uncle Bob's Organic Dried Pears
Category : Produce
Supplier : Grandma Kelly's Homestead
Price : 30,0000
```

Multiple Category

```
ObjectQuery q = em.CreateQuery<Product>();

q.AddCriteria(Criteria.Between("UnitPrice", 18, 40));
q.AddCriteria(Criteria.Equal("CategoryName", "Confections"));
q.AddOrder("UnitPrice", OrderBy.DESENDING);

List<Product> products = q.List<Product>();

foreach (Product p in products)
{
    Console.WriteLine("Product Name : " + p.ProductName);
    Console.WriteLine("Category ID : " + p.Category.CategoryID);
    Console.WriteLine("Supplier : " + p.Supplier.CompanyName);
    Console.WriteLine("Price : " + p.UnitPrice);
    Console.WriteLine();
}
```

A screenshot of a Windows console window. The title bar reads "file:///E:/@Work/XERIS Research/Enterprise-D/Enterprise-D/OrigamiORM/bin/Debug/Origami...". The console output displays three product records. Each record consists of four lines: "Product Name", "Category ID", "Supplier", and "Price". The first record is for "Pavlova" with Category ID 3, Supplier "Mayumi's", and Price 39,0000. The second record is for "NuNuCa Nuß-Nougat-Creme" with Category ID 3, Supplier "PB Knäckebröd AB", and Price 31,2300. The third record is for "Sir Rodney's Scones" with Category ID 3, Supplier "Pavlova, Ltd.", and Price 21,0000.

```
Product Name : Pavlova
Category ID : 3
Supplier : Mayumi's
Price : 39,0000

Product Name : NuNuCa Nuß-Nougat-Creme
Category ID : 3
Supplier : PB Knäckebröd AB
Price : 31,2300

Product Name : Sir Rodney's Scones
Category ID : 3
Supplier : Pavlova, Ltd.
Price : 21,0000
```

6.9.2 Native Query

Selain menggunakan criteria query, Origami mendukung juga penggunaan native SQL DBMS bersangkutan. Native query dapat berisi perintah sql SELECT, INSERT, UPDATE dan DELETE baik dengan ataupun tanpa parameter.

```
ObjectQuery q = em.CreateNativeQuery("SELECT * FROM Customers WHERE "
    + "CustomerID=@CustomerID");

q.SetParameter("@CustomerID", "ALFKI");

List<Customer> customers = q.List<Customer>();
foreach (Customer cust in customers)
{
    Console.WriteLine(cust.CompanyName);
    Console.WriteLine(cust.Address);
}
```

Method `SetParameter()` memiliki 2 parameter yaitu : param dan value. Param adalah parameter SQL seperti `@CustomerID` dan value adalah nilai dari parameter tersebut. Tipe data Value bisa string, integer, double atau tipe data lainnya. Jika SQL tidak memiliki parameter method `SetParameter()` tidak usah dipanggil.

Contoh lain jika parameter SQL nya lebih dari satu adalah sebagai berikut :

```
ObjectQuery q = em.CreateNativeQuery("INSERT INTO Customers"
    + " (CustomerID,CompanyName,Address) "
    + "VALUES (@CustomerID,@CompanyName,@Address)");

q.SetParameter("@CustomerID", "XERIS");
q.SetParameter("@CompanyName", "Xeris ");
q.SetParameter("@Address", "Sentul Valley");

q.Execute();
```

SQL Update :

```
ObjectQuery q = em.CreateNativeQuery("UPDATE Customers SET "
    + "CompanyName=@CompanyName, "
    + "Address=@Address "
    + "WHERE CustomerID=@CustomerID");

q.SetParameter("@CustomerID", "XERIS");
q.SetParameter("@CompanyName", "Xeris System Interface");
q.SetParameter("@Address", "Bukit Sentul");

q.Execute();
```

Method `Execute()` digunakan untuk SQL Insert, Update dan Delete

Class Customer nya sebagai berikut :

```
using System;
using Origami.Data.Mapping;


namespace OrigamiORM.Entity
{
    [Table(Name = "Customers")]
    public class Customer
    {
        private string customerId;
        private string companyName;
        private string contactName;

        [Id(Name = "CustomerID")]
        public string CustomerID
        {
            get { return customerId; }
            set { customerId = value; }
        }

        [Column(Name = "CompanyName")]
        public string CompanyName
        {
            get { return companyName; }
            set { companyName = value; }
        }

        [Column(Name = "ContactName")]
        public string ContactName
        {
            get { return contactName; }
            set { contactName = value; }
        }
    }
}
```

Tabel Customers

Table - dbo.Customers		Summary	
	Column Name	Data Type	Allow Nulls
	CustomerID	varchar(5)	<input type="checkbox"/>
	CompanyName	varchar(40)	<input checked="" type="checkbox"/>
	ContactName	varchar(30)	<input checked="" type="checkbox"/>
	Address	varchar(60)	<input checked="" type="checkbox"/>
	City	varchar(15)	<input checked="" type="checkbox"/>
	Country	varchar(15)	<input checked="" type="checkbox"/>
	Phone	varchar(24)	<input checked="" type="checkbox"/>

6.9.3 Named Native Query

Jika penulisan perintah sql di program utama dinilai “mengotori” kode, perintah-perintah sql dapat ditulis sekaligus di entity class bersangkutan. Sebelumnya harus disertai attribute yang sesuai. Attribute untuk named native query adalah sebagai berikut :

Attribute	Keterangan
SQLSelect	<p>Attribute untuk sql SELECT</p> <p><code>[SQLSelect (Name, Sql)]</code></p> <p>Contoh : <code>[SQLSelect (Name="SelectAll", CommandText="SELECT * FROM Suppliers")]</code></p> <p>Callable = set menjadi true jika menggunakan stored procedure Name = nama query yang akan dipanggil diprogram Sql = perintah sql</p>
SQLInsert	<p>Attribute untuk sql INSERT</p> <p><code>[SQLInsert (Name, Sql)]</code></p> <p>Contoh <code>[SQLInsert (Name="Insert", CommandText=" INSERT INTO Suppliers (CompanyName,Address) VALUES (@CompanyName,@Address) ")]</code></p>
SQLUpdate	<p>Attribute untuk sql UPDATE</p> <p><code>[SQLUpdate (Name, Sql)]</code></p>

	<p>Contoh :</p> <pre>[SQLUpdate(Name="Update", CommandText="UPDATE Suppliers SET CompanyName=@ CompanyName,Address=@Address WHERE SupplierID=@SupplierID")]</pre>
SQLDelete	<p>Attribute untuk sql DELETE</p> <pre>[SQLDelete(Name,Sql)]</pre> <p>Contoh :</p> <pre>[SQLDelete(Name="Delete", CommandText="DELETE FROM Suppliers WHERE SupplierID=@SupplierID")]</pre>

Berikut contohnya :

```
using System;
using Origami.Data.Mapping;

namespace OrigamiORM.Entity
{
    [SQLSelect(Name="selectAll",CommandText="SELECT * FROM Suppliers")]
    [SQLSelect(Name = "SelectByID", CommandText = "SELECT * FROM Suppliers "
        + "WHERE SupplierID=@SupplierID")]

    [SQLSelect(Name = "Find", CommandText = "SELECT * FROM Suppliers "
        + "WHERE CompanyName LIKE @CompanyName")]

    [SQLInsert(Name = "Insert", CommandText="INSERT INTO Suppliers "
        + "(CompanyName,Address) VALUES (@CompanyName,@Address)")]

    [SQLUpdate(Name = "Update", CommandText = "UPDATE Suppliers SET "
        + "CompanyName=@CompanyName,Address=@Address "
        + "WHERE SupplierID=@SupplierID")]

    [SQLDelete(Name = "Delete", CommandText = "DELETE FROM Suppliers "
        + "WHERE SupplierID=@SupplierID")]

    [Table(Name = "Suppliers")]
    public class Supplier
    {
        private int supplierId;
        private string companyName;
        private string contactName;
        private string address;
        private string city;
        private string country;
        private string phone;

        //property dan mapping
    }
}
```

Sedangkan cara memanggil named query di entity class :

SELECT

```
ObjectQuery q = em.GetNamedQuery<Supplier>(SqlType.SELECT, "SelectAll");  
List<Supplier> suppliers = q.List<Supplier>();  
foreach (Supplier s in suppliers)  
{  
    Console.WriteLine(s.CompanyName);  
    Console.WriteLine(s.Address);  
}
```

```
ObjectQuery q = em.GetNamedQuery<Supplier>(SqlType.SELECT, "SelectByID");  
q.SetParameter("@SupplierID", 1);  
List<Supplier> suppliers = q.List<Supplier>();  
foreach (Supplier s in suppliers)  
{  
    Console.WriteLine(s.CompanyName);  
    Console.WriteLine(s.Address);  
}
```

Output



```
file:///E:/@Work/XERIS Research/Enterprise-D/Enterprise-D/OrigamiORM/bin/Debug/Origami...  
Exotic Liquids  
49 Gilbert St.  
New Orleans Cajun Delights  
P.O. Box 78934  
Grandma Kelly's Homestead  
707 Oxford Rd.  
Tokyo Traders  
'9-8 Sekimai  
Tokyo  
Japan  
Cooperativa de Quesos 'Las Cabras'  
Calle del Rosal 4  
Mayumi's  
'92 Setsuko  
Paulova, Ltd.  
'74 Rose St.  
Melbourne  
Australia
```



```
file:///E:/@Work/XERIS Research/Enterprise-D/Enterprise-D/OrigamiORM/bin/Debug/Origami...  
Exotic Liquids  
49 Gilbert St.
```

INSERT

```
ObjectQuery q = em.GetNamedQuery<Supplier>(SqlType.INSERT, "Insert");
q.SetParameter("@CompanyName", "Microsoft");
q.SetParameter("@Address", "Seattle");

q.Execute();
```

UPDATE

```
ObjectQuery q = em.GetNamedQuery<Supplier>(SqlType.UPDATE, "Update");
q.SetParameter("@SupplierID", 39);
q.SetParameter("@CompanyName", "Microsoft Corp");
q.SetParameter("@Address", "Seattle, USA");

q.Execute();
```

DELETE

```
ObjectQuery q = em.GetNamedQuery<Supplier>(SqlType.DELETE, "Delete");
q.SetParameter("@SupplierID", 39);
q.Execute();
```

6.9.4 Aggregate Query

Origami mendukung perintah Aggregate SQL MAX,MIN,SUM,AVG,STDDEV dan COUNT.

Aggregate	Keterangan
MAX	Mengambil nilai terbesar dari kumpulan data q.AddAggregate(Aggregate.Max("UnitPrice"))
MIN	Mengambil nilai terkecil dari kumpulan data q.AddAggregate(Aggregate.Min("UnitPrice"))
SUM	Menghitung total nilai q.AddAggregate(Aggregate.Sum("UnitPrice"))
AVERAGE	Menghitung rata-rata dari kumpulan data q.AddAggregate(Aggregate.Average("UnitPrice"))
STDDEV	Menghitung standar deviasi dari kumpulan data q.AddAggregate(Aggregate.StdDev("UnitPrice"))

COUNT	Menghitung banyaknya data <pre>q.AddAggregate(Aggregate.Count("UnitPrice"))</pre>
-------	--

Pernyataan aggregate query bisa dikelompokkan berdasarkan group tertentu, selain itu bisa juga menggunakan parameter query untuk menyeleksi data tertentu saja. Hal ini dapat dilakukan dengan memanggil method `AddGroup()`, `AddCriteria()`, atau `AddHaving()`

Berikut contoh penggunaannya:

MAX

```
ObjectQuery q = em.CreateQuery<Product>();
q.AddAggregate(Aggregate.Max("UnitPrice"));

List<Product> products = q.List<Product>();
foreach (Product p in products)
{
    Console.WriteLine(p.UnitPrice);
}
```

MIN

```
ObjectQuery q = em.CreateQuery<Product>();
q.AddAggregate(Aggregate.Min("UnitPrice"));

List<Product> products = q.List<Product>();
foreach (Product p in products)
{
    Console.WriteLine(p.UnitPrice);
}
```

SUM berdasarkan group tertentu

```
ObjectQuery q = em.CreateQuery<Product>();

q.AddColumn("SupplierID");
q.AddAggregate(Aggregate.Sum("UnitPrice"));
q.AddGroup("SupplierID");

List<Product> products = q.List<Product>();

foreach (Product p in products)
{
    Console.WriteLine(p.UnitPrice);
}
```

Aggregate Query dengan Kriteria :

AVERAGE

```
ObjectQuery q = em.CreateQuery<Product>();

q.AddColumn("SupplierID");
q.AddAggregate(Aggregate.Average("UnitPrice"));
q.AddCriteria(Criteria.Equal("CompanyName", "Tokyo Traders"));
q.AddGroup("SupplierID");

List<Product> products = q.List<Product>();
foreach (Product p in products)
{
    Console.WriteLine(p.UnitPrice);
}
```

HAVING

```
ObjectQuery q = em.CreateQuery<Product>();

q.AddColumn("SupplierID");
q.AddAggregate(Aggregate.Max("UnitPrice"));
q.AddGroup("SupplierID");
q.AddHaving(Having.Max(Criteria.GreaterThan("UnitPrice", 38)));

List<Product> products = q.List<Product>();

foreach (Product p in products)
{
    Console.WriteLine(p.UnitPrice);
}
```

HAVING dan Order

```
ObjectQuery q = em.CreateQuery<Product>();

q.AddColumn("SupplierID");
q.AddAggregate("UnitPrice", Expression.MAX);
q.AddGroup("SupplierID");
q.AddHaving(Having.Max(Criteria.GreaterThan("UnitPrice", 38)));
q.AddOrder("SupplierID", OrderBy.DESCENDING);

List<Product> products = q.List<Product>();
foreach (Product p in products)
{
    Console.WriteLine(p.UnitPrice);
}
```

6.10 Stored Procedure

Kebanyakan DBMS bertipe *client server* memiliki fitur stored procedure. Dengan menggunakan stored procedure trafik jaringan akibat permintaan query ke database server dapat dikurangi. Selain itu dari segi security dapat mengatasi masalah SQL injection. Maintenance juga dapat dilakukan dengan mudah karena business process yang mudah berubah ditulis di stored procedure bukan pada kode program.

Untuk mengaktifkan stored procedure set parameter attribute `Callable` SQL bersangkutan menjadi true, parameter `Name` diisi nama stored procedure yang akan dipanggil di program, dan parameter `Sql` nya diisi nama stored procedure di database server.

Berikut contohnya :

```
using System;
using Origami.Data.Mapping;

namespace OrigamiORM.Entity
{
    [SQLSelect(IsStoredProcedure=true, Name="SelectByID", CommandText="spCustomerByID" ) ]
    [SQLSelect(IsStoredProcedure=true, Name="FindByName", CommandText="spFindCustomer" ) ]

    [SQLInsert(IsStoredProcedure=true, Name="Insert", CommandText="spInsertCustomer" ) ]
    [SQLUpdate(IsStoredProcedure=true, Name="Update", CommandText="spUpdateCustomer" ) ]
    [SQLDelete(IsStoredProcedure=true, Name="Delete", CommandText="spDeleteCustomer" ) ]

    [Table(Name = "Customers" ) ]
    public class Customer
    {
        private string customerId;
        private string companyName;
        private string contactName;

        //property dan mapping
    }
}
```

Stored procedure yang bias dipanggil meliputi perintah-perintah SQL : SELECT, INSERT, UPDATE, dan DELETE. Dalam satu entity class bisa dipanggil banyak perintah

Stored Procedure :

```
CREATE PROCEDURE [dbo].[spCustomerByID]
    @CustomerID int
AS
BEGIN
    SELECT * FROM Customers WHERE CustomerID=@CustomerID
END
```

```

CREATE PROCEDURE [dbo].[spFindCustomer]
    @CompanyName int
AS
BEGIN
    SELECT * FROM Customers WHERE CompanyName LIKE @CompanyName
END

CREATE PROCEDURE [dbo].[spInsertCustomer]
    @CompanyName varchar,
    @Address varchar
AS
BEGIN
    INSERT INTO Customers (CompanyName,Address) VALUES
        (@CompanyName,@Address)
END

CREATE PROCEDURE [dbo].[spUpdateCustomer]
    @CustomerID int,
    @CompanyName varchar,
    @Address varchar
AS
BEGIN
    INSERT INTO Customers (CompanyName,Address) VALUES
        (@CompanyName,@Address) WHERE CustomerID=@CustomerID
END

CREATE PROCEDURE [dbo].[spDeleteCustomer]
    @CustomerID int,
AS
BEGIN
    DELETE FROM Customers WHERE CustomerID=@CustomerID
END

```

INSERT

```

ObjectQuery qry = em.GetNativeQuery<Category>(SqlType.INSERT, "Insert");
qry.SetParameter("@CustomerName", "XERIS");
qry.SetParameter("@Address", "Bogor");

qry.Execute();

```

UPDATE

```

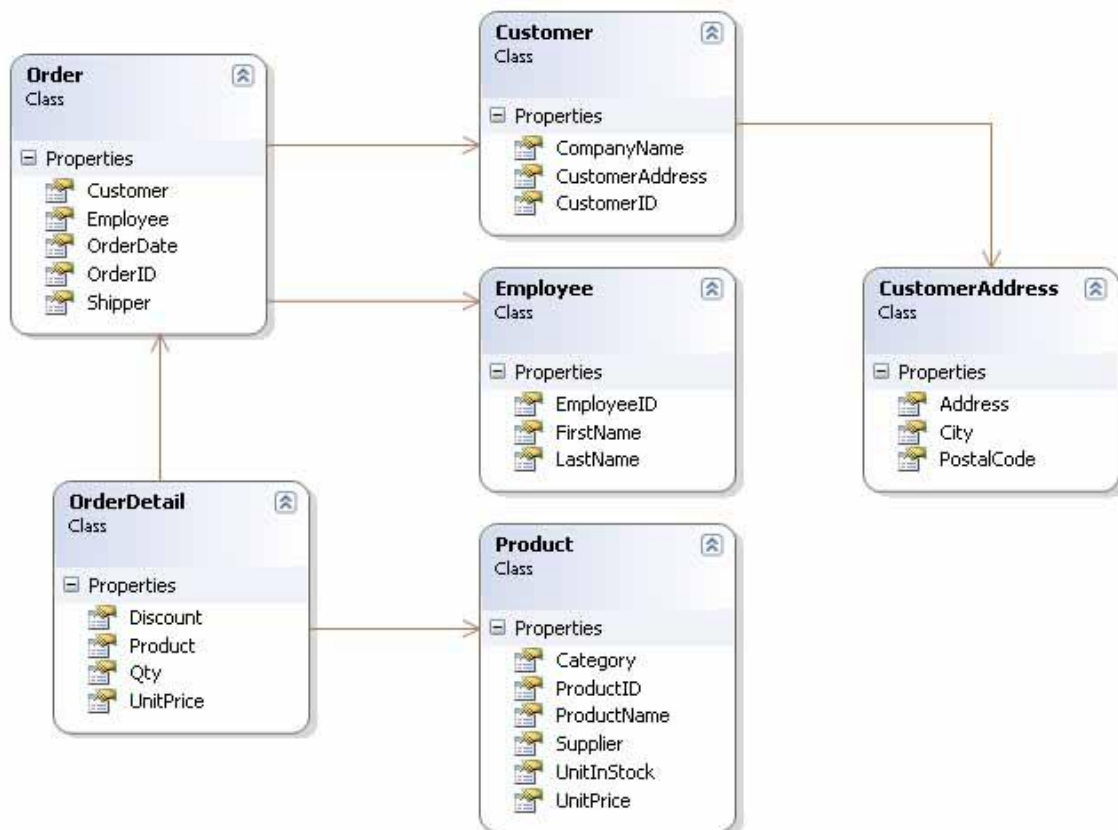
ObjectQuery qry = em.GetNativeQuery<Category>(SqlType.UPDATE, "Update");
qry.SetParameter("@CompanyName", "System Interface");
qry.SetParameter("@Address", "Jakarta");
qry.Execute();

```

DELETE

```
ObjectQuery qry = em.GetNativeQuery<Category>(SqlType.DELETE, "Delete");  
qry.SetParameter("@CustomerID", 1);  
  
qry.Execute();
```


6.11 Relationship



Pada aplikasi yang berhubungan dengan basis data, kita tidak hanya berurusan dengan satu class/entitas saja, melainkan berurusan dengan beberapa entitas yang saling berelasi. Tidak seperti relasi pada *relational database* yang lebih sederhana, pada pendekatan *object oriented* ada beberapa jenis relasi yang dikenal yaitu : asosiasi, agregasi, generalisasi, dan realisasi.

Origami mendukung relasi tipe asosiasi *one to one*, *one to many*, dan *many to one*. Untuk lebih jelasnya perhatikan penerjemahan class diagram berikut :

Tabel Orders

Table - dbo.Orders		Summary	
	Column Name	Data Type	Allow Nulls
	OrderID	int	<input type="checkbox"/>
	CustomerID	varchar(5)	<input checked="" type="checkbox"/>
	EmployeeID	int	<input checked="" type="checkbox"/>
	OrderDate	datetime	<input checked="" type="checkbox"/>
	ShippedDate	datetime	<input checked="" type="checkbox"/>
	ShipName	varchar(40)	<input checked="" type="checkbox"/>
	ShipAddress	varchar(60)	<input checked="" type="checkbox"/>

Class Order

```
using System;
using System.Collections.Generic;
using Origami.Data.Mapping;

namespace OrigamiORM.Entity
{
    [Table(Name = "Orders")]
    public class Order
    {
        private int orderID;
        private int customerID;
        private Customer customer;
        private int employeeID;
        private Employee employee;
        private DateTime orderDate;
        private string shipName;
        private string shipAddress;
        private List<OrderDetail> orderDetails;

        [Id(Name = "OrderID", IsIdentity=true)]
        public int OrderID
        {
            get { return orderID; }
            set { orderID = value; }
        }

        [Column(Name="CustomerID")]
        public int CustomerID
        {
            get { return customerID; }
            set { customerID = value; }
        }
    }
}
```

```

    }

    [Association(EntityRef=typeof(Customer),Key="CustomerID")]
    public Customer Customer
    {
        get { return customer; }
        set { customer = value; }
    }

    [Column(Name = "EmployeeID")]
    public int EmployeeID
    {
        get { return employeeID; }
        set { employeeID = value; }
    }

    [Association(EntityRef=typeof(Employee),Key="EmployeeID")]
    public Employee Employee
    {
        get { return employee; }
        set { employee = value; }
    }

    [Column(Name = "OrderDate")]
    public DateTime OrderDate
    {
        get { return orderDate; }
        set { orderDate = value; }
    }

    [Column(Name = "ShipName")]
    public string ShipName
    {
        get { return shipName; }
        set { shipName = value; }
    }



    [Column(Name = "ShipAddress")]
    public string ShipAddress
    {
        get { return shipAddress; }
        set { shipAddress = value; }
    }

    public List<OrderDetail> OrderDetails
    {
        get { return orderDetails; }
        set { orderDetails = value; }
    }

    }
}

```

Tabel OrderDetails

Table - dbo.OrderDetails		Summary	
	Column Name	Data Type	Allow Nulls
	OrderID	int	<input type="checkbox"/>
	ProductID	int	<input type="checkbox"/>
	UnitPrice	money	<input checked="" type="checkbox"/>
	Quantity	int	<input checked="" type="checkbox"/>
	Discount	int	<input checked="" type="checkbox"/>

Class OrderDetail

```
using System;
using Origami.Data.Mapping;

namespace OrigamiORM.Entity
{
    [Table(Name = "OrderDetails")]
    public class OrderDetail
    {
        private int orderID;
        private int productID;
        private Product product;
        private int qty;
        private decimal unitPrice;
        private double discount;

        [Column(Name = "OrderID")]
        public int OrderID
        {
            get { return orderID; }
            set { orderID = value; }
        }

        [Column(Name = "ProductID")]
        public int ProductID
        {
            get { return productID; }
            set { productID = value; }
        }

        [Association(EntityRef=typeof(Product), Key="ProductID")]
        public Product Product
        {
            get { return product; }
            set { product = value; }
        }

        [Column(Name = "Quantity")]
        public int Qty
        {
            get { return qty; }
            set { qty = value; }
        }
    }
}
```

```

        [Column(Name = "UnitPrice")]
        public decimal UnitPrice
        {
            get { return unitPrice; }
            set { unitPrice = value; }
        }

        [Column(Name = "Discount")]
        public double Discount
        {
            get { return discount; }
            set { discount = value; }
        }
    }
}

```

Class Order memiliki hubungan *one to many* dengan class OrderDetail, karena itu di class Order atribut orderDetails disimpan pada List (`List<OrderDetail> orderDetails`), ini artinya satu order ada banyak order detail

Class Customer

```

using System;
using Origami.Data.Mapping;

namespace OrigamiORM.Entity
{
    [Table(Name = "Customers")]
    public class Customer
    {
        private string customerId;
        private string companyName;
        private string contactName;
        private CustomerAddress customerAddress;

        [Id(Name = "CustomerID")]
        public string CustomerID
        {
            get { return customerId; }
            set { customerId = value; }
        }

        [Column(Name = "CompanyName")]
        public string CompanyName
        {
            get { return companyName; }
            set { companyName = value; }
        }

        [Column(Name = "ContactName")]
        public string ContactName
        {
            get { return contactName; }
            set { contactName = value; }
        }

        [Association(EntityRef=typeof(CustomerAddress), Key="CustomerID")]
    }
}

```

```

        public CustomerAddress CustomerAddress
        {
            get { return customerAddress; }
            set { customerAddress = value; }
        }
    }
}

```

Tabel CustomerAddress

Table - dbo.CustomerAddress			Summary
Column Name	Data Type	Allow Nulls	
CustomerID	varchar(5)	<input checked="" type="checkbox"/>	
Address	varchar(50)	<input checked="" type="checkbox"/>	
City	varchar(50)	<input checked="" type="checkbox"/>	
PostalCode	varchar(50)	<input checked="" type="checkbox"/>	
Country	varchar(50)	<input checked="" type="checkbox"/>	

Class CustomerAddress

```

using System;
using Origami.Data.Mapping;

namespace OrigamiORM.Entity
{
    [Table(Name="CustomerAddress")]
    public class CustomerAddress
    {
        private string customerID;
        private string address;
        private string city;
        private string country;
        private string phone;

        [Column(Name = "CustomerID")]
        public string CustomerID
        {
            get { return customerID; }
            set { customerID = value; }
        }

        [Column(Name = "Address")]
        public string Address
        {
            get { return address; }
            set { address = value; }
        }

        [Column(Name = "City")]
        public string City
        {

```

```

        get { return city; }
        set { city = value; }
    }


    [Column(Name = "Country")]
    public string Country
    {
        get { return country; }
        set { country = value; }
    }

    [Column(Name = "Phone")]
    public string Phone
    {
        get { return phone; }
        set { phone = value; }
    }
}

```

Class Customer memiliki hubungan *one to one* dengan class CustomerAddress, ini artinya satu customer hanya memiliki satu detail alamat.

Tabel Employees

Table - dbo.Employees		Summary	
	Column Name	Data Type	Allow Nulls
	EmployeeID	int	<input type="checkbox"/>
	LastName	varchar(20)	<input checked="" type="checkbox"/>
	FirstName	varchar(10)	<input checked="" type="checkbox"/>
	Title	varchar(30)	<input checked="" type="checkbox"/>
	BirthDate	datetime	<input checked="" type="checkbox"/>
	HireDate	datetime	<input checked="" type="checkbox"/>

Class Employee

```

using System;
using Origami.Data.Mapping;

namespace OrigamiORM.Entity
{
    [Table(Name = "Employees")]
    public class Employee
    {
        private int employeeID;
        private string lastName;
        private string firstName;
        private string title;
        private DateTime birthDate;
    }
}

```

```

        private DateTime hireDate;

        [Id(Name = "EmployeeID", IsIdentity=true)]
        public int EmployeeID
        {
            get { return employeeID; }
            set { employeeID = value; }
        }

        [Column(Name = "LastName")]
        public string LastName
        {
            get { return lastName; }
            set { lastName = value; }
        }

        [Column(Name = "FirstName")]
        public string FirstName
        {
            get { return firstName; }
            set { firstName = value; }
        }

        [Column(Name = "Title")]
        public string Title
        {
            get { return title; }
            set { title = value; }
        }

        [Column(Name = "BirthDate")]
        public DateTime BirthDate
        {
            get { return birthDate; }
            set { birthDate = value; }
        }

        [Column(Name = "HireDate")]
        public DateTime HireDate
        {
            get { return hireDate; }
            set { hireDate = value; }
        }
    }
}

```

Class Product

```

using System;
using Origami.Data.Mapping;

namespace OrigamiORM.Entity
{
    [Table(Name = "Products")]
    public class Product
    {
        private int productID;
        private string productName;
        private int categoryID;
    }
}

```

```

private Category category;
private int supplierID;
private Supplier supplier;
private decimal unitPrice;
private int unitInStock;

[Id(Name = "ProductID",IsIdentity=true)]
public int ProductID
{
    get { return productID; }
    set { productID = value; }
}

[Column(Name = "ProductName")]
public string ProductName
{
    get { return productName; }
    set { productName = value; }
}

[Column(Name="CategoryID")]
public int CategoryID
{
    get { return categoryID; }
    set { categoryID = value; }
}

[Association(EntityRef = typeof(Category),Key="CategoryID")]
public Category Category
{
    get { return category; }
    set { category = value; }
}

[Column(Name = "SupplierID")]
public int SupplierID
{
    get { return supplierID; }
    set { supplierID = value; }
}

[Association(EntityRef = typeof(Supplier),Key="SupplierID")]
public Supplier Supplier
{
    get { return supplier; }
    set { supplier = value; }
}

[Column(Name = "UnitPrice")]
public decimal UnitPrice
{
    get { return unitPrice; }
    set { unitPrice = value; }
}

[Column(Name = "UnitsInStock")]
public int UnitInStock
{
    get { return unitInStock; }
    set { unitInStock = value; }
}
}

```


Class Category

```
using System;
using Origami.Data.Mapping;

namespace OrigamiORM.Entity
{
    [Table(Name="Categories")]
    public class Category
    {
        private int categoryID;
        private string categoryName;
        private string description;

        [Id(Name="CategoryId", IsIdentity=true)]
        public int CategoryID
        {
            get { return categoryID; }
            set { categoryID = value; }
        }

        [Column(Name="CategoryName")]
        public string CategoryName
        {
            get { return categoryName; }
            set { categoryName = value; }
        }

        [Column(Name="Description")]
        public string Description
        {
            get { return description; }
            set { description = value; }
        }
    }
}
```

6.11.1 One to One

```
List<Customer> customers = em.Get<Customer>();

foreach (Customer cust in customers)
{
    Console.WriteLine("Customer Name : " + cust.CompanyName);
    Console.WriteLine("Address      : " + cust.CustomerAddress.Address);
    Console.WriteLine("City        : " + cust.CustomerAddress.City);
    Console.WriteLine("Country    : " + cust.CustomerAddress.Country);
    Console.WriteLine();
}
```

Output



```
file:///E:/@Work/XERIS Research/Enterprise-D/Enterprise-D/OrigamiORM/bin/Debug/Origami...
Customer Name : Ana Trujillo Emparedados y helados
Address       : Avda. de la Constitución 2222
City          : México D.F.
Postal Code   : 67000
Country       : Mexico

Customer Name : Around the Horn
Address       : 120 Hanover Sq.
City          : London
Postal Code   : W01 1DP
Country       : UK

Customer Name : Berglunds snabbköp
Address       : Berguvsvägen 8
City          : Luleå
Postal Code   : S-958 22
Country       : Sweden
```

6.11.2 Many to One

```
List<Order> orders = em.Get<Order>();

foreach (Order o in orders)
{
    Console.WriteLine("Order ID      : " + o.OrderID);
    Console.WriteLine("Order Date   : " + o.OrderDate);
    Console.WriteLine("Customer   : " + o.Customer.CompanyName);
    Console.WriteLine("Employee   : " + o.Employee.FirstName);
}
```

Output



```
file:///E:/@Work/XERIS Research/Enterprise-D/Enterprise-D/OrigamiORM/bin/Debug/Origami...
Order Date    : 13/05/1996 0:00:00
Customer      : Rancho grande
Employee      : Michael

Order ID      : 11020
Order Date    : 14/05/1996 0:00:00
Customer      : Ottilies Käseladen
Employee      : Andrew

Order ID      : 11021
Order Date    : 14/05/1996 0:00:00
Customer      : QUICK-Stop
Employee      : Janet

Order ID      : 11022
Order Date    : 14/05/1996 0:00:00
Customer      : Hanari Carnes
Employee      : Anne
```

6.11.3 One to many

Produk per kategori

```
List<Category> categories = em.Get<Category>();

foreach (Category cat in categories)
{
    Console.WriteLine("Category Name : " + cat.CategoryName);
    ObjectQuery q = em.CreateQuery<Product>();
    q.AddCriteria(Criteria.Equal("CategoryID", cat.CategoryID));
    cat.Products = q.List<Product>();
    Console.WriteLine();

    foreach (Product p in cat.Products)
    {
        Console.WriteLine("    Product Name : " + p.ProductName);
        Console.WriteLine("    Supplier      : " + p.Supplier.CompanyName);
        Console.WriteLine();
    }
}
```

Output



```
Category Name : Condiments
Product Name : Chicken Legs
Supplier      : Grandma Kelly's Homestead
Product Name : Aniseed Syrup
Supplier      : Exotic Liquids
Product Name : Chef Anton's Cajun Seasoning
Supplier      : New Orleans Cajun Delights
Product Name : Chef Anton's Gumbo Mix
Supplier      : New Orleans Cajun Delights
Product Name : Grandma's Boysenberry Spread
Supplier      : Grandma Kelly's Homestead
Product Name : Northwoods Cranberry Sauce
Supplier      : Grandma Kelly's Homestead
```

Order dan order detail

```
List<Order> orders = em.Get<Order>();

foreach (Order o in orders)
{
    Console.WriteLine("Order ID      : " + o.OrderID);
    Console.WriteLine("Order Date    : " + o.OrderDate);
    ObjectQuery q = em.CreateQuery<OrderDetail>();
    q.AddCriteria(Criteria.Equal("OrderID", o.OrderID));

    o.OrderDetails = q.List<OrderDetail>();
}
```

```

foreach (OrderDetail od in o.OrderDetails)
{
    Console.WriteLine(" Product      : " + od.Product.ProductName);
    Console.WriteLine(" Qty          : " + od.Qty);
    Console.WriteLine(" Price       : " + od.UnitPrice);
}
}

```

Output

```

file:///E:/@Work/XERIS Research/Enterprise-D/Enterprise-D/OrigamiORM/bin/Debug/Origami...
Order ID      : 11061
Order Date    : 30/05/1996 0:00:00
Product       : Camembert Pierrot
Qty           : 15
Price         : 34.0000

Order ID      : 11062
Order Date    : 30/05/1996 0:00:00
Product       : Perth Pasties
Qty           : 10
Price         : 33.0000

Product       : Outback Lager
Qty           : 12
Price         : 15.0000

```

6.12 Query pada Relationship

Pencarian berdasarkan tgl order

```

DateTime from = new DateTime(1994, 8, 4);
DateTime to = new DateTime(1994, 9, 1);

ObjectQuery q = em.CreateQuery<Order>();
q.AddCriteria(Criteria.Between("OrderDate", from.ToString("yyyy/MM/dd"),
    to.ToString("yyyy/MM/dd")));

List<Order> orders = q.List<Order>();

foreach (Order o in orders)
{
    Console.WriteLine("Order ID      : " + o.OrderID);
    Console.WriteLine("Order Date    : " + o.OrderDate);
}

```

Query diatas menghasilkan semua order dari tanggal 4/8/1994 sampai tanggal 1/9/1994. Criteria yang digunakan adalah BETWEEN

Output



```
Order ID : 10248
Order Date : 04/08/1994 0:00:00

Order ID : 10249
Order Date : 05/08/1994 0:00:00

Order ID : 10250
Order Date : 08/08/1994 0:00:00

Order ID : 10251
Order Date : 08/08/1994 0:00:00

Order ID : 10252
Order Date : 09/08/1994 0:00:00
```

Pencarian berdasarkan Employee

```
ObjectQuery q = em.CreateQuery<Order>();
q.AddCriteria(Criteria.Equal("FirstName", "Nancy"));

List<Order> orders = q.List<Order>();
foreach (Order o in orders)
{
    Console.WriteLine("Order ID : " + o.OrderID);
    Console.WriteLine("Order Date : " + o.OrderDate);
    Console.WriteLine("Employee : " + o.Employee.FirstName);
    Console.WriteLine();
}
```

Output



```
Order Date : 07/08/1995 0:00:00
Employee : Nancy

Order ID : 10598
Order Date : 14/08/1995 0:00:00
Employee : Nancy

Order ID : 10604
Order Date : 18/08/1995 0:00:00
Employee : Nancy

Order ID : 10605
Order Date : 21/08/1995 0:00:00
Employee : Nancy
```

Pencarian berdasarkan Customer

```
ObjectQuery q = em.CreateQuery<Order>();
q.AddCriteria(Criteria.Equal("CustomerID", "ISLAT"));

List<Order> orders = q.List<Order>();
foreach (Order o in orders)
{
    Console.WriteLine("Order ID : " + o.OrderID);
    Console.WriteLine("Order Date : " + o.OrderDate);
}
```

```

Console.WriteLine("Customer : " + o.Customer.CompanyName);
Console.WriteLine();
}

```

Output

```

file:///E:/@Work/XERIS Research/Enterprise-D/Enterprise-D/OrigamiORM/bin/Debug/Origami...
Order ID : 10315
Order Date : 27/10/1994 0:00:00
Customer : Island Trading

Order ID : 10318
Order Date : 01/11/1994 0:00:00
Customer : Island Trading

Order ID : 10321
Order Date : 03/11/1994 0:00:00
Customer : Island Trading

Order ID : 10473
Order Date : 13/04/1995 0:00:00
Customer : Island Trading

Order ID : 10621
Order Date : 05/09/1995 0:00:00
Customer : Island Trading

Order ID : 10674
Order Date : 19/10/1995 0:00:00
Customer : Island Trading

```

6.13 Data Binding

ADO.NET menyediakan cara yang elegan untuk melakukan data binding. Data binding adalah pengikatan data pada komponen-komponen yang bersifat “*data aware*” baik pada Win Form (GUI) maupun Web Form (ASP.NET). Komponen yang bersifat *data aware* contohnya adalah combobox, listbox dan data grid/grid view. Semua komponen tersebut memiliki property data source yang bisa diisi DataSet atau objek.

Karena query yang dihasilkan Origami return value nya adalah object collection, maka Origami dapat melakukan binding ke komponen-komponen *data aware* tersebut.

```

List<Employee> employees = em.Get<Employee>();

cbEmployee.DataSource = employees;
cbSupplier.DisplayMember = "FirstName ";

lstEmployee.DataSource = employees;
lstEmployee.DisplayMember = "FirstName";

dataGrid1.DataSource = employees;

```

Property `DataSource` dari `cbSupplier` (combo box), `lstSupplier` (list box), dan `dataGrid1` (grid view) diisi object collection **employees**. Sedangkan property `DisplayMember` diisi kolom pada

tabel Employees yang hendak ditampilkan. Data binding bisa dikenakan juga pada textbox dan binding navigator. Berikut contoh lengkapnya :

Form Employee (WinForm)

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;
using Origami.Container;
using Origami.Data.Provider;
using Origami.Data;

namespace OrigamiORM
{
    public partial class Form1 : Form
    {
        private IOrigamiContainer container;
        private IEntityManager em;
        private BindingSource bs;

        public Form1()
        {
            InitializeComponent();
            container = new OrigmiContainer();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            DataSource dataSource =
                container.GetObject<DataSource>("dataSource");

            em = ConnectionFactory.Build(dataSource);
            List<Employee> employees = em.Get<Employee>();

            bs = new BindingSource();
            bs.DataSource = employees;
            bindingNavigator1.BindingSource = bs;

            LoadData();
        }

        private void LoadData()
        {
            txtFirstName.DataBindings.Add(new Binding("Text", bs,
                "FirstName"));
            txtLastName.DataBindings.Add(new Binding("Text", bs,
                "LastName"));
            txtTitle.DataBindings.Add(new Binding("Text", bs,
                "Title"));
        }

        private void btnMoveFirst_Click(object sender, EventArgs e)
        {
            bs.MoveFirst();
        }

        private void btnMovePrev_Click(object sender, EventArgs e)
        {

```

```

        bs.MovePrevious();
    }

    private void btnMoveNext_Click(object sender, EventArgs e)
    {
        bs.MoveNext();
    }

    private void btnMoveLast_Click(object sender, EventArgs e)
    {
        bs.MoveLast();
    }
}

```

6.14 Meta Data

Ada kalanya kita memerlukan informasi mengenai metadata database. Metadata (data mengenai data) ini biasanya sangat diperlukan ketika ingin memetakan objek-objek database secara dinamis. Dengan mengetahui metadata kita bisa mendapatkan informasi mengenai kolom, tipe data dan ukuran field dari sebuah tabel secara run time.

Berikut cara penggunaannya :

```

List<MetaData> schema = em.GetMetaData<Customer>();

foreach (MetaData metadata in schema)
{
    Console.WriteLine("Column Name      : " + metadata.ColumnName);
    Console.WriteLine("Data Type       : " + metadata.DataType);
    Console.WriteLine("Size         : " + metadata.ColumnSize);
    Console.WriteLine();
}

```

Output

```

file:///E:/@Work/XERIS Research/Enterprise-D/Enterprise-D/OrigamiORM/bin/Debug/Origami...
Column Name      : CustomerID
Data Type       : System.String
Size            : 5

Column Name      : CompanyName
Data Type       : System.String
Size            : 40

Column Name      : ContactName
Data Type       : System.String
Size            : 30

Column Name      : Address
Data Type       : System.String
Size            : 60

Column Name      : City
Data Type       : System.String
Size            : 15

```


6.15 Object Serialization

Object Serialization adalah fitur tambahan untuk membuat objek yang berada dalam memori (collection), *persistent* ke file untuk kemudian diolah lebih lanjut. Dengan kata lain objek/kumpulan objek di export ke format dokumen tertentu. Export dapat dilakukan ke format CSV, Excel, HTML, atau XML. Method yang digunakan adalah `Transform()` yang terletak pada interface `IEntityManager`, dan class `Query`.

```
em.Transform<Category>(DocType.CSV, "c:\\category.csv");
em.Transform<Employee>(DocType.EXCEL, "c:\\employees.xls");
em.Transform<Customer>(DocType.HTML, "c:\\customer.htm");
em.Transform<Supplier>(DocType.XML, "c:\\supplier.xml");
```

Selain melalui interface `IEntityManager`, transformasi objek bisa dilakukan melalui query. Jadi hasil query criteria di simpan langsung ke format CSV, EXCEL, HTML, atau XML.

```
ObjectQuery q = em.CreateQuery<Customer>();
q.AddOrder("CompanyName", OrderBy.ASCENDING);
q.Transform(DocType.XML, "c:\\customers.xml");
```

Criteria dan Sorting

```
ObjectQuery q = em.CreateQuery<Product>();
q.AddCriteria(Criteria.Equal("CategoryID", 2));
q.AddOrder("ProductName", OrderBy.ASCENDING);
q.Transform(DocType.HTML, "c:\\products.htm");
```

Native Query

```
ObjectQuery q = em.CreateNativeQuery("SELECT * FROM Customers");
q.Transform(DocType.HTML, "c:\\customers.htm");
```

Named Native Query

```
ObjectQuery q = em.GetNamedQuery<Supplier>(SqlType.SELECT, "SelectAll");
q.Transform(DocType.HTML, "c:\\suppliers.htm");
```

6.16 ORM dan DAO Pattern

DAO Pattern merupakan katalog pattern yang dikeluarkan oleh SUN Microsystem dan terangkum dalam Core J2EE Pattern. Pattern ini menerapkan prinsip *separation of concern* atau pemisahan tanggung jawab dengan cara memisah class-class menjadi 3 layer yaitu : User Interface, Business Logic, dan Data Access. Sebuah ORM bisa juga memanfaatkan DAO Pattern untuk mendapatkan solusi yang lebih elegan. Berikut ini adalah contohnya :

Product

```
using System;
using Origami.Data.Mapping;

namespace OrigamiORM.Entity
{
    [Table(Name = "Products")]
    public class Product
    {
        private int productID;
        private string productName;
        private int categoryID;
        private Category category;
        private int supplierID;
        private Supplier supplier;
        private decimal unitPrice;
        private int unitInStock;

        //property dan mapping

    }
}
```

Category

```
using System;
using Origami.Data.Mapping;

namespace OrigamiORM.Entity
{
    [Table(Name="Categories")]
    public class Category
    {
        private int categoryID;
        private string categoryName;
        private string description;

        //property dan mapping

    }
}
```

Supplier

```
using System;
using Origami.Data.Mapping;

namespace OrigamiORM.Entity
{
    [Table(Name = "Suppliers")]
    public class Supplier
    {
        private int supplierId;
        private string companyName;
        private string contactName;
        private string address;
        private string city;
        private string country;
        private string phone;

        //property dan mapping

    }
}
```

IDataAccess

```
using System;
using System.Collections.Generic;

namespace ORMPattern.Repository
{
    public interface IDataAccess<T>
    {
        T GetByID(int id);
        List<T> GetAll();
        List<T> FindByName(string name);
        void Save(T obj);
        void Update(T obj);
        void Delete(T obj);
    }
}
```

CategoryDAO

```
using System;
using System.Collections.Generic;
using Origami.Configuration;
using Origami.Data;
using Origami.Data.Provider;
using Origami.Data.Query;
using Origami.Container;
using OrigamiORM.Entity;

namespace OrigamiORM.Repository
{
}
```

```

public class CategoryDao : IDataAccess<Category>
{
    private IEntityManager em;

    [Dependency(Name = "dataSource")]
    public CategoryDao(DataSource dataSource)
    {
        em = new EntityManager(dataSource);
    }

    public Category GetByID(int categoryID)
    {
        Category category = em.Get<Category>(categoryID);
        return category;
    }

    public List<Category> GetAll()
    {
        List<Category> list = em.Get<Category>();
        return list;
    }

    public List<Category> FindByName(string name)
    {
        ObjectQuery q = em.CreateQuery<Category>();
        q.AddCriteria(Criteria.Like("CategoryName", "%" + name + "%"));
        List<Category> list = q.List<Category>();
        return list;
    }

    public void Save(Category category)
    {
        em.Save(category);
    }

    public void Update(Category category)
    {
        em.Update(category);
    }

    public void Delete(Category category)
    {
        em.Delete(category);
    }
}

```

SupplierDAO

```

using System;
using System.Collections.Generic;
using Origami.Data;
using Origami.Data.Provider;
using Origami.Configuration;
using Origami.Data.Query;
using OrigamiORM.Entity;
using Origami.Container;

namespace OrigamiORM.Repository
{

```

```

public class SupplierDao : IDataAccess<Supplier>
{
    private IEntityManager em;

    [Dependency(Name = "dataSource")]
    public SupplierDao(DataSource dataSource)
    {
        em = new EntityManager(dataSource);
    }

    public Supplier GetByID(int supplierID)
    {
        Supplier supplier = em.Get<Supplier>(supplierID);
        return supplier;
    }

    public List<Supplier> GetAll()
    {
        List<Supplier> list = em.Get<Supplier>();
        return list;
    }

    public List<Supplier> FindByName(string name)
    {
        ObjectQuery q = em.CreateQuery<Supplier>();
        q.AddCriteria(Criteria.Like("CompanyName", "%" + name + "%"));
        List<Supplier> list = q.List<Supplier>();

        return list;
    }

    public void Save(Supplier supplier)
    {
        em.Save(supplier);
    }

    public void Update(Supplier supplier)
    {
        em.Update(supplier);
    }

    public void Delete(Supplier supplier)
    {
        em.Delete(supplier);
    }
}

```

ProductDAO

```

using System;
using System.Collections.Generic;
using Origami.Configuration;
using Origami.Data.Provider;
using Origami.Data;
using Origami.Data.Query;
using Origami.Container;
using OrigamiORM.Entity;
using OrigamiORM.Repository;

```

```

namespace OrigamiORM.Repository
{
    public class ProductDao : IDataAccess<Product>
    {
        private IEntityManager em;

        [Dependency(Name = "dataSource")]
        public ProductDao(DataSource dataSource)
        {
            em = new EntityManager(dataSource);
        }

        public Product GetByID(int productID)
        {
            Product product = em.Get<Product>(productID);
            return product;
        }

        public List<Product> GetAll()
        {
            List<Product> list = em.Get<Product>();
            return list;
        }

        public List<Product> FindByName(string name)
        {
            ObjectQuery q = em.CreateQuery<Product>();
            q.AddCriteria(Criteria.Like("CompanyName", "%" + name + "%"));
            List<Product> list = q.List<Product>();

            return list;
        }

        public void Save(Product product)
        {
            em.Save(product);
        }

        public void Update(Product product)
        {
            em.Update(product);
        }

        public void Delete(Product product)
        {
            em.Delete(product);
        }
    }
}

```

DAOFactory

```

using System;
using System.Collections.Generic;
using Origami.Container;
using Origami.Data.Provider;

```

```

namespace OrigamiORM.Repository
{
    public class DAOFactory
    {
        public static CategoryDao GetCategoryDaoInstance(
            IOrigamiContainer container)
        {
            return container.GetObject<CategoryDao, DataSource>("catDao");
        }

        public static SupplierDao GetSupplierDaoInstance(
            IOrigamiContainer container)
        {
            return container.GetObject<SupplierDao, DataSource>("suppDao");
        }

        public static ProductDao GetProductDaoInstance(
            IOrigamiContainer container)
        {
            return container.GetObject<ProductDao, DataSource>("prodDao");
        }
    }
}

```

Main Program

```

using System;
using System.Collections.Generic;
using Origami.Container;
using Origami.Data.Provider;
using OrigamiORM.Repository;
using OrigamiORM.Entity;

namespace ORMPattern
{
    class Program
    {
        static void Main(string[] args)
        {
            IOrigamiContainer container = new OrigamiContainer();
            IDataAccess<Product> prodDao =
                DAOFactory.GetProductDaoInstance(container);

            List<Product> list = prodDao.GetAll();
            foreach (Product p in list)
            {
                Console.WriteLine(p.ProductName + p.Category.CategoryName);
            }
            Console.ReadLine();
        }
    }
}

```

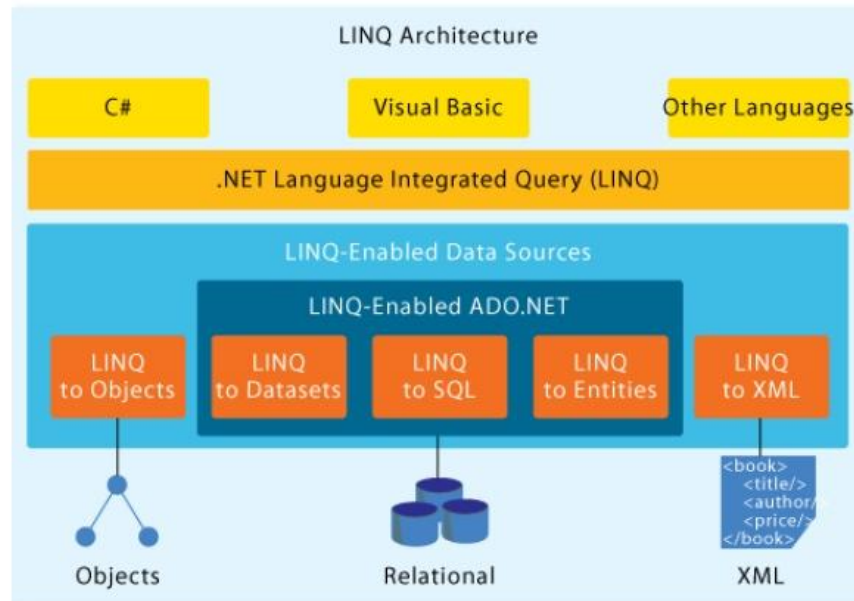
App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="objects" type="Origami.Configuration.ConfigurationHandler,
      Origami" />
  </configSections>
  <objects>
    <object id="dataSource" type="Origami.Data.Provider.DataSource,Origami">
      <property name="Provider" value="System.Data.SqlClient" />
      <property name="ConnectionString" value="Data
        Source=SYSTEMINTERFACE\SQLEXPRESS;Initial Catalog=NWIND;
        Integrated Security=True" />
    </object>
    <object id="catDao" type="OrigamiORM.Repository.CategoryDao,OrigamiSample" />
    <object id="suppDao" type="OrigamiORM.Repository.SupplierDao,OrigamiSample" />
    <object id="prodDao" type="OrigamiORM.Repository.ProductDao,OrigamiSample" />
  </objects>
</configuration>
```


BAB 7

Integrasi Dengan LINQ

LINQ adalah kependekan dari *Language Integrated Query*, sebuah bahasa query universal yang bertujuan untuk menyeragamkan cara dalam mengakses data dalam program, baik data dalam bentuk objek, database, atau XML. Sebelum ada LINQ developer memiliki cara berbeda untuk mengakses ketiga jenis data tersebut. Syntax LINQ merupakan bagian dari bahasa pemrograman (C# 3.0 dan VB 9.0) sehingga lebih compiler friendly. Walaupun syntax nya mirip perintah SQL, IDE dapat mendeteksi kesalahan penulisan syntax sehingga lebih type safe. Berikut adalah gambar arsitektur LINQ



Arsitektur LINQ

LINQ to SQL adalah API yang dibuat oleh Microsoft untuk solusi ORM (*Object Relational Mapping*). LINQ to SQL memudahkan developer dalam mengakses data dalam sebuah Relational Database dengan cara mengenkapsulasi record menjadi objek berdasarkan informasi meta data di entity class. Jadi sebetulnya fungsi LINQ to SQL dengan Origami ORM itu sama. Lalu dimana integrasinya? bukankah sia-sia memakai Origami ORM kalau Microsoft sendiri sudah menyediakan fitur ORM di Net Framework 3.5?

Sejujurnya saya sangat senang sekali ketika Microsoft berencana mengeluarkan fitur LINQ di NET Framework 3.5. Ketika membaca LINQ CTP (*Community Technology Preview*) saya

mengasosiasikan LINQ ini melalui hanya sebuah ORM, ternyata tidak hanya itu, ORM di LINQ (LINQ to SQL) hanya salah satu fitur untuk memudahkan hidup developer menjadi lebih mudah. Fitur andalan dari LINQ sebetulnya adalah sintaks query universal nya yang dapat meng-query apa saja. Sehingga jika kita punya solusi lain yang dianggap lebih baik dari LINQ to SQL - sepengetahuan saya LINQ to SQL hanya bisa berhubungan dengan MS SQL Server, belum bisa dengan database lain- kita masih bisa menggunakan ORM dari vendor lain atau yang open source seperti NHibernate tanpa harus mencampakkan LINQ. Justru kita akan mengambil banyak manfaat dengan menggunakan syntax LINQ. Kebanyakan ORM mengembalikan record dari Relational Database dalam bentuk object atau object collection, sehingga dengan demikian LINQ dapat digunakan untuk meng-query object tersebut.

Contoh Integrasi Origami dengan LINQ

Mapping

```
using System;
using System.Collections.Generic;
using Origami.Data.Mapping;

namespace OrigamiSample
{
    [Table(Name="Suppliers")]
    public class Supplier
    {
        private int supplierID;
        private string companyName;
        private string contactName;

        [Id(Name="SupplierID", IsIdentity=true)]
        public int SupplierID
        {
            get { return supplierID; }
            set { supplierID = value; }
        }

        [Column(Name="CompanyName")]
        public string CompanyName
        {
            get { return companyName; }
            set { companyName = value; }
        }

        [Column(Name="ContactName")]
        public string ContactName
        {
            get { return contactName; }
            set { contactName = value; }
        }
    }
}
```

Setelah membuat entity class mapping, buatlah sebuah class yang menghubungkan database dengan masing-masing entity class. Class tersebut mirip dengan DataContext di LINQ to SQL. Class ini mewarisi class EntityManager

Northwind

```
using System;
using System.Collections.Generic;
using Origami.Data;
using Origami.Data.Provider;
using Origami.Data.Mapping;

namespace OrigamiSample
{
    [Database(Name="NWIND")]

    public class Northwind : EntityManager
    {
        public Northwind(DataSource ds)
            : base(ds)
        {
        }

        public Northwind()
            : base(typeof(Northwind))
        {
        }

        public List<Supplier> Suppliers
        {
            get { return Get<Supplier>(); }
        }
    }
}
```

Constructor Northwind() kedua memerlukan atribut [Database] Yang merujuk ke file XML App.config

Class Northwind mewarisi EntityManager dari namespace Origami.Data. Class ini memiliki dua buah constructor. Constructor pertama memiliki parameter DataSource dari namespace Origami.Data.Provider. Constructor kedua tidak memiliki parameter apa-apa. Jika ingin menspesifikasikan DataSource secara programmatic maka gunakan constructor pertama, tapi jika ingin memanfaatkan Origami Container gunakan constructor kedua. DataSource merujuk ke objek yang telah didaftarkan pada sebuah file XML App.config, ditandai dengan penggunaan atribut [Database (Name="NWIND")] pada class Northwind. File App.config nya sebagai berikut :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="objects" type="Origami.Configuration.ConfigurationHandler,
      Origami" />
  </configSections>
  <objects>
    <object id="NWIND" type="Origami.Data.Provider.DataSource,Origami">
      <property name="Provider" value="System.Data.SqlClient" />
      <property name="ConnectionString" value="Data
        Source=SYSTEMINTERFSCE\SQLEXPRESS;Initial Catalog=NWIND;
        Integrated Security=True"/>
    </object>
  </objects>
</configuration>
```

```
</objects>
</configuration>
```

Instantiasi Northwind

```
DataSource dataSource = new DataSource();
dataSource.Provider="System.Data.SqlClient";
dataSource.ConnectionString="Data Source=SYSTEMINTERFACE\\SQLEXPRESS;Initial"
    + "Catalog=NWIND;Integrated Security=True";

Northwind db = new Northwind(dataSource);

//atau

Northwind db = new Northwind();

//Koneksi ke DataSource dilakukan oleh Container
```

Mengakses Objek dengan LINQ

```
foreach (Supplier s in db.Suppliers)
{
    Console.WriteLine(s.CompanyName);
}
```

Sorting

```
foreach (Supplier s in db.Suppliers.OrderBy(s=>s.CompanyName))
{
    Console.WriteLine(s.CompanyName);
}
```

Penggunaan Standard Query Operator dan Lambda Expression

```
Supplier supplier = db.Suppliers.Single(s => s.CompanyName == "Exotic Liquids");
Console.WriteLine(supplier.SupplierID);
Console.WriteLine(supplier.CompanyName);
Console.WriteLine(supplier.ContactName);
```

LINQ Query

```
var q = from s in db.Suppliers
        where s.CompanyName == "Exotic Liquids"
        select s;

foreach (Supplier s in q)
{
    Console.WriteLine(s.SupplierID);
}
```

```
    Console.WriteLine(s.CompanyName);  
}
```

SELECT, INSERT, UPDATE, DELETE

Select

```
foreach (Supplier s in db.Suppliers.Where(s=>s.Country=="USA"))  
{  
    Console.WriteLine(s.CompanyName);  
}
```

Insert

```
Supplier supplier = new Supplier()  
{  
    CompanyName="XERIS",  
    ContactName="Ariyanto"  
};  
  
db.Save(supplier);
```

Update

```
Supplier supplier = db.Suppliers.Single(s => s.SupplierID == 1);  
supplier.CompanyName = "XERIS";  
supplier.ContactName = "XERIS System Interface";  
  
db.Update(supplier);
```

Delete

```
Supplier supplier = db.Suppliers.Single(s => s.SupplierID == 1);  
db.Delete(supplier);
```

Relationship

Many To One

```
using System;
using Origami.Data.Mapping;

namespace OrigamiSample
{
    [Table(Name="Products")]
    public class Product
    {
        private int productID;
        private string productName;
        private Supplier supplier;

        [Id(Name="SupplierID", IsIdentity=true)]
        public int ProductID
        {
            get { return productID; }
            set { productID = value; }
        }

        [Column(Name="ProductName")]
        public string ProductName
        {
            get { return productName; }
            set { productName = value; }
        }

        [Association(EntityRef=typeof(Supplier), Key="SupplierID")]
        public Supplier Supplier
        {
            get { return supplier; }
            set { supplier = value; }
        }
    }
}
```

Tambahkan akses ke class Product di class Northwind

```
using System;
using System.Collections.Generic;
using Origami.Data;
using Origami.Data.Provider;
using Origami.Data.Mapping;

namespace OrigamiSample
{
    [Database(Name="NWIND")]
    public class Northwind : EntityManager
    {
        //Constructor
        //.....
    }
}
```

```

public List<Supplier> Suppliers
{
    get { return Get<Supplier>(); }
}

public List<Product> Products
{
    get { return Get<Product>(); }
}
}

```

Cara Akses

```

foreach (Product p in db.Products)
{
    Console.WriteLine("Product Name : " + p.ProductName);
    Console.WriteLine("Supplier Name : " + p.Supplier.CompanyName);
    Console.WriteLine("Contact      : " + p.Supplier.ContactName);
}

```

One To Many

```

using System;
using System.Collections.Generic;
using Origami.Data.Mapping;

namespace OrigamiSample
{
    [Table(Name="Suppliers")]
    public class Supplier
    {
        private int supplierID;
        private string companyName;
        private string contactName;

        private List<Product> products;

        //Property untuk SupplierID,CompanyName,dan ContactName
        //.....

        public List<Product> Products
        {
            get { return products; }
            set { products = value; }
        }
    }
}

```

Cara Akses

```
foreach (Supplier s in db.Suppliers)
{
    Console.WriteLine(s.CompanyName);
    s.Products = db.Products.Where(p => p.Supplier.SupplierID ==
        s.SupplierID).ToList();

    foreach (Product p in s.Products)
    {
        Console.WriteLine("\t" + p.ProductName);
    }
}
```

One To One

Untuk relasi One To One cara mapping di Entity class dan pengaksesannya mirip dengan relasi Many To One.

BAB 8

Unit Testing

8.1 Tahapan Pengembangan Software

Ketika mengembangkan perangkat lunak, paling tidak kita menggunakan sebuah metodologi tertentu agar software yang dihasilkan sesuai dengan requirement, sesuai dengan jadwal, dan tentu saja sesuai dengan budget. Ilmu yang digunakan untuk memastikan hal tersebut adalah *Software Engineering*. Banyak metodologi yang dikenal dalam *Software Engineering*, dari yang klasik sampai paling moderen. Mulai dari waterfall, RUP, MSF, Scrum sampai Xtreme Programming.

Tahapan pengembangan software :

Profesi	Proses	Kegiatan
System Analyst	<div>Analisis</div>	Menganalisis Requirement
Application Designer	<div>Desain</div>	Desain Database, user interface, report, dan arsitektur sistem
Programmer	<div>Coding</div>	Menulis program
Tester	<div>Testing</div>	Unit testing, user testing, integration testing

1. Requirement Gathering dan Analisis Kebutuhan

Tahap pertama adalah ***Requirement gathering***, yaitu mengumpulkan spesifikasi kebutuhan sebuah perangkat lunak dari semua stakeholder yang terlibat. Stakeholder adalah semua orang/organisasi yang terlibat dalam pengembangan sistem mulai dari top manager sampai end user. Proses ini bisa jadi sangat melelahkan tergantung seberapa besar dari sistem yang akan dibangun.

Requirement bisa diperoleh dari wawancara atau investigasi terhadap business process yang berjalan. Jika perusahaan sudah memiliki SOP (*Standar Operation Procedure*) yang jelas, investigasi bisa dilakukan secara bottom up dengan cara menganalisis laporan yang ingin

dihasilkan. Jika tidak ada SOP yang baku mau tidak mau kita lakukan pendekatan top-down, dengan cara wawancara semua stakeholder. Setelah requirement terkumpul barulah dimulai tahap analisis sistem. Pada tahap ini menghasilkan dokumen spesifikasi kebutuhan perangkat lunak atau disebut juga SRS (*Software Requirement Specification*). Proses analisis dilakukan oleh seorang atau lebih *system analyst*

2. Desain

Ruang lingkup fase ini adalah : desain database, desain user interface, dan desain arsitektur system. Berdasarkan hasil analisis dibuatkan tabel-tabel beserta relasi nya, selanjutnya dibuat juga tampilan user interface untuk input data ke tabel yang sudah dibuat. Dan terakhir merancang arsitektur aplikasi apakah stand alone, client-server atau terdistribusi. Perancangan arsitektur sistem sangat penting karena akan menentukan teknik apa yang digunakan berikutnya di fase coding. Sumber daya manusia yang dibutuhkan di fase ini cukup bervariasi, tergantung tingkat kompleksitas sistem. Teamwork bisa terdiri dari database designer, graphic designer, dan network administrator. Dokumen yang dihasilkan dari proses desain adalah SDD (*Software Design Description*)

3. Coding

Aktivitas coding sebagian besar dianggap sebagai kegiatan utama dalam proses software development, sehingga hampir keseluruhan time line project digunakan untuk menulis program. Ini sebenarnya cara pandang yang keliru, idealnya aktivitas coding tidak menyedot waktu paling banyak dalam project, agar fase lainnya seperti analisis dan desain bisa lebih lama. Dengan demikian diharapkan analisis lebih matang, dan desain jauh lebih baik.

Karena itu untuk mempercepat pemrograman hendaknya menggunakan *best practice* yang sudah diakui dan terbukti seperti menggunakan *pattern* yang sesuai, membiasakan *refactoring*, dan mengikuti prinsip-prinsip berorientasi objek seperti *separation of concern*, *dependency inversion*, *interface segregation*, dan *single responsibility principle*. Selain itu tidak ada salahnya menggunakan tools yang dapat mempercepat pemrograman seperti menggunakan code generator atau menggunakan library dan Framework tertentu.

4. Testing

Tujuan dari testing adalah memastikan software tidak memiliki bugs/kesalahan, atau setidaknya tidaknya menjamin bahwa software yang sedang dikembangkan sedikit bugs nya. Selain harus memenuhi requirement, sebuah software hendaknya memiliki sedikit bugs, apalagi jika software yang dikembangkan merupakan sistem kritis yang margin error nya harus sangat rendah.

Bayangkan sebuah sistem yang menangani reaktor nuklir atau pengendali rudal jika memiliki bugs, akan sangat fatal akibatnya bukan. Sebagai contoh pada tahun 1996, *European Space Agency* mengalami musibah dengan meledaknya roket Ariane 5, 59 detik setelah penerbangan. Tragedi lainnya terjadi pada NASA di program Mars Climate Orbiter yang menghabiskan dana \$125 juta, kedua kegagalan proyek tersebut disebabkan oleh kesalahan software. Software testing berhubungan erat dengan kualitas software (*software quality*).

8.2 Unit Testing

Unit Testing adalah test yang berfungsi memvalidasi bagian individual/bagian kecil dari source code untuk memastikan berjalan dengan semestinya. Pada pemrograman terstruktur, bagian/unit yang dites adalah function atau procedure, sedangkan pada pemrograman berorientasi objek unit terkecil adalah method, abstract class, atau class turunan. Unit testing biasanya dilakukan secara otomatis dengan menggunakan tools tertentu.

Dalam buku “Pragmatic Unit Testing” karya Andrew Hunt, disebutkan prinsip-prinsip utama dalam membuat unit testing yang baik, yaitu :

- Automatic
Testing harus bisa dilakukan secara otomatis dengan menggunakan tools, tertentu
- Through
Testing harus dilakukan secara keseluruhan
- Repeatable
Testing hasilnya tetap sama walaupun dilakukan secara berulang-ulang
- Independent
Tidak tergantung pada modul lain
- Professional
Menulis unit test sama prioritas nya seperti menulis source code utama

Fitur Unit testing di Origami :

- Testing dapat dilakukan untuk satu class (Test Class) atau lebih (Test Suite)
- Megetes keseluruhan method dalam class
- Menampilkan output ke Console
- Menampilkan kecepatan eksekusi tiap method dalam detik
- Metadata menggunakan custome attribute

Berikut ini contoh sederhana menggunakan unit testing :

Calculator

```
using System;

namespace OrigamiORM.Test
{
    public class Calculator
    {
        public int Add(int num1, int num2)
        {
            return num1 + num2;
        }

        public int Multiply(int num1, int num2)
        {
            return num1 * num2;
        }
    }
}
```

Class Calculator memiliki dua buah method yaitu : `Add()`, dan `Multiply()`. Kedua method ini berguna untuk menambah dan mengalikan dua buah bilangan integer. Masing-masing method memiliki return value dengan tipe integer. Untuk melakukan test pada class tersebut buatlah sebuah test class nya seperti berikut :

CalcualtorTest

```
using System;
using Origami.Testing;

namespace OrigamiORM.Test
{
    [TestClass]
    public class CalculatorTest
    {
        private Calculator cal;

        public CalculatorTest()
        {
            cal = new Calculator();
        }

        [TestMethod]
        public void TestAdd()
        {
            int result = cal.Add(1, 1);
            Assert.AreEqual(2, result);
        }

        [TestMethod]
        public void TestMultiply()
        {
            int result = cal.Multiply(1, 1);
            Assert.AreEqual(2, result);
        }
    }
}
```

Untuk menggunakan unit testing di Origami, namespace yang harus dipanggil adalah `Origami.Testing`.

Atribut yang digunakan :

Atribut	Keterangan
[TestClass]	Menandakan bahwa class tersebut adalah sebuah test class
[TestMethod]	Menandakan method yang akan di test

Setelah menandai semua method yang akan ditest dengan menggunakan atribut `[TestMethod]`, selanjutnya panggil static class **Assert** , berikut method-method yang ada :

- **AreEqual<T>()**
Membandingkan nilai expected(yang diharapkan) dengan nilai actual. Jika sama(equal) maka return value nya true

- **AreNotEqual<T>()**
Membandingkan nilai expected(yang diharapkan) dengan nilai actual. Jika tidak sama sama(*not equal*) maka return value nya true
- **AreSame<T>()**
Membandingkan apakah kedua objek sama
- **AreNotSame<T>()**
Membandingkan apakah kedua objek tidak sama
- **AreNotSame<T>()**
Membandingkan apakah kedua objek tidak sama
- **Greater()**
Membandingkan apakah suatu nilai lebih besar dari yang lain
- **Less()**
Membandingkan apakah suatu nilai lebih kecil dari yang lain
- **IsEmpty()**
Mengecek apakah sebuah string adalah kosong
- **IsTrue()**
Mengecek apakah sebuah variabel bertipe boolean bernilai True
- **IsFalse()**
Mengecek apakah sebuah variabel bertipe boolean bernilai false
- **IsNull()**
Mengecek apakah sebuah objek bernilai null
- **IsNotNull()**
Mengecek apakah sebuah objek bernilai tidak null
- **IsNaN()**
Mengecek apakah sebuah variabel bukan bilangan/number

AreEqual<T>()

```
int expected=2;
int actual = 1 + 1;
Assert.AreEqual(expected, actual);
```

AreNotEqual<T>()

```
int expected=2;
int actual = 2 + 1;
Assert.AreNotEqual(expected, actual);
```

AreSame<T>()

```
string expected = "Hello";
string actual = "Hello";
Assert.AreSame(expected, actual);
```

Greater()

```
int arg1 = 3;
int arg2 = 7;

Assert.Greater(arg1, arg2);
```

Less()

```
int arg1 = 7;
int arg2 = 3;
Assert.Less(arg1, arg2);
```

IsEmpty()

```
string str = "";

Assert.IsEmpty(str);
```

IsTrue()

```
bool b = true;
Assert.IsTrue(b);
```

IsFalse()

```
bool b = false;
Assert.IsFalse(b);
```

IsNull()

```
object obj = null;
Assert.IsNull(obj);
```

Test class dipanggil pada program utama

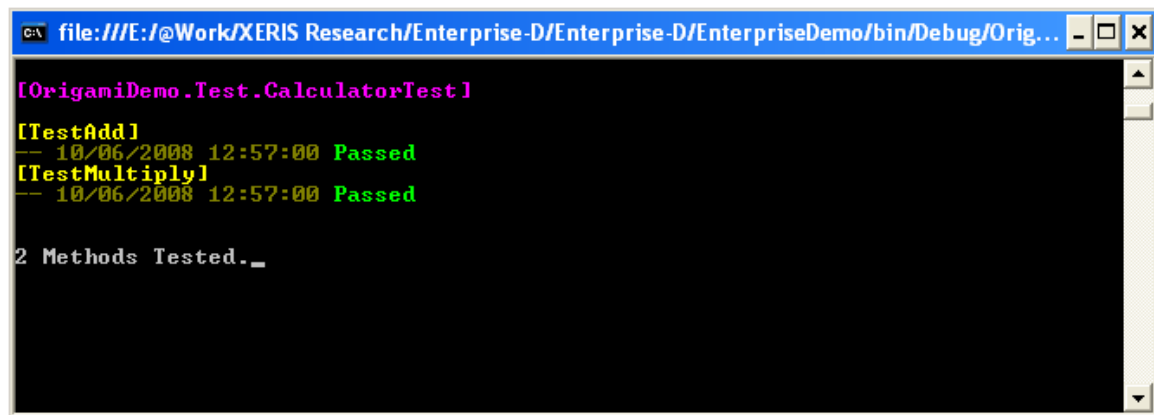
Main

```
using System;
using Origami.Testing;
using OrigamiORM.Test;

namespace OrigamiORM
{
    class Program
    {
        static void Main(string[] args)
        {
            ITest test = new TestCase(typeof(CalculatorTest));
            test.Run();

            Console.ReadLine();
        }
    }
}
```

Output



```
file:///E:/@Work/XERIS Research/Enterprise-D/Enterprise-D/EnterpriseDemo/bin/Debug/Orig...
[OrigamiDemo.Test.CalculatorTest]
[TestAdd]
-- 10/06/2008 12:57:00 Passed
[TestMultiply]
-- 10/06/2008 12:57:00 Passed

2 Methods Tested._
```

- **[OrigamiDemo.Test.CalculatorTest]**
Nama class yang sedang dites
- **[TestAdd], [TestMultiply]**
Nama method yang sedang dites
- **10/06/08 12:57:00 Passed**
Keterangan tanggal dan jam kapan tes tersebut dilakukan beserta hasilnya. “Passed” artinya kedua method tersebut sukses dites dan tidak memiliki kesalahan

DAO Test

```
using System;
using System.Collections.Generic;

using Origami.Container;
using Origami.Data.Provider;
using Origami.Testing;

using OrigamiORM.Repository;
using OrigamiORM.Entity;

namespace OrigamiORM.Test
{
    [TestClass]
    public class SupplierTest
    {
        private ISupplierDao suppDao;

        public SupplierTest()
        {
            IOrigamiContainer container = new OrigamiContainer();
            suppDao = container.GetObject<SupplierDao, DataSource>("suppDao");
        }

        [TestMethod]
        public void TestGetById()
        {
            Supplier supplier = suppDao.GetByID(1);
            Assert.AreEqual("Exotic Liquids", supplier.CompanyName);
        }

        [TestMethod]
        public void TestGetAll()
        {
            List<Supplier> suppliers = suppDao.GetAll();
            Assert.AreEqual(95, suppliers.Count);
        }

        [TestMethod]
        public void TestFindByName()
        {
            List<Supplier> suppliers = suppDao.FindByName("XERIS");
            Assert.AreEqual(1, suppliers.Count);
        }

        [TestMethod]
        public void TestSave()
        {
            Supplier supplier = new Supplier();
            supplier.CompanyName = "XERIS";
            supplier.ContactName = "Ari";
            supplier.Address = "";
            supplier.City = "";
            supplier.Country = "";
            supplier.Phone = "";

            Assert.AreEqual(1, suppDao.Save(supplier));
        }
    }
}
```



```

[TestMethod]
public void TestUpdate()
{
    Supplier supplier = new Supplier();
    supplier.SupplierID = 46;
    supplier.CompanyName = "XERIS System Interface";
    supplier.ContactName = "Ari";
    supplier.Address = "";
    supplier.City = "";
    supplier.Country = "";
    supplier.Phone = "";

    Assert.AreEqual(1, suppDao.Update(supplier));
}

[TestMethod]
public void TestDelete()
{
    Supplier supplier = new Supplier();
    supplier.SupplierID = 46;

    Assert.AreEqual(1, suppDao.Delete(supplier));
}
}

```

```

C:\ file:///F:/XERIS Research/Origami3/OrigamiSample/bin/Debug/OrigamiSample.EXE

[OrigamiORM.Test.SupplierTest]
[TestGetById]
--8/1/2008 4:54:02 PM Passed
[TestGetAll]
--8/1/2008 4:54:02 PM Failed Expected:<95>, Actual:<37>
[TestFindByName]
--8/1/2008 4:54:02 PM Passed
[TestSave]
--8/1/2008 4:54:02 PM Passed
[TestUpdate]
--8/1/2008 4:54:02 PM Passed
[TestDelete]
--8/1/2008 4:54:02 PM Passed

6 Methods Tested.

```

SupplierDao

```
using System;
using System.Collections.Generic;
using Origami.Data;
using Origami.Data.Provider;
using Origami.Data.Query;
using OrigamiORM.Entity;
using Origami.Container;
using Origami.Logging;

namespace OrigamiORM.Repository
{
    public class SupplierDao : ISupplierDao
    {
        private IEntityManager em;

        [Dependency(Name = "dataSource")]
        public SupplierDao(DataSource dataSource)
        {
            em = new EntityManager(dataSource);
        }

        public Supplier GetByID(int supplierID)
        {
            Supplier supplier = em.Get<Supplier>(supplierID);
            return supplier;
        }

        public List<Supplier> GetAll()
        {
            List<Supplier> list = em.Get<Supplier>();
            return list;
        }

        public List<Supplier> FindByName(string name)
        {
            ObjectQuery q = em.CreateQuery<Supplier>();
            q.AddCriteria(Criteria.Like("CompanyName", "%" + name + "%"));
            List<Supplier> list = q.List<Supplier>();
            return list;
        }

        public int Save(Supplier supplier)
        {
            return em.Save(supplier);
        }

        public int Update(Supplier supplier)
        {
            return em.Update(supplier);
        }

        public int Delete(Supplier supplier)
        {
            return em.Delete(supplier);
        }
    }
}
```

ISupplierDao

```
using System;
using OrigamiORM.Entity;
using Origami.Logging;

namespace OrigamiORM.Repository
{
    public interface ISupplierDao : IDataAccess<Supplier>
    {
    }
}
```

IDataAccess

```
using System;
using System.Collections.Generic;

namespace OrigamiORM.Repository
{
    public interface IDataAccess<T>
    {
        T GetByID(int id);
        List<T> GetAll();
        List<T> FindByName(string name);
        int Save(T obj);
        int Update(T obj);
        int Delete(T obj);
    }
}
```

Main

```
using Origami.Testing;
using OrigamiORM.Test;

namespace ORMPattern
{
    class Program
    {
        static void Main(string[] args)
        {
            TestCase test = new TestCase(typeof(SupplierTest));
            test.Run();

            Console.ReadLine();
        }
    }
}
```

BAB 9

Logging

Sebuah aplikasi yang masuk kategori *enterprise scale* seharusnya ditambahkan kemampuan *application logging* untuk memonitor kondisi aplikasi yang sedang berjalan. Logging ini sangat berperan jika suatu saat terjadi kesalahan/failure pada program, administrator atau programmer yang bertanggung jawab dapat dengan mudah melacak kesalahan yang terjadi dan segera melakukan perbaikan secepatnya.

Umumnya log aplikasi ini disimpan disebuah file text yang dapat dibaca dengan mudah yang berisi tanggal dan waktu kejadian beserta pesan kesalahan.

Origami menyediakan fitur logging yang cukup lengkap untuk monitoring aplikasi, diantaranya :

- Konfigurasi logger secara dinamis melalui XML
- Mendukung banyak tipe logging seperti : console, file, database, event log, SMTP, dan trace
- Mendukung penggunaan lebih dari satu tipe logger secara bersamaan
- Tipe logger dapat diubah dengan hanya dengan mengedit konfigurasinya .

Untuk menggunakan fitur logging di Origami, sebelumnya harus dipanggil terlebih dahulu namespace Origami.Logging. Selanjutnya kita bisa menggunakan class-class Logging yang tersedia

```
ConsoleLogger logger = new ConsoleLogger();  
  
logger.Write(Severity.Information, "Logging Initialize");  
logger.Write(Severity.Error, DateTime.Now + " Error Found at XFD67");
```

Origami Logging memiliki beberapa macam tipe logger

Class Logger	Keterangan
ConsoleLogger()	Output ke Console
FileLogger()	Output ke file
DbLogger()	Log ditulis ke tabel di basis data, tabel tersebut harus di buat terlebih dahulu dengan nama LOGGER
EventLogger()	Log ditulis ke Event Log milik Windows yang bisa diakses dari Control Panel -> Administrative Tools -> Event Viewer
Smtpllogger()	Log dikirim lewat e-mail melalui SMTP
TraceLogger()	Log dimunculkan di debug window Visual Studio.NET/Express Edition

Untuk menuliskan log ke gunakan method `Write()`

```
Write(Severity severity,string message)
```

Parameter **severity** pada method `Write` menyatakan jenis pesan log yang muncul yaitu : Debug, Error, Fatal, Information, dan Warning. Sedangkan **message** adalah pesan yang hendak dikirim pada log

File Logger

```
FileLogger logger = new FileLogger();  
logger.FileName = "c:\\log.txt";  
logger.Write(Severity.Information, "Log ini disimpan di file");
```

Db Logger

```
DbLogger logger = new DbLogger();  
  
logger.Provider = "System.Data.SqlClient";  
logger.ConnectionString = "Data Source=SYSTEMINTERFACE\\SQLEXPRESS;"  
    + "Initial Catalog=NWIND;Integrated Security=True";  
logger.Write(Severity.Information, "Log ini disimpan di database");
```

Event Logger

```
EventLogger logger = new EventLogger();  
logger.EventSource = "Application";  
logger.Write(Severity.Information, "Log ini ditulis ke EventLog Windows");
```

SMTP Logger

```
SmtplibLogger logger = new SmtplibLogger();  
  
logger.Host = "smtp.gmail.com";  
logger.Password = "secret";  
logger.Port = 25;  
logger.Mail = "neonerdy@gmail.com";  
logger.Destination = "myapplog@gmail.com";  
logger.Write(Severity.Information, "Log dikirim ke email");
```

Trace Logger

```
TraceLogger logger = new TraceLogger();  
logger.Write(Severity.Information, "Trace log");
```

9.1 Logging Menggunakan Origami Container

Selain menggunakan pemanggilan class-class logger secara langsung, Origami menyediakan cara yang lebih baik lagi dengan menggunakan **Container**. Objek logger cukup didefinisikan di App.config, dan container akan mengerjakan selebihnya untuk Anda. Hmm Origami membuat hidup Anda lebih mudah bukan?

Konfigurasi logger di App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="objects" type="Origami.Configuration.ConfigurationHandler,
      Origami" />
  </configSections>
  <objects>

    <object id="logger" type="Origami.Logging.ConsoleLogger,Origami"/>
  </objects>
</configuration>
```

Program Utama

```
IOrigamiContainer container = new OrigamiContainer();
ILogger logger = container.GetObject<ConsoleLogger>("logger");

logger.Write(Severity.Information, "Logging Initialize");
logger.Write(Severity.Error, DateTime.Now + " Error Found at XFD67");
```

Konfigurasi untuk beberapa tipe logger

File Logger

```
<object id="logger" type="Origami.Logging.FileLogger,Origami">
  <property name="FileName" value="c:\log.txt"/>
</object>
```

DB Logger

```
<object id="logger" type="Origami.Logging.DbLogger,Origami">
  <property name="Driver" value="System.Data.SqlClient"/>
  <property name="ConnectionString" value="
    Data Source=SYSTEMINTERFACE\SQLEXPRESS;
    Initial Catalog=NWIND;Integrated Security=True"/>
</object>
```

SMTP Logger

```
<object id="logger" type="Origami.Logging.FileLogger,Origami">
  <property name="Host" value="smtp.gmail.com"/>
  <property name="Port" value="25"/>
  <property name="Mail" value="neonerdy@yahoo.com"/>
  <property name="Password" value="secret"/>
  <property name="Destination" value="myapplog@gmail.com"/>
</object>
```

9.2 DAO Logging

Logging sebaiknya diletakkan pada bagian kode yang kira-kira berpeluang menimbulkan kesalahan. Sebuah Exception yang terjadi bisa kita lempar ke logger yang sebelumnya sudah didefinisikan.

Berikut ini saya akan menunjukkan bagaimana cara meletakkan Logger di layer Data Access untuk monitoring kemungkinan kesalahan yang terjadi pada proses manipulasi data.

```
using System;
using System.Collections.Generic;
using Origami.Data;
using Origami.Data.Provider;
using Origami.Configuration;
using Origami.Data.Query;
using OrigamiORM.Entity;
using Origami.Container;
using Origami.Logging;

namespace OrigamiORM.Repository
{
    public class SupplierDao : ISupplierDao
    {
        private ILogger logger;
        private IEntityManager em;

        [Dependency(Name = "dataSource")]
        public SupplierDao(DataSource dataSource)
        {
            em = new EntityManager(dataSource);
        }

        //setter dependency

        public ILogger Logger
        {
            get { return logger; }
            set { logger = value; }
        }
    }
}
```

```

    }

    public Supplier GetByID(int supplierID)
    {
        Supplier supplier = em.Get<Supplier>(supplierID);

        return supplier;
    }

    public List<Supplier> GetAll()
    {
        List<Supplier> list = em.Get<Supplier>();
        return list;
    }

    public List<Supplier> FindByName(string name)
    {
        ObjectQuery q = em.CreateQuery<Supplier>();
        q.AddCriteria(Criteria.Like("CompanyName", "%" + name + "%"));
        List<Supplier> list = q.List<Supplier>();

        return list;
    }

    public void Save(Supplier supplier)
    {
        try
        {
            em.Save(supplier);
        }
        catch (Exception ex)
        {
            logger.Write(Severity.Debug, ex.Message.ToString());
        }
    }

    public void Update(Supplier supplier)
    {
        try
        {
            em.Update(supplier);
        }
        catch (Exception ex)
        {
            logger.Write(Severity.Debug, ex.Message.ToString());
        }
    }

    public void Delete(Supplier supplier)
    {
        try
        {
            em.Delete(supplier);
        }
        catch (Exception ex)
        {
            logger.Write(Severity.Debug, ex.Message.ToString());
        }
    }
}

```


ISupplierDao

```
using System;
using OrigamiORM.Entity;
using Origami.Logging;

namespace OrigamiORM.Repository
{
    public interface ISupplierDao : IDataAccess<Supplier>
    {
        ILogger Logger { get; set; }
    }
}
```

Main

```
using System;
using Origami.Logging;
using Origami.Container;
using Origami.Data.Provider;
using OrigamiORM.Repository;

namespace OrigamiORM
{
    class Program
    {
        static void Main(string[] args)
        {
            IOrigamiContainer container = new OrigamiContainer();

            ISupplierDao suppDao = container.GetObject<SupplierDao,
                DataSource>("suppDao");
            suppDao.Logger = container.GetObject<ConsoleLogger>("logger");

            Console.ReadLine();
        }
    }
}
```

BAB 10

Security

10.1 Cryptography

Cryptography adalah teknik untuk menyandikan (*encode*) sebuah pesan menjadi tidak bisa dibaca oleh orang yang tidak berkepentingan. Teknik kriptografi menjamin kerahasiaan (*confidentiality*), integritas data (*data integrity*), dan pengesahan (*authentication*). Kerahasiaan berarti bahwa data yang dienkripsi teracak dan tersembunyi maknanya. Integritas data mencegah kerusakan data, dan pengesahan adalah pembuktian identitas dari pengirim untuk memastikan orang yang berhak.

Fitur Origami Cryptography :

- Mendukung Symmetric dan Asymmetric Cryptography
- Algoritma Symmetric yang didukung : DES, RC2, Rijndael, TripleDES
- Algoritma Asymmetric yang didukung : RC4
- Private key dan public key dapat dikonfigurasi secara dinamis pada App.config
- Mendukung algoritma Hashing : MD5, RIPEMD160, SHA1, SHA256, SHA384, SHA512, MACTripleDES, HMACSHA1

Untuk menggunakan fitur cryptography di Origami import namespace `Origami.Security`, tersedia 3 buah class yang berhubungan dengan proses enkripsi

Class	Keterangan	Algoritma
<code>SymmetricCryptography</code>	Menggunakan key/kunci yang sama untuk enkripsi dan dekripsi data	DES, RC2, Rijndael, TripleDES
<code>AsymmetricCryptography</code>	Menggunakan private key dan public key untuk enkripsi dan dekripsi data	RSA
<code>HashCryptography</code>	Digunakan untuk keperluan <i>digital signature</i>	MD5, RIPEMD160, SHA1, SHA256, SHA384, SHA512, MACTripleDES, HMACSHA1

Symmetric Cryptography

Encrypt

```
SymmetricCryptography crypto = new SymmetricCryptography(SymmetricProvider.DES);  
crypto.Key = "secret";  
crypto.Salt = "xf09uxc6";  
  
string encrypted = crypto.Encrypt("Helo World");  
Console.WriteLine(encrypted);
```

Source code diatas akan mengenkripsi teks “Hello World” dengan algoritma DES menjadi ONEUFoGzbbc43X/yj/Wl2g==

Decrypt

```
SymmetricCryptography crypto = new SymmetricCryptography(SymmetricProvider.DES);  
crypto.Key = "secret";  
crypto.Salt = "xf09uxc6";  
  
string decrypted = crypto.Decrypt("ONEUFoGzbbc43X/yj/Wl2g==");  
Console.WriteLine(encrypted);
```

Asymmetric Cryptography

```
AsymmetricCryptography crypto = new AsymmetricCryptography();  
crypto.PrivateKey = "c:\\private.txt";  
crypto.PublicKey = "c:\\public.txt";  
crypto crypt = asym.Encrypt("HELLO");
```

Output

wOw29NVUE4JxJBvKvAxHIT54mJiklUpUi3amTNOJpcYwhTZUVVx9GIj26AecJ5VUItPVcFUBi8NJFmRD
04eTUzPd+eMjkjLFc6xpSuK/y/L6skBG7g5qOD17ze4Wtk7MMWh0Lhni1jeDJ8zOeFwNmYr9Dyg+LJ+F
7nUpeDm+6MI=

Private key dan public key disimpan di file text. Untuk mendapatkan kedua kunci ini gunakan method `GenerateKeyPairs()`

```
AsymmetricCryptography asym = new AsymmetricCryptography();  
asym.GenerateKeyPairs("c:\\public.txt", "c:\\private.txt");
```

Dengan memanfaatkan fitur Origami Microkernel, algoritma dan key bisa dideklarasikan di Container yang dapat diubah secara dinamis

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="objects" type="Origami.Configuration.ConfigurationHandler,
      Origami" />
  </configSections>
  <objects>

    <object id="symmetric" type="Origami.Security.SymmetricCryptography,
      Origami">
      <property name="Algorithm" value="DES" />
      <property name="Key" value="secret" />
      <property name="Salt" value="xf09uxc6" />
    </object>

    <object id="asymmetric" type="Origami.Security.AsymmetricCryptography,
      Origami">
      <property name="PrivateKey" value="c:\private.txt" />
      <property name="PublicKey" value="c:\public.txt" />
    </object>

  </objects>
</configuration>
```

Symmetric

```
IOrigamiContainer container = new OrigamiContainer();
SymmetricCryptography crypto = container.GetObject<SymmetricCryptography>(
  "symmetric");
string crypt = crypto.Encrypt("HELLO");
Console.WriteLine(crypt);

string decrypt = crypto.Decrypt(crypt);
Console.WriteLine(decrypt);
```

Asymmetric

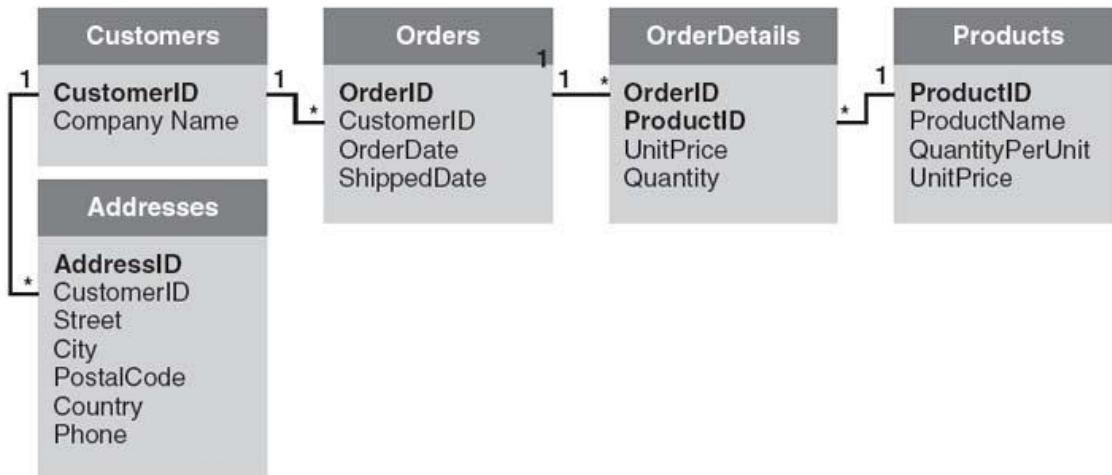
```
IOrigamiContainer container = new OrigamiContainer();

AsymmetricCryptography crypto = container.GetObject<AsymmetricCryptography>(
  "asymmetric");
string crypt = crypto.Encrypt("HELLO");
Console.WriteLine(crypt);
```

BAB 11

Contoh Kasus

Albert Einstein pernah berkata “Belajar melalui contoh adalah satu-satunya cara belajar”. Setelah melalui berbagai istilah teknis dan contoh-contoh program beripe Console, kali ini saya akan menunjukkan penerapan praktis pada sebuah aplikasi berbasis GUI (WinForm). Kita masih menggunakan database NWIND seperti pada contoh sebelumnya, perhatikan lagi ringkasan struktur tabel beserta relasi nya :



Buatlah entity class yang merepresentasikan tabel masing-masing, satu tabel satu class. Setelah itu lakukan mapping dengan menggunakan attribute

Customer

```
using System;
using Origami.Data.Mapping;

namespace OrigamiORM.Entity
{
    [Table(Name = "Customers")]
    public class Customer
    {
        private string customerId;
        private string companyName;
        private string contactName;
        private CustomerAddress customerAddress;

        [Id(Name = "CustomerID")]
        public string CustomerID
        {
            get { return customerId; }
            set { customerId = value; }
        }

        [Column(Name = "CompanyName")]
        public string CompanyName
        {
            get { return companyName; }
            set { companyName = value; }
        }

        [Column(Name = "ContactName")]
        public string ContactName
        {
            get { return contactName; }
            set { contactName = value; }
        }

        [Association(EntityRef=typeof(CustomerAddress), Key="CustomerID")]
        public CustomerAddress CustomerAddress
        {
            get { return customerAddress; }
            set { customerAddress = value; }
        }
    }
}
```

CustomerAddress

```
using System;
using Origami.Data.Mapping;

namespace OrigamiORM.Entity
{
    [Table(Name="CustomerAddress")]
    public class CustomerAddress
    {
        private string customerId;
        private string address;
        private string city;
```

```

        private string country;
        private string phone;

        [Column(Name = "CustomerID")]
        public string CustomerID
        {
            get { return customerID; }
            set { customerID = value; }
        }

        [Column(Name = "Address")]
        public string Address
        {
            get { return address; }
            set { address = value; }
        }

        [Column(Name = "City")]
        public string City
        {
            get { return city; }
            set { city = value; }
        }

        [Column(Name = "Country")]
        public string Country
        {
            get { return country; }
            set { country = value; }
        }

        [Column(Name = "Phone")]
        public string Phone
        {
            get { return phone; }
            set { phone = value; }
        }
    }
}

```

Order

```

using System;
using System.Collections.Generic;
using Origami.Data.Mapping;

namespace OrigamiORM.Entity
{
    [Table(Name = "Orders")]
    public class Order
    {
        private int orderID;
        private string customerID;
        private Customer customer;
        private int employeeID;
        private Employee employee;
        private DateTime orderDate;
        private string shipName;
    }
}

```

```

private string shipAddress;
private List<OrderDetail> orderDetails;

[Id(Name = "OrderID", IsIdentity=true)]
public int OrderID
{
    get { return orderID; }
    set { orderID = value; }
}

[Column(Name="CustomerID")]
public string CustomerID
{
    get { return customerID; }
    set { customerID = value; }
}

[Association(EntityRef=typeof(Customer), Key="CustomerID")]
public Customer Customer
{
    get { return customer; }
    set { customer = value; }
}

[Column(Name = "EmployeeID")]
public int EmployeeID
{
    get { return employeeID; }
    set { employeeID = value; }
}

[Association(EntityRef=typeof(Employee), Key="EmployeeID")]
public Employee Employee
{
    get { return employee; }
    set { employee = value; }
}

[Column(Name = "OrderDate")]
public DateTime OrderDate
{
    get { return orderDate; }
    set { orderDate = value; }
}

[Column(Name = "ShipName")]
public string ShipName
{
    get { return shipName; }
    set { shipName = value; }
}

[Column(Name = "ShipAddress")]
public string ShipAddress
{
    get { return shipAddress; }
    set { shipAddress = value; }
}

public List<OrderDetail> OrderDetails
{
    get { return orderDetails; }
}

```



```

        set { orderDetails = value; }
    }
}

```

OrderDetail

```

using System;
using Origami.Data.Mapping;

namespace OrigamiORM.Entity
{
    [Table(Name = "OrderDetails")]
    public class OrderDetail
    {
        private int orderID;
        private int productID;
        private Product product;
        private int qty;
        private decimal unitPrice;
        private double discount;

        [Column(Name = "OrderID")]
        public int OrderID
        {
            get { return orderID; }
            set { orderID = value; }
        }

        [Column(Name = "ProductID")]
        public int ProductID
        {
            get { return productID; }
            set { productID = value; }
        }

        [Association(EntityRef=typeof(Product),Key="ProductID")]
        public Product Product
        {
            get { return product; }
            set { product = value; }
        }

        [Column(Name = "Quantity")]
        public int Qty
        {
            get { return qty; }
            set { qty = value; }
        }

        [Column(Name = "UnitPrice")]
        public decimal UnitPrice
        {
            get { return unitPrice; }
            set { unitPrice = value; }
        }

        [Column(Name = "Discount")]
        public double Discount
    }
}

```

```

        {
            get { return discount; }
            set { discount = value; }
        }
    }
}

```

Product

```

using System;
using Origami.Data.Mapping;

namespace OrigamiORM.Entity
{
    [Table(Name = "Products")]
    public class Product
    {
        private int productID;
        private string productName;
        private int categoryID;
        private Category category;
        private int supplierID;
        private Supplier supplier;
        private decimal unitPrice;
        private int unitInStock;

        [Id(Name = "ProductID", IsIdentity=true)]
        public int ProductID
        {
            get { return productID; }
            set { productID = value; }
        }

        [Column(Name = "ProductName")]
        public string ProductName
        {
            get { return productName; }
            set { productName = value; }
        }

        [Column(Name="CategoryID")]
        public int CategoryID
        {
            get { return categoryID; }
            set { categoryID = value; }
        }

        [Association(EntityRef = typeof(Category), Key="CategoryID")]
        public Category Category
        {
            get { return category; }
            set { category = value; }
        }

        [Column(Name = "SupplierID")]
        public int SupplierID
        {
            get { return supplierID; }
            set { supplierID = value; }
        }
    }
}

```

```

        [Association(EntityRef = typeof(Supplier), Key="SupplierID")]
        public Supplier Supplier
        {
            get { return supplier; }
            set { supplier = value; }
        }

        [Column(Name = "UnitPrice")]
        public decimal UnitPrice
        {
            get { return unitPrice; }
            set { unitPrice = value; }
        }

        [Column(Name = "UnitsInStock")]
        public int UnitInStock
        {
            get { return unitInStock; }
            set { unitInStock = value; }
        }
    }
}

```

Supplier

```

using System;
using Origami.Data.Mapping;

namespace OrigamiORM.Entity
{
    [Table(Name = "Suppliers")]
    public class Supplier
    {
        private int supplierId;
        private string companyName;
        private string contactName;
        private string address;
        private string city;
        private string country;
        private string phone;

        [Id(Name = "SupplierId", IsIdentity=true)]
        public int SupplierID
        {
            get { return supplierId; }
            set { supplierId = value; }
        }

        [Column(Name = "CompanyName")]
        public string CompanyName
        {
            get { return companyName; }
            set { companyName = value; }
        }

        [Column(Name = "ContactName")]
        public string ContactName
        {

```

```

        get { return contactName; }
        set { contactName = value; }
    }

    [Column(Name = "Address")]
    public string Address
    {
        get { return address; }
        set { address = value; }
    }

    [Column(Name = "City")]
    public string City
    {
        get { return city; }
        set { city = value; }
    }

    [Column(Name = "Country")]
    public string Country
    {
        get { return country; }
        set { country = value; }
    }

    [Column(Name = "Phone")]
    public string Phone
    {
        get { return phone; }
        set { phone = value; }
    }
}

```

Category

```

using System;
using System.Collections.Generic;
using Origami.Data.Mapping;

namespace OrigamiORM.Entity
{
    [Table(Name="Categories")]
    public class Category
    {
        private int categoryID;
        private string categoryName;
        private string description;
        private List<Product> products;

        [Id(Name="CategoryId", IsIdentity=true)]
        public int CategoryID
        {
            get { return categoryID; }
            set { categoryID = value; }
        }

        [Column(Name="CategoryName")]
        public string CategoryName
    }
}

```

```

        {
            get { return categoryName; }
            set { categoryName = value; }
        }

        [Column(Name="Description")]
        public string Description
        {
            get { return description; }
            set { description = value; }
        }

        public List<Product> Products
        {
            get { return products; }
            set { products = value; }
        }
    }
}

```

Northwind

```

using System;
using System.Collections.Generic;
using Origami.Data;
using Origami.Data.Mapping;
using Origami.Data.Provider;
using OrigamiORM.Entity;

namespace OrigamiORM.Repository
{
    [Database(Name = "NWIND")]
    public class Northwind : EntityManager
    {
        public Northwind():base(typeof(Northwind))
        {
        }

        public List<Supplier> Suppliers
        {
            get { return Get<Supplier>(); }
        }

        public List<Category> Categories
        {
            get { return Get<Category>(); }
        }

        public List<Product> Products
        {
            get { return Get<Product>(); }
        }

        public List<Customer> Customers
        {
            get { return Get<Customer>(); }
        }
    }
}

```

```

    public List<Order> Orders
    {
        get { return Get<Order>(); }
    }

    public List<OrderDetail> OrderDetails
    {
        get { return Get<OrderDetail>(); }
    }
}

```

App.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="objects" type="Origami.Configuration.ConfigurationHandler,
      Origami" />
  </configSections>
  <objects>
    <object id="NWIND" type="Origami.Data.Provider.DataSource,Origami">
      <property name="Provider" value="System.Data.SqlClient" />
      <property name="ConnectionString" value="Data
        Source=SYSTEMINTERFACE\SQLEXPRESS;
        Initial Catalog=NWIND;Integrated Security=True" />
    </object>
  </objects>
</configuration>

```

Form ProductUI

Product - 13 Item (s) in Condiments Category

ID	Product Name	Category	Supplier	Price	Stock
1	Chicken Legs	Condiments	Grandma Kelly's Homestead	40,0000	100
3	Aniseed Syrup	Condiments	Exotic Liquids	10,0000	13
4	Chef Anton's Cajun Sea...	Condiments	New Orleans Cajun Delights	22,0000	53
5	Chef Anton's Gumbo Mix	Condiments	New Orleans Cajun Delights	21,0000	0
6	Grandma's Boysenberry ...	Condiments	Grandma Kelly's Homestead	25,0000	120
8	Northwoods Cranberry S...	Condiments	Grandma Kelly's Homestead	40,0000	6
15	Genen Shouyu	Condiments	Cooperativa de Quesos 'Las ...	17,4500	39
44	Gula Malacca	Condiments	Aux joyeux ecclésiastiques	0	27
61	Sirop d'érable	Condiments	G'day, Mate	0	113
63	Veggie-spread	Condiments	Mayumi's	0	24
65	Louisiana Fiery Hot Pep...	Condiments	New Orleans Cajun Delights	0	76
66	Louisiana Hot Spiced O...	Condiments	New Orleans Cajun Delights	0	4
77	Original Frankfurter grün...	Condiments	Refrescos Americanas LTDA	0	32

ProductUI

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;

using Origami.Data;
using Origami.Data.Provider;
using Origami.Data.Query;

using OrigamiORM.Entity;
using OrigamiORM.Repository;

namespace OrigamiORM.View
{
    public partial class ProductUI : Form
    {
        private Northwind db;

        public ProductUI()
        {
            InitializeComponent();
            db = new Northwind();
        }

        private void FillListView(Product p)
        {
            ListViewItem item = new ListViewItem(p.ProductID.ToString());

            item.SubItems.Add(p.ProductName);
            item.SubItems.Add(p.Category.CategoryName);
            item.SubItems.Add(p.Supplier.CompanyName);
            item.SubItems.Add(p.UnitPrice.ToString());
            item.SubItems.Add(p.UnitInStock.ToString());

            lvwProduct.Items.Add(item);
        }

        private void FillCategory()
        {
            foreach (Category c in db.Categories)
            {
                cbCategory.Items.Add(c.CategoryName);
            }
        }

        private void GetByCategory()
        {
            ObjectQuery q = db.CreateQuery<Product>();
            q.AddCriteria(Criteria.Equal("CategoryName", cbCategory.Text));

            List<Product> list = q.List<Product>();

            lvwProduct.Items.Clear();
            foreach (Product p in list)
```

```

        {
            FillListView(p);
        }
        this.Text = "Product - " + list.Count + " Item (s) in "
            + cbCategory.Text + " Category";
    }

    private void FillProduct()
    {
        foreach (Product p in db.Products)
        {
            FillListView(p);
        }
    }

    private void FindProduct()
    {
        ObjectQuery q = db.CreateQuery<Product>();
        q.AddCriteria(Criteria.Like("ProductName", "%"
            + txtFind.Text + "%"));
        List<Product> products = q.List<Product>();

        lvwProduct.Items.Clear();

        foreach (Product p in products)
        {
            FillListView(p);
        }
    }

    private void ProductUI_Load(object sender, EventArgs e)
    {
        FillCategory();
        FillProduct();
    }

    private void cbCategory_SelectedIndexChanged(object sender, EventArgs e)
    {
        GetByCategory();
    }

    private void btnFind_Click(object sender, EventArgs e)
    {
        FindProduct();
    }
}

```


Form SupplierUI

Supplier

Search

List Detail

ID	Company Name	Contact Name	Address	City	Phone
1	Exotic Liquids	Charlotte Cooper	49 Gilbert St.	London	UK
2	New Orleans Cajun Deli...	Shelley Burke	P.O. Box 78934	New Orl...	USA
3	Grandma Kelly's Homest...	Regina Murphy	707 Oxford Rd.	Ann Arbor	USA
4	Tokyo Traders	Yoshi Nagase	"9-8 Sekimai		
5	Tokyo		Japan	(03) 355...	
6	Cooperativa de Quesos '...	Antonio del Valle S...	Calle del Rosal 4	Oviedo	Spain
7	Mayumi's	Mayumi Ohno	"92 Setsuko		
8	Pavlova, Ltd.	Ian Devling	"74 Rose St.		
9	Melbourne	Victoria	Australia	(03) 444...	
10	Specialty Biscuits, Ltd.	Peter Wilson	29 King's Way	Manche...	UK
11	PB Knäckebröd AB	Lars Peterson	Kaloadagatan 13	Göteborg	Sweden
12	Refrescos Americanas L...	Carlos Diaz	Av. das Americanas 12.890	São Pa...	Brazil
13	Heli Süßwaren GmbH & ...	Petra Winkler	Tiergartenstraße 5	Berlin	Germany
14	Nord-Ost-Fisch Handels...	Sven Petersen	Frahmredder 112a	Cuxhaven	Germany
15	Formaggi Fortini s.r.l.	Elio Rossi	Viale Dante, 75	Ravenna	Italy
16	Norske Meierier	Beate Vileid	Hatlevegen 5	Sandvika	Norway
17	Bigfoot Breweries	Cheryl Saylor	"3400 - 8th Avenue		
18	Bend	OR	USA	(503) 55...	
19	Svensk Sjöföda AB	Michael Björn	Brovallavägen 231	Stockh...	Sweden
20	Aux joyeux ecclésiastiqu...	Guylène Nodier	203, Rue des Francs-Bourge...	Paris	France

Supplier

Search

List Detail

Company Name: Exotic Liquids

Contact Name: Charlotte Cooper

Address: 49 Gilbert St.

City: London

Country: UK

Phone: (171) 555-2222

SupplierUI

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

using Origami.Data;
using Origami.Data.Provider;
using Origami.Data.Query;

using OrigamiORM.Entity;
using OrigamiORM.Repository;

namespace OrigamiORM.View
{
    public partial class SupplierUI : Form
    {
        private Northwind db;
        private BindingSource bs;

        public SupplierUI()
        {
            InitializeComponent();
            db = new Northwind();
        }

        private void FillListView(Supplier s)
        {
            ListViewItem item = new ListViewItem(s.SupplierID.ToString());

            item.SubItems.Add(s.CompanyName);
            item.SubItems.Add(s.ContactName);
            item.SubItems.Add(s.Address);
            item.SubItems.Add(s.City);
            item.SubItems.Add(s.Country);
            item.SubItems.Add(s.Phone);

            lvwSupplier.Items.Add(item);
        }

        private void FillSupplier()
        {
            bs = new BindingSource();
            bs.DataSource = db.Suppliers;

            lvwSupplier.Items.Clear();
            foreach (Supplier s in db.Suppliers)
            {
                FillListView(s);
            }
        }

        //data binding

        private void DataBind()
```

```

    {
        txtCompanyName.DataBindings.Add(new Binding("Text", bs,
            "CompanyName"));
        txtContactName.DataBindings.Add(new Binding("Text", bs,
            "ContactName"));
        txtAddress.DataBindings.Add(new Binding("Text", bs, "Address"));
        txtCity.DataBindings.Add(new Binding("Text", bs, "City"));
        txtCountry.DataBindings.Add(new Binding("Text", bs, "Country"));
        txtPhone.DataBindings.Add(new Binding("Text", bs, "Phone"));
    }

private void Find()
{
    ObjectQuery q = db.CreateQuery<Supplier>();
    q.AddCriteria(Criteria.Like("CompanyName", "%"
        + txtFind.Text + "%"));

    List<Supplier> suppliers=q.List<Supplier>();

    lvwSupplier.Items.Clear();
    foreach (Supplier s in suppliers)
    {
        FillListView(s);
    }
}

private void ViewDetail()
{
    Supplier supplier = db.Get<Supplier>(lvwSupplier.FocusedItem.Text);

    txtCompanyName.Text = supplier.CompanyName;
    txtContactName.Text = supplier.ContactName;
    txtAddress.Text = supplier.Address;
    txtCity.Text = supplier.City;
    txtCountry.Text = supplier.Country;
    txtPhone.Text = supplier.Phone;
}

private void Clear()
{
    txtCompanyName.Clear();
    txtContactName.Clear();
    txtAddress.Clear();
    txtCity.Clear();
    txtCountry.Clear();
    txtPhone.Clear();
}

private void Save()
{
    Supplier supplier = new Supplier();
    supplier.CompanyName = txtCompanyName.Text;
    supplier.ContactName = txtContactName.Text;
    supplier.Address = txtAddress.Text;
    supplier.City = txtCity.Text;
    supplier.Country = txtCountry.Text;
    supplier.Phone = txtPhone.Text;

    db.Save(supplier);
}

```

```

private void Delete()
{
    Supplier supplier = new Supplier();
    supplier.SupplierID = Convert.ToInt32(lvwSupplier.FocusedItem.Text);
    db.Delete(supplier);
}

private void Export()
{
    db.Transform<Supplier>(DocType.HTML, "c:\\supplier.htm");
    MessageBox.Show("Success", "Information",
        MessageBoxButtons.OK, MessageBoxIcon.Information);
}

private void Form1_Load(object sender, EventArgs e)
{
    FillSupplier();
    DataBind();
}

private void btnFind_Click(object sender, EventArgs e)
{
    Find();
    tabControl1.SelectedTab = this.tabPage1;
}

private void lvwSupplier_DoubleClick(object sender, EventArgs e)
{
    ViewDetail();
    tabControl1.SelectedTab = this.tabPage2;
}

private void btnNew_Click(object sender, EventArgs e)
{
    tabControl1.SelectedTab = this.tabPage2;
    Clear();
}

private void btnSave_Click(object sender, EventArgs e)
{
    Save();
    Clear();
    FillSupplier();
    tabControl1.SelectedTab = this.tabPage1;
}

private void btnDelete_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("Are you sure want to delete '"
        + lvwSupplier.FocusedItem.SubItems[1].Text + "'", "Confirm",
        MessageBoxButtons.YesNo, MessageBoxIcon.Question) ==
        DialogResult.Yes)
    {
        Delete();
        FillSupplier();
    }
}

//export to HTML

private void btnExport_Click(object sender, EventArgs e)
{

```

```

        Export();
    }

    //move to previous record

    private void btnPrev_Click(object sender, EventArgs e)
    {
        bs.MovePrevious();
        this.Text = "Supplier - Record " + (bs.Position+1)
            + " From " + bs.Count;
    }

    //move to next record

    private void btnNext_Click(object sender, EventArgs e)
    {
        bs.MoveNext();
        this.Text = "Supplier - Record " + (bs.Position+1)
            + " From " + bs.Count;
    }
}

```

Form OrderUI

Order - 13 Order (s) from Around the Horn

Customer: Around the Horn Search

ID	Order Date	Customer	Employee	Ship Name	Ship Address
10355	16/12/1994	Around the Horn	Michael	Around the Horn	"Brook Farm
10383	16/01/1995	Around the Horn	Laura	Around the Horn	"Brook Farm
10453	24/03/1995	Around the Horn	Nancy	Around the Horn	"Brook Farm
10558	05/07/1995	Around the Horn	Nancy	Around the Horn	"Brook Farm
10707	16/11/1995	Around the Horn	Margaret	Around the Horn	"Brook Farm
10741	15/12/1995	Around the Horn	Margaret	Around the Horn	"Brook Farm
10743	18/12/1995	Around the Horn	Nancy	Around the Horn	"Brook Farm
10768	08/01/1996	Around the Horn	Janet	Around the Horn	"Brook Farm
10793	24/01/1996	Around the Horn	Janet	Around the Horn	"Brook Farm
10864	04/03/1996	Around the Horn	Margaret	Around the Horn	"Brook Farm
10920	02/04/1996	Around the Horn	Margaret	Around the Horn	"Brook Farm
10953	15/04/1996	Around the Horn	Anne	Around the Horn	"Brook Farm
11016	10/05/1996	Around the Horn	Anne	Around the Horn	"Brook Farm

Order

Customer Search

Order Order Details

Order ID : 10262
Order Date : 22/08/1994 0:00:00

Shipped To
Rattlesnake Canyon Grocery
2817 Milton Dr.

Product Name	Quantity	Unit Price	Discount
Chef Anton's Gumbo Mix	12	17,0000	0
Uncle Bob's Organic Dried Pears	15	24,0000	0
Gnocchi di nonna Alice	2	30,0000	0

3 Item(s) **Total : 624,0000**

OrderUI

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

using Origami.Data;
using Origami.Data.Provider;
using Origami.Data.Query;
using OrigamiORM.Entity;
using OrigamiORM.Repository;

namespace OrigamiORM.View
{
    public partial class OrderUI : Form
    {
        private BindingSource bs;
        private Northwind db;

        public OrderUI()
        {
            InitializeComponent();
            db = new Northwind();
        }
    }
}
```

```

private void FillListView(Order o)
{
    ListViewItem item = new ListViewItem(o.OrderID.ToString());

    item.SubItems.Add(o.OrderDate.ToShortDateString());
    item.SubItems.Add(o.Customer.CompanyName);
    item.SubItems.Add(o.Employee.FirstName);
    item.SubItems.Add(o.ShipName);
    item.SubItems.Add(o.ShipAddress);

    lvwOrder.Items.Add(item);
}

private void FillCustomer()
{
    foreach (Customer c in db.Customers)
    {
        cbCustomer.Items.Add(c.CompanyName);
    }
}

private void GetByCustomer()
{
    ObjectQuery q = db.CreateQuery<Order>();
    q.AddCriteria(Criteria.Equal("CompanyName", cbCustomer.Text));

    List<Order> list = q.List<Order>();

    lvwOrder.Items.Clear();
    foreach (Order o in list)
    {
        FillListView(o);
    }
    this.Text = "Order - " + list.Count + " Order (s) from "
        + cbCustomer.Text;
}

private void FillOrder()
{
    lvwOrder.Items.Clear();
    foreach (Order o in db.Orders)
    {
        FillListView(o);
    }

    bs = new BindingSource();
    bs.DataSource = db.Orders;
}

private void DataBind()
{
    lblOrderID.DataBindings.Add(new Binding("Text", bs, "OrderID"));
    lblOrderDate.DataBindings.Add(new Binding("Text", bs, "OrderDate"));
}

```

```

private void FindOrder()
{
    ObjectQuery q = db.CreateQuery<Order>();
    q.AddCriteria(Criteria.Equal("OrderDate", txtFind.Text));
    List<Order> orders = q.List<Order>();

    lvwOrder.Items.Clear();

    foreach (Order o in orders)
    {
        FillListView(o);
    }
}

private void GetOrderByID(string id)
{
    Order order=db.Get<Order>(id);

    lblOrderID.Text = "Order ID : " + order.OrderID;
    lblOrderDate.Text = "Order Date : " + order.OrderDate;
    lblCustomer.Text = order.Customer.CompanyName;
    lblShipAddress.Text = order.ShipAddress;

    ObjectQuery q=db.CreateQuery<OrderDetail>();
    q.AddCriteria(Criteria.Equal("OrderID", order.OrderID));
    order.OrderDetails = q.List<OrderDetail>();

    lvwOrderDetail.Items.Clear();

    decimal totalPrice = 0;

    foreach (OrderDetail orderdetail in order.OrderDetails)
    {
        ListViewItem item = new ListViewItem(
            orderdetail.Product.ProductName);
        item.SubItems.Add(orderdetail.Qty.ToString());
        item.SubItems.Add(orderdetail.UnitPrice.ToString());
        item.SubItems.Add(orderdetail.Discount.ToString());
        lvwOrderDetail.Items.Add(item);

        totalPrice =totalPrice+(orderdetail.Qty *
            orderdetail.UnitPrice);
    }

    lblItem.Text = order.OrderDetails.Count + " Item (s)";
    lblTotal.Text = "Total : " + totalPrice;
}

private void Delete()
{
    Order order = new Order();
    order.OrderID = Convert.ToInt32(lvwOrder.FocusedItem.Text);
    db.Delete(order);
}

private void Export()
{
    db.Transform<Order>(DocType.EXCEL, "c:\\order.xls");
    MessageBox.Show("Success", "Information", MessageBoxButtons.OK,
        MessageBoxIcon.Information);
}

```



```

    }

    private void OrderUI_Load(object sender, EventArgs e)
    {
        FillOrder();
        FillCustomer();
        DataBind();
    }

    private void cbCustomer_SelectedIndexChanged(object sender, EventArgs e)
    {
        GetByCustomer();
        tabControl1.SelectedTab = this.tabPage1;
    }

    private void btnFind_Click(object sender, EventArgs e)
    {
        FindOrder();
        tabControl1.SelectedTab = this.tabPage1;
    }

    private void lvwOrder_DoubleClick(object sender, EventArgs e)
    {
        GetOrderByID(lvwOrder.FocusedItem.Text);
        tabControl1.SelectedTab = this.tabPage2;
    }

    private void btnDelete_Click(object sender, EventArgs e)
    {
        if (MessageBox.Show("Are you sure want to delete '"
            + lvwOrder.FocusedItem.SubItems[1].Text + "'", "Confirm",
            MessageBoxButtons.YesNo, MessageBoxIcon.Question) ==
            DialogResult.Yes)
        {
            Delete();
            FillOrder();
        }
    }

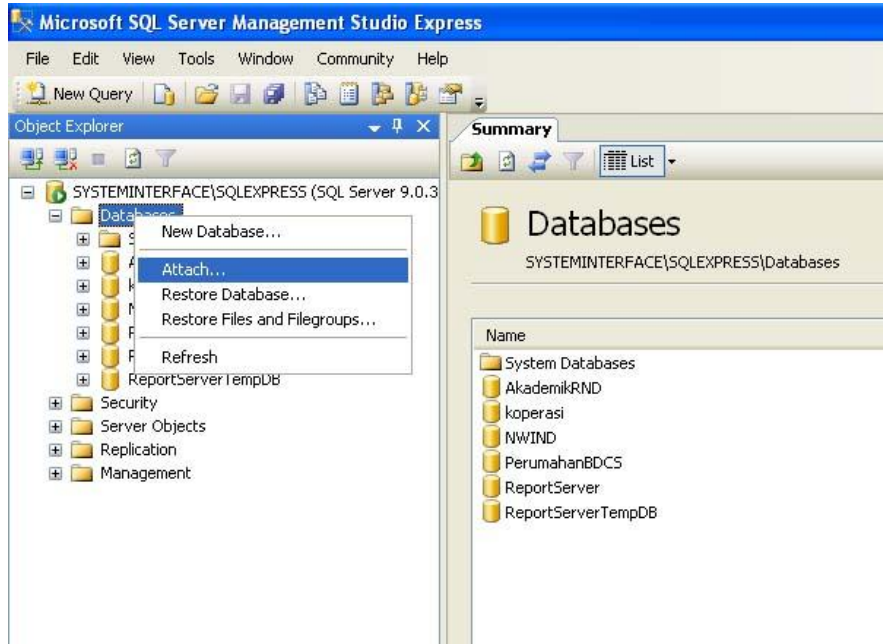
    private void btnExport_Click(object sender, EventArgs e)
    {
        Export();
    }
}

```

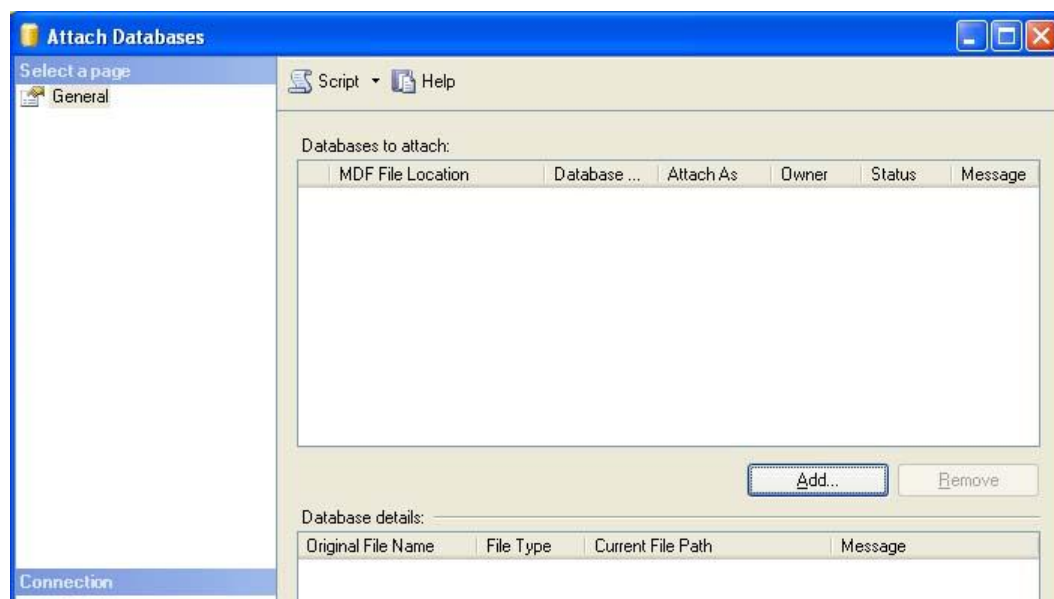
Lampiran

Meng-Import File Database ke SQL Server Express 2005

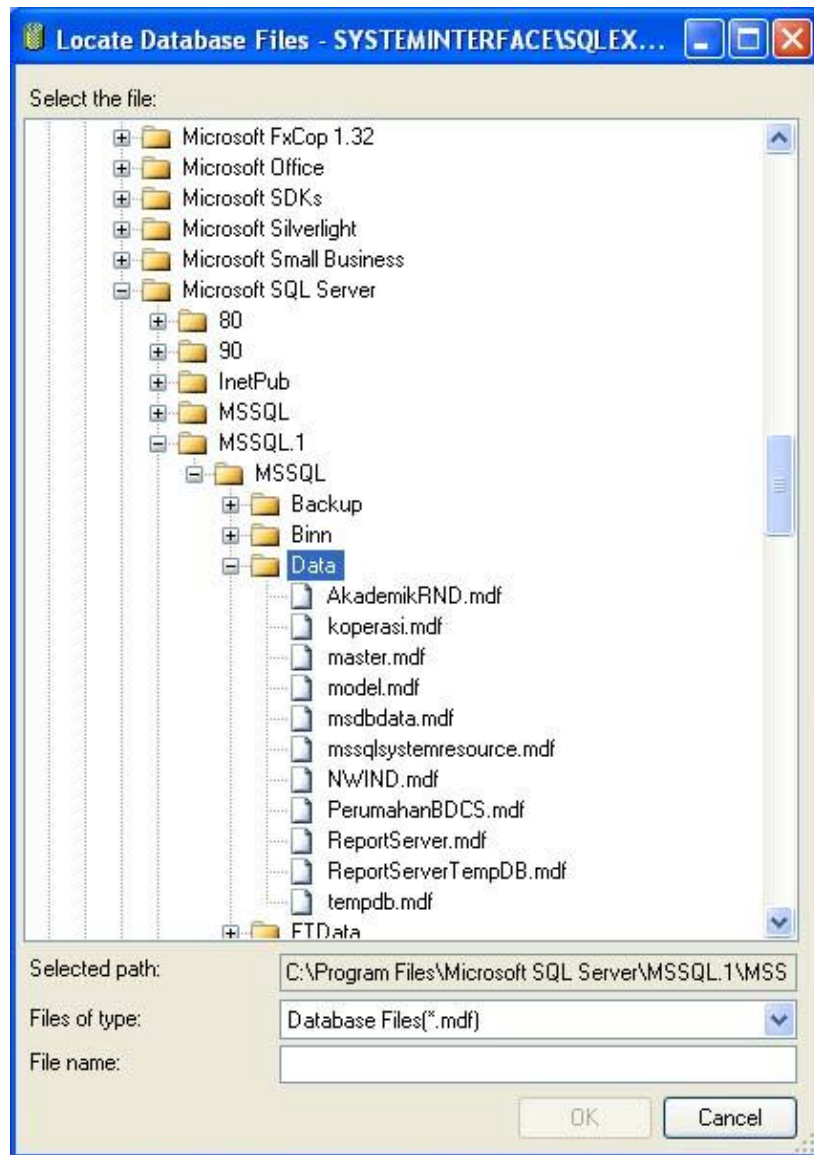
1. Copy lah file NWIND.mdf dari CD ke direktori C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data
2. Bukalah Microsoft SQL Server Management Studio Express. Setelah connect.Klik kanan Databases pada Object Explorer. Pada context menu yang muncul pilih **Attach**



3. Setelah muncul Window Attach Database klik **Add**

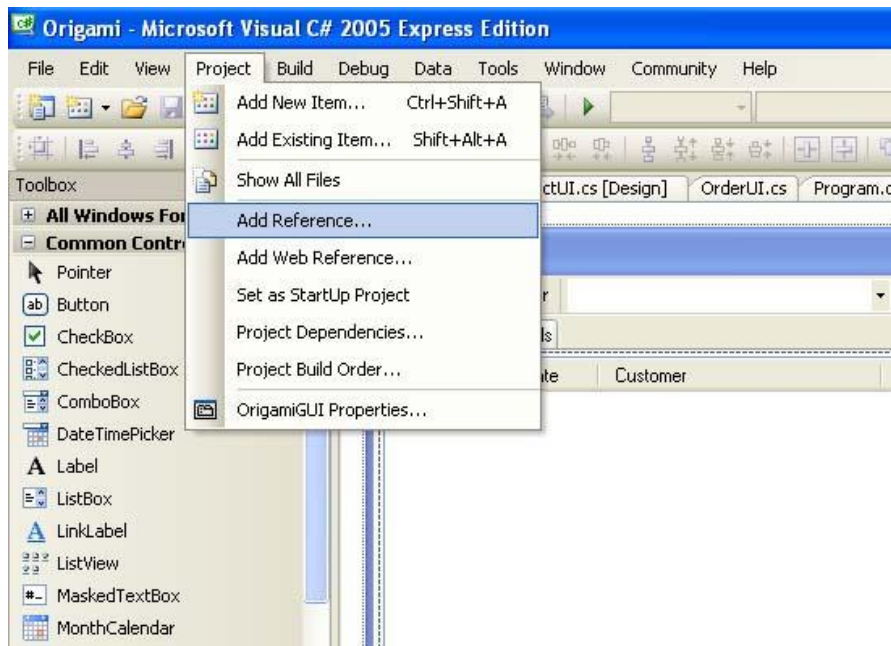


4. Carilah file mdf yang akan di attach, klik OK

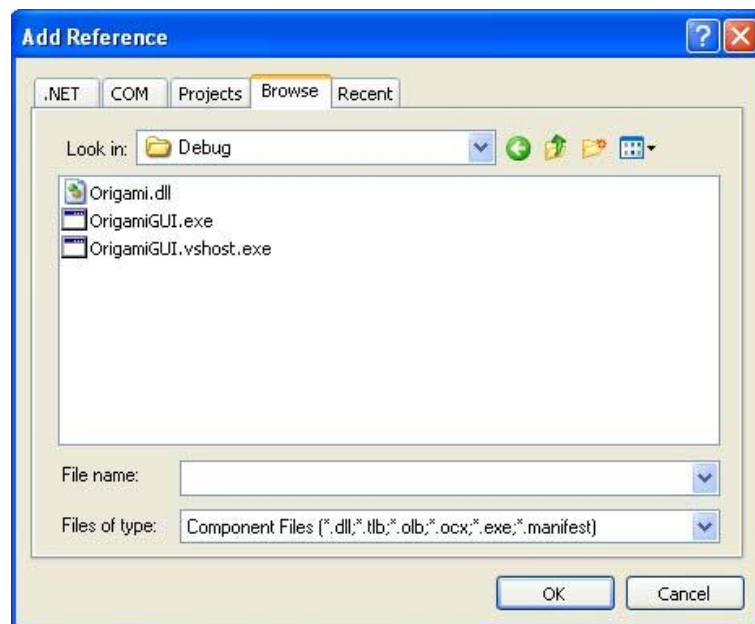


Meng-Import Framework Origami ke Project di Visual C# 2005/2008 Express Edition

1. Bukalah Visual C# 2005 Express Edition Anda
2. Mulailah Project baru, Console atau Windows Application
3. Pada menu Project klik **Add Reference**



4. Setelah muncul Window Add Reference, pilihlah tab Browse. Selanjutnya cari file Origami.dll di komputer Anda



Daftar Attribute

Berikut adalah daftar semua custom attribute yang digunakan di Origami

Attribute	Fungsi	Parameter	Contoh
[Table]	Mapping ke nama tabel di database	Name	[Table (Name="SUPPLIERS")]
[Column]	Mapping ke kolom tabel database	Name	[Column (Name="SUPPLIER_NAME")]
[Id]	ID yang menjadi primary key	Name IsIncrement IsIdentity	[Id (Name="SUPPLIER_ID")] [Id (Name="SUPPLIER_ID", IsIdentity=True)]
[Association]	Relasi antar class	EntityRef Key	[Association (EntityRef=typeof (Supplier), Key="SupplierID")]
[SQLSelect]	Query native untuk perintah SELECT	Callable Sql	[SQLSelect (Sql="SELECT * FROM Customers WHERE CustomerID=@ID")]
[SQLInsert]	Query native untuk perintah INSERT	Callable Sql	[SQLInsert (Sql="INSERT INTO Customers (name,email) VALUES (@name,@email))] [SQLInsert (Callable=true, Sql="INSERT INTO Customers (name,email) VALUES (@name,@email))]
[SQLUpdate]	Query native untuk perintah Update	Callable Sql	[SQLUpdate (Sql="UPDATE Customers SET Name=@name, Email=@email WHERE Customer_ID=@id)]
[SQLDelete]	Query native untuk perintah DELETE	Callable Sql	[SQLDelete (Sql="DELETE * FROM Customers WHERE CustomerID=@ID)]
[Dependency]	Dependency Injection	Name	[Dependency (Name="dataSource")]
[TestClass]	Menandakan sebuah class adalah Test Class		[TestClass] public class Test { }

Attribute	Fungsi	Parameter	Contoh
[TestMethod]	Menandakan method yang akan di test		<pre>[TestMethod] public void TestAdd() { }</pre>
[Database]	Menspesifikasikan DataSource lewat attribut		<pre>[Database(Name="NWIND")] public class Northwind : EntityManager { }</pre>

Konvensi dan Best Practice

Walaupun Origami dikembangkan sedinamis mungkin, ada beberapa hal yang harus dipatuhi agar Framework bekerja sebagaimana mestinya. Beberapa aturan tersebut adalah :

Konvensi :

1. Nama field yang menjadi foreign key harus sama dengan nama field primary key tabel tersebut.

PRODUCTS	SUPPLIERS
PRODUCT_ID PRODUCT_NAME * SUPPLIER_ID	* SUPPLIER_ID SUPPLIER_NAME

Foreign key SUPPLIER_ID di tabel Products namanya harus sama dengan field primary key SUPPLIER_ID di tabel Suppliers

2. Kecuali primary/foreign key, tidak boleh ada nama field yang sama antara tabel yang berelasi

PRODUCTS	SUPPLIERS
PRODUCT_ID *NAME SUPPLIER_ID	PRODUCT_ID *NAME SUPPLIER_ID

Field “NAME” pada tabel Products tidak boleh sama dengan field “NAME” yang ada di tabel Supplier. Jadi seharusnya keduanya diberi nama yang berbeda

Disarankan :

PRODUCTS	SUPPLIERS
PRODUCT_ID *PRODUCT_NAME SUPPLIER_ID	PRODUCT_ID *SUPPLIER_NAME SUPPLIER_ID

Best Practice :

Origami dirancang khusus untuk database tipe Client-Server seperti MS SQL Server, Oracle, atau MySQL. Penggunaan database desktop seperti MS Access tidak disarankan, karena beberapa fitur seperti transaction, dan stored procedure tidak dapat digunakan.

Konfigurasi Untuk Berbagai Macam DBMS

Salah satu kelebihan Origami adalah mampu mengakses database dari provider DBMS yang berbeda. Jika terjadi perubahan implementasi DBMS cukup diedit bagian konfigurasinya saja di App.config. Berikut ini adalah konfigurasi beberapa jenis DBMS populer :

SQL Server 2005

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="objects" type="Origami.Configuration.ConfigurationHandler,
      Origami" />
  </configSections>
  <objects>
    <object id="NWIND" type="Origami.Data.Provider.DataSource,Origami">
      <property name="Provider" value="System.Data.SqlClient" />
      <property name="ConnectionString" value="Data
        Source=SYSTEMINTERFACE\SQLEXPRESS;
        Initial Catalog=NWIND;Integrated Security=True" />
    </object>
  </objects>
</configuration>
```

Oracle 10g

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="objects" type="Origami.Configuration.ConfigurationHandler,
      Origami" />
  </configSections>
  <objects>
    <object id="dataSource" type="Origami.Data.Provider.DataSource,Origami">
      <property name="Provider" value="System.Data.OracleClient" />
      <property name="ConnectionString" value="Data Source=XE;user
        id=system;password=excalibur" />
    </object>
  </objects>
</configuration>
```


Daftar Pustaka

- Amelia, Tan, “*Pemrograman Database Menggunakan ADO.NET*”, Graha Ilmu, Yogyakarta, 2007.
- Andrew, Hunt, “*Pragmatic Unit Testing In C# with NUnit*”, The pragmatic Programers , LLC, USA, 2004.
- Christian, Gross, “*Foundations of Object-Oriented Programming Using .NET 2.0 Patterns*”, Apress, USA, 2006.
- Eric, Gamma, “*Design Patterns : Element of Reusable Object-Oriented Software*”, Addison Wesley, USA, 1995.
- Fowler, Martin, “*Patterns of Enterprise Application Architecture*”, Addison Wesley, USA, 2002.
- Freeman, Eric, “*Head First Design Pattern*”, O’Reilly Media, Inc., USA, 2007.
- Irwan, Djon, “*Perancangan Object Oriented Software dengan UML*”, Andi Yogyakarta, 2006.
- James, Newkirk, “*Test Driven Development in Microsoft.NET*”, Microsoft Press, USA, 2004.
- King, Galvin, “*Java Persistence With Hibernate*”, Manning publication Co., New York, 2007.
- Len, Fenster, “*Effectife Use of Microsoft Enterprise Library : Building Blocks for Creating Enterprise Application and Service*”, Addison Wesley, USA, 2006.
- Microsoft, “*Application Architecture for .NET : Designing Applications and Services*”, Microsoft Corporation, USA, 2002.
- Microsoft, “*.NET Data Access Architecture Guide*”, Microsoft Corporation, USA, 2003.
- Prasetyo, Didik Dwi, “*Pemrograman Aplikasi Database dengan Visual Basic.NET 2005 dan MS Access*”, Elex Media Komputindo.Jakarta, 2006.
- Will, Iverson, “*Hibernate : A J2EE Developer’s Guide*”, Addison Wesley, USA, 2004.
- Xin, Chen, “*Developing Application Framework in .NET*”, Apress, USA, 2004.

Biografi Penulis



Ariyanto, Pendiri dan *software architect* XERIS System Interface, sebuah micro ISV (*Independent Software Vendor*) yang mengembangkan sendiri tools untuk keperluan *software development* pada platform .NET. Beberapa tools yang telah dikembangkan yaitu : Code Weaver (code generator) dan Origami (ORM Framework).

Pada waktu senggangnya mengajar di Direktorat Program Diploma IPB program keahlian Manajemen Informatika dan Teknik Komputer, sebagai koordinator dan dosen mata kuliah Pemrograman Berorientasi Objek, Basis Data Client Server, Sistem Client Server, Sistem Operasi Komputer, Keamanan Jaringan, Dasar Pemrograman, dan Bahasa Pemrograman C/C++. Saat ini tertarik pada *Object Oriented Technology*, *Software Engineering*, dan *Design Pattern*.

Informasi lebih lanjut tentang penulis bisa didapat melalui :

E-mail : neonerdy@yahoo.com

Blogs : <http://geeks.netindonesia.net/blogs/neonerdy>