
Indonesia
.NET Developer Community
Ariyanto
.NET Data Access
Layer Framework
U N D E R C O V E R
Microsoft®
.NET

If something is hard to do, then it's not worth of doing

-HOMER SIMPSON-

Kata Pengantar

Erwin Shrodinger, salah seorang pakar dan pencetus Fisika Quantum pernah berkata “Bila Anda – sesudah lama bergelut – tidak dapat menceritakan apa yang tengah Anda lakukan pada orang lain, maka pekerjaan Anda akan sia-sia belaka”. Mengikuti nasihat beliau, saya mencoba untuk bercerita mengenai apa yang saya geluti selama ini (object oriented programming, software engineering, design pattern) melalui sebuah e-book dan framework kecil yang mudah-mudahan bermanfaat atau setidak-tidaknya memberi inspirasi untuk menuliskan dan mewujudkan ide-ide Anda.

Martin Fowler, Chief Scientist Thought Work begitu mempesona saya. Bukunya, *Pattern of Enterprise Application Architecture* merubah cara pandang saya tentang bagaimana mendesain software yang baik dan elegan. Dengan dasar buku tersebut saya membuat EntityMap, sebuah *lightweight data access layer framework* yang menerapkan repository pattern, data mapper pattern, dan fluent interface.

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.

Bogor, Mei 2009

Ariyanto

Daftar Isi

Data Access Layer Framework

Menurut Wikipedia, Data Access Layer (DAL) adalah layer dari sebuah program komputer yang menyediakan kemudahan akses bagi data yang tersimpan di persistent storage seperti relational database. Sedangkan framework adalah design reusable dari sebuah sistem atau sub sistem. Framework dapat terdiri dari kode program, library, atau bagian lainnya dari sebuah komponen software yang terpisah dari aplikasi itu sendiri. Bagian dari framework dapat di expose keluar melaui API (*Application Programming Interface*).

Data Access Layer Framework Requirements

Sebuah data access layer framework sebaiknya memiliki functional requirements sebagai berikut :

Database Independence

Database independence berarti sebuah DAL memiliki service yang sama untuk berbagai macam DBMS. Tidak peduli jenis DBMS yang digunakan. Sebuah framework DAL yang baik mampu mengakses SQL Server, Oracle, DB2, MySQL dan DBMS lainnya. Idealnya jika terjadi perubahan implementasi DBMS, tidak akan meneybabkan perubahan yang berarti pada aplikasi, cukup dengan mengedit konfigurasi data provider nya saja di XML.

Persisting Application Object Model

Memprogram menggunakan pendekatan OOP berarti memodelkan domain permasalahan menjadi objek-objek yang “hidup” di memory. Objek yang hanya ada di memory disebut transient object. Transient object dalam sebuah aplikasi biasanya perlu disimpan ke media penyimpanan seperti file atau DBMS sehingga bisa persistent dan mudah untuk diakses dikemudian hari. Sebuah DAL framework harus mampu menyimpan transient object menjadi persistent ke database atau sebaliknya mengkosntruksi transient object dari peristent data di database.

Tanggung Jawab Data Access Layer

CRUD Service

CRUD adalah kumpulan method yang bertanggung jawab terhadap manipulasi dan query data pada sebuah relational DBMS. CRUD akronim dari Create, Read, Update, dan Delete.

Query Servcie

R pada CRUD Service adalah Read, ini artinya sebuah DAL framework harus menyediakan mekanisme untuk me-retrieve persistent data dari database. Selain itu harus menyediakan mekanisme query kompleks dengan berbagai kriteria tertentu.

Transaction Management

Memanage transactional object dalam satu unit of work

Pengenalan EntityMap

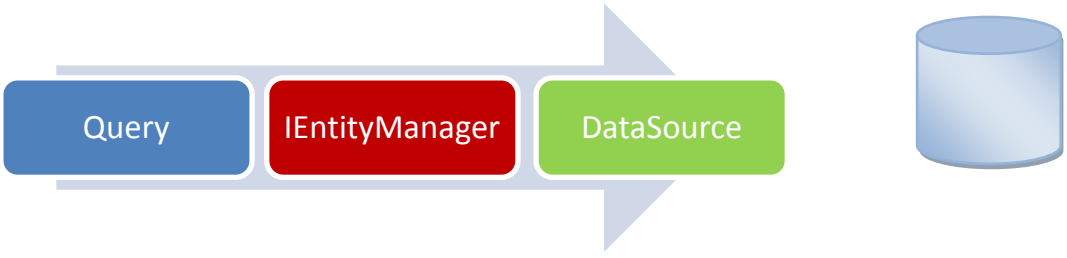
EntityMap adalah *lightweight data access layer framework* yang mengimplementasikan repository pattern, data mapper pattern (katalog pattern dari buku Martin Fowler : *Pattern of Enterprise Application Architecture*), dan fluent interface. Selain itu framework ini juga mengadopsi konsep DDD (*Domain Driven Design*) yang dipopulerkan oleh Eric Evans.

EntityMap membantu mempercepat pengembangan aplikasi Enterprise (baca : aplikasi Database) pada platform .NET tanpa mengorbankan desain arsitektur yang baik dan elegan dengan menerepkan pattern dan practice yang sudah terbungkus dengan rapi dalam sebuah framework. EntityMap dikembangkan oleh penulis sendiri dibawah lisensi GPL (*General Public Licence*) dan dapat didownload di http://code.google.com/p/entitymap/

Fitur EntityMap :

1. Menyederhanakan API (*Application Programming Interface*) pengaksesan data via ADO.NET
2. Menyediakan Data Access helper yang membungkus perintah-perintah ADO.NET untuk manipulasi dan query data seperti : ExecuteNonQuery(), ExecuteDataReader(), ExecuteDataSet(), dan ExecuteScalar()
3. Memiliki kemampuan Generic Data Access, yaitu dapat mengakses database yang beraneka ragam selama disediakan driver .NET oleh vendor DBMS.
4. Dapat memetakan record/row pada tabel ke objek
5. Mendukung stored procedure
6. Menyediakan pembungkus untuk objek Command
7. Mendukung programmatic transcation
8. Mendukung fluent interface query
9. Mendukung Dependency Injection
10. Konfigurasi Data Source dapat diletakkan di XML

Arsitektur EntitMap



Keterangan

DataSource	Menspesifikasi provider dan connection string database
IEntityManager	Interface ADO.NET Helper
Query	Mengkonstruksi query CRUD dengan gaya fluent interface

Classic ADO.NET

```
string connStr = "Data Source=XERIS\\SQLEXPRESS;"
      + "Initial Catalog=Northwind;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"]);
}

rdr.Close();
```

EntityMap Way

```
DataSource dataSource = new DataSource();

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

IEntityManager em = EntityManagerFactory.CreateInstance(dataSource);

IDataReader rdr = em.ExecuteReader("SELECT * FROM Customers");
while (rdr.Read())
{
    Console.WriteLine(rdr["CustomerId"]);
    Console.WriteLine(rdr["CompanyName"]);
}

rdr.Close();
```

EntityMap menggunakan fitur Data Mapper

```
IEntityManager em = EntityManagerFactory.CreateInstance(dataSource);
string sql="SELECT * FROM Customers";

List<Customer> custs = em.ExecuteList<Customer>(sql, new CustomerMapper());
foreach (Customer cust in custs)
{
    Console.WriteLine(cust.CustomerId);
    Console.WriteLine(cust.CompanyName);
}
```

Daftar connection string untuk database populer :

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;
MSQL	System.Data.MySqlClient	Server=myServerAddress;Database=myDataBase;Uid=myUsername;Pwd=myPassword;
PostreSQL	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	System.Data.OleDb	Provider=DB2OLEDB;Network Transpor 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 (OleDb)	System.Data.OleDb	Provider=IBMDA400;Data Source=MY_SYSTEM_NAME;User Id=myUsername;Password=myPassword;
Paradox (OleDb)	System.Data.OleDb	Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\myDb;Extended Properties=Paradox 5.x;

Untuk lebih lengkapnya silahkan kunjungi http://connectionstrings.com untuk informasi lebih lanjut.

Selain mespesifikasikan DataSource secara programatic, EntityMap juga mendukung penulisan konfigurasi data source lewat XML (disarankan menggunakan pendekatan ini), sehingga jika terjadi perubahan pada connection string atau pada implementasi DBMS cukup mengedit konfigurasinya saja, tanpa harus meng-compile ulang code nya.

Konfigurasi XML tersebut secara default disimpan di file App.config. Berikut ini cara mengakses DataSource secara programatic dan lewat konfigurasi XML

Secara programatic :

```
DataSource dataSource = new DataSource();

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

IEntityManager em = EntityManagerFactory.CreateInstance(dataSource);
```

Lewat konfigurasi XML :

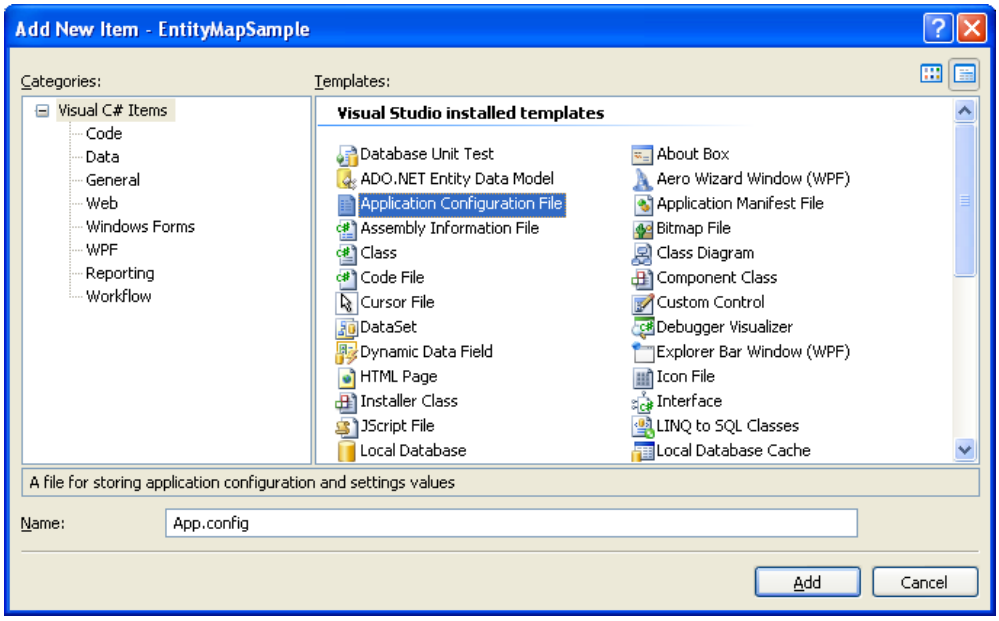
```
IEntityManager em = EntityManagerFactory.CreateInstance();
```

Class **EntityManagerFactory** memiliki dua buah constructor, jika diakses menggunakan parameter DataSource maka setting data source dilakukan secara programatic, sedangkan jika tanpa parameter secara default EntityMap akan membaca konfigurasi di file App.config.

Contoh konfigurasi XML :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
 <appSettings>
  <add key="Provider" value="System.Data.SqlClient"/>
  <add key="ConnectionString" value="Data Source=XERIS\SQLEXPRESS;
      InitialCatalog=Northwind;Integrated Security=True"/>
  </appSettings>
</configuration>
```

Untuk membuat file App.config , klik kanan Project ->Add New Item, kemudian pilih Application Configuration File.



Data Access Helper

Salah satu feature EntityMap adalah Data Access Helper. Berikut ini method-method yang memudahkan pengaksesan ADO.NET :

Method	Keterangan
ExecuteReader()	Method yang digunakan untuk mengeksekusi perintah SQL SELECT Parameter : Sql Contoh : ExecuteReader("SELECT * FROM Employees") Return value : IDataReader
ExecuteNonQuery()	Method yang digunakan untuk mengeksekusi perintah SQL INSERT, UPDATE, dan DELETE Parameter : Sql Contoh : ExecuteNonQuery("DELETE FROM Employees WHERE EmployeeID=1") Return value : int
ExecuteDataSet()	Method yang digunakan untuk mengambil row dari da (SQL SELECT) untuk selanjutnya disimpan ke DataSe Parameter : Sql Contoh : ExecuteReader("SELECT * FROM Employees") 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>)

Buka Koneksi

```
DataSource dataSource = new DataSource();

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

IEntityManager em = EntityManagerFactory.CreateInstance(dataSource);
```

ExecuteReader()

```
IDataReader rdr=em.ExecuteReader("SELECT * FROM Customers");
while (rdr.Read())
{
    Console.WriteLine(rdr["CustomerId"]);
    Console.WriteLine(rdr["CompanyName"]);
}
```

ExecuteNonQuery()

```
string sql="INSERT INTO Customers (CustomerId,CompanyName) VALUES "
         + "('MSFT','Microsoft')";

em.ExecuteNonQuery(sql);
```

ExecuteDataSet()

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

foreach (TableRow row in ds.Tables[0].Rows)
{
    Console.WriteLine(row["CustomerId"]);
    Console.WriteLine(row["CompanyName"]);
}
```

ExecuteScalar()

```
object num=em.ExecuteScalar("SELECT COUNT(*) FROM Customers");
```

ExecuteObject()

```
string sql="SELECT * FROM Customers ";

Customer cust = em.ExecuteObject<Customer>(sql, new Customer Mapper());
Console.WriteLine(cust.CustomerId);
Console.WriteLine(cust.CompanyName);
```

ExecuteList()

```
string sql="SELECT * FROM Customers ";

List<Customer> custs = em.ExecuteList< Customer >(sql, new CustomerMapper());
foreach (Customer cust in custs)
{
    Console.WriteLine(cust.CustomerId);
    Console.WriteLine(cust.CompanyName);
}
```

Fluent Query

Fluent Query didasarkan oleh konsep Fluent Interface yang dikemukakan oleh Martin Fowler. Fluent Interface adalah teknik dalam merancang method yang bersifat chainable (berantai), sehingga lebih alami (baca : mudah) untuk diimplementasikan.

EntityMap menyediakan satu class yang sangat powerfull bernama **Query**. Class ini mengenkapsulasi perintah-perintah SQL CRUD (Create, Read, Update, Delete) dan method-method ADO.NET seperti ExecuteReader(), ExecuteNonQuery(), ExecuteDataSet(), dan ExecuteScalar(). Class tersebut menyediakan pula method ExecuteObject() dan ExecuteList() yang merupakan fitur ORM (*Object Relational Mapping*) dari EntityMap.

```
Query q = new Query().From("Customers").Where("CustomerId").Equal("ALFKI");
Console.WriteLine(q.GetSql());
```

Query diatas akan menghasilkan perintah SQL berikut : SELECT * FROM Customers WHERE CustomerId='ALFKI'. Menariknya, dalam satu chained method, kita bisa memanggil method ExecuteReader(), atau method lainnya. Contoh :

ExecuteReader()

```
IDataReader rdr = new Query().From("Customers ")
      .Where("CustomerId").Equal("ALFKI").ExecuteReader();

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

ExecuteScalar()

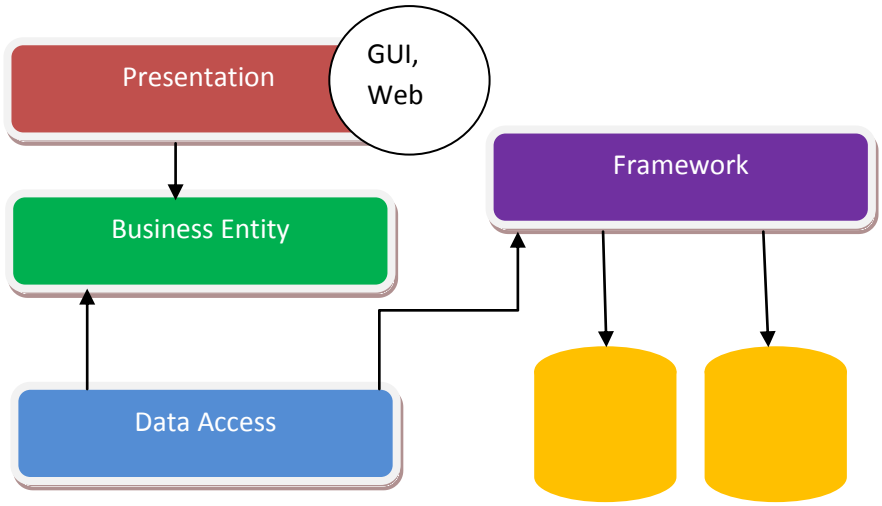
```
int result = new Query().Select("Max(Qty)").From("Products").ExecuteScalar();
```

ExecuteNonQuery()

```
string[] fields = {"CustomerId","CompanyName"};
object[] values = {"MSFT","Microsoft"};

int result = new Query().Select(fields).From("Customers").Insert(values)
      .ExecuteNonQuery();

string[] fields = {"CompanyName"};
object[] values = {"Microsoft Corporation"};

int result = new Query().Select(fields)
      .From("Customers").Update(values).Where("CustomerId").Equal("MSFT")
      .ExecuteNonQuery();
```

Layered Architecture

Layered architecture membagi sistem menjadi beberapa group, dimana masing-masing group menangani fungsi yang spesifik. Dalam menghadapi kompleksitas ada baiknya "memecah belah" aplikasi menjadi beberapa layer. Pemisahan ini bisa jadi hanya secara logik (software), tidak harus berkorespondensi secara fisik (hardware).



Pada pendekatan layered architecture, biasanya aplikasi ini dipecah menjadi 3 layer yaitu : Presentation, Business Entity dan Data Access layer.

Presentation layer bertanggung jawab dengan tampilan (user interface), **Business Entity** dengan logika business/domain permasalahan dan **Data Access** bertanggung jawab untuk memanipulasi tabel-tabel di basis datanya. Dengan pemisahan ini aplikasi tidak tergantung dengan user apa interface nya (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.

Berikut ini perbedaan pendekatan 1 layer dengan 3 layer :

Classic (1 layer)	3 layer
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 reusable (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

Business Entity

Business Entity adalah objek-objek yang terdapat dalam sebuah problem domain/domain pemasalahan disebuah sistem tertentu. Business entity ini dihasilkan dari proses analisis sebuah requirement software. Biasanya business entity disebut juga domain, model, atau business logic.

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
Business Entity Object	Entity class yang merepresentasikan business entity. Class ini berisi enkapsulasi field, property, dan method.
CRUD Business Object	Entity class yang memiliki kemampuan CRUD (Create, Read, Update, Delete)

Representasi paling dinamis dari tabel diatas adalah dengan menggunakan Business Entity Object yang diimplementasikan dalam bentuk Entity Class. Bisanya entity class cukup berisi field dan property saja.

Contoh :

```
namespace Northwind.Model
{
    public class Customer
    {
        private string customerId;
        private string companyName;
        private string contactName;
        private string address;
        private string phone;
        private string fax;

        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; }
        }

        public string Fax
        {
            get { return fax; }
            set { phone = value; }
        }

    }
}
```

Jika menggunakan .NET Framework 3.5, penulisan property dapat disingkat sebagai berikut :

```
namespace Northwind.Model
{
    public class Customer
    {
        public string CustomerId { get; set; }

        public string CompanyName { get; set; }

        public string ContactName { get; set; }

        public string Address { get; set; }

        public string Phone { get; set; }

        public string Fax { get; set; }

    }
}
```

Data Access Via Repository Pattern

Menurut Martin Fowler ,

"*A Repository mediates between the domain and data mapping layers, acting like an in-memory domain object collection. Client objects construct query specifications declaratively and submit them to Repository for satisfaction. Objects can be added to and removed from the Repository, as they can from a simple collection of objects, and the mapping code encapsulated by the Repository will carry out the appropriate operations behind the scenes. Conceptually, a Repository encapsulates the set of objects persisted in a data store and the operations performed over them, providing a more object-oriented view of the persistence layer. Repository also supports the objective of achieving a clean separation and one-way dependency between the domain and data mapping layers.*"

Repository mengenkapsulasi method-method untuk manipulasi dan query data dalam sebuah class yang berkorespondensi dengan model/business entity. Sebuah repository biasanya memiliki method **CRUD** (Create, Read, Update, Delete) seperti : FindById(), FindAll(), Save(), Update(), dan Delete() seperti yang didefenisikan di interface IRepository<T> generic sebagai berikut :

```
public interface IRepository<T>
{
   T FindById(object id);
   List<T> FindAll();
   int Save(T entity);
   int Update(T entity);
   int Delete(T entity);
}
```

Defenisikan interface masing-masing class repository yang mewarisi generic interface IRepository<T>

```
namespace Northwind.Repository
{
    public interface ICustomerRepository : IRepository<Customer>
    {
    }
}
```

dan selanjutnya buatkan implementasi interface nya

```
namespace Northwind.Repository.Implementation
{
    public class CustomerRepository : ICustomerRepository
    {
        private IEntityManager em;
        private string tableName = "Customers";

        public CustomerRepository(IEntityManager em)
        {
            this.em = em;
        }
```

```
        public User FindById(object id)
        {
        }

        public List<Customer> FindAll()
        {
        }

        public int Save(Customer cust)
        {
        }

        public int Update(Customer cust)
        {
        }

        public int Delete(Customer cust)
        {
        }
    }
}
```

Pada masing-masing constructor class repository memiliki ketergantungan terhadap objek dataSource. Objek ini yang bertanggung jawab terhadap implementasi provider database dan connection string nya. Constructor tersebut memiliki dependency terhadap interface IEntityManager yang akan di inject melalui mekanisme dependency injection di class ServiceRegistry.

IEntityManager sendiri adalah interface EntityMap yang berisi method-method data access helper. Berikut ini isi dari interface tersebut :

```
public interface IEntityManager
{
    int ExecuteNonQuery(string sql);
    int ExecuteNonQuery(string sql, Transaction tx);
    int ExecuteNonQuery(DbCommandWrapper dbCommandWrapper);
    int ExecuteNonQuery(DbCommandWrapper dbCommandWrapper, Transaction tx);
    IDataReader ExecuteReader(string sql);
    IDataReader ExecuteReader(DbCommandWrapper dbCommandWrapper);
    DataSet ExecuteDataSet(string sql);
    DataSet ExecuteDataSet(DbCommandWrapper dbCommandWrapper);
    object ExecuteScalar(string sql);
    object ExecuteScalar(DbCommandWrapper dbCommandWrapper);
    T ExecuteObject<T>(string sql, IDataMapper<T> rowMapper);
    T ExecuteObject<T>(DbCommandWrapper dbCommandWrapper,
         IDataMapper<T> dataMapper);
    List<T> ExecuteList<T>(string sql, IDataMapper<T> rowMapper);
    List<T> ExecuteList<T>(DbCommandWrapper dbCommandWrapper,
        IDataMapper<T> dataMapper);
    DbCommandWrapper CreateCommand();
    DbCommandWrapper CreateCommand(CommandWrapperType cmdWrapperType,
        string query);
    Transaction BeginTransaction();}
}
```

Fluent Interface Query

Implementasi masing-masing method CRUD menggunakan pendekatan fluent interface. Seperti yang telah dijelaskan diatas, fluent interface adalah suatu cara mendesign sebuah *chainable method* yang mudah dibaca. Contoh :

```
public User FindById(object id)
{
   Query q = new Query().From(tableName).Where("CustomerId").Equal(id);

   return em.ExecuteObject<Customer>(q.GetSql(), new CustomerMapper());
}
```

Contoh Lain :

Save()

```
public int Save(Customer cust)
{
    string[] fields = {"CustomerId", "CompanyName", "ContactName",
                       "Address", "Phone" };
    object[] values = {cust.CustomerId, cust.CompanyName, cust.ContactName,
                       cust.Address, cust.Phone };

    Query q = new Query().Select(fields).From(tableName).Insert(values);

    return em.ExecuteNonQuery(q.GetSql());
}
```

Update()

```
public int Update(Customer cust)
{
   string[] fields = {"CompanyName", "ContactName", "Address", "Phone"};
   object[] values = {cust.CompanyName, cust.ContactName, cust.Address,
                      cust. Phone};

   Query q = new Query().Select(fields).From(tableName).Update(values)
             .Where("CustomerId").Equal(cust.CustomerId);

   return em.ExecuteNonQuery(q.GetSql());
}
```

Delete()

```
public int Delete(Customer cust)
{
   Query q = new Query().From(tableName).Delete()
             .Where("CustomerId").Equal(user.UserId);

   return em.ExecuteNonQuery(q.GetSql());
}
```

Data Mapper

Data Mapper adalah pattern yang digunakan untuk melakukan mapping row ke object. Mapping dilakukan karena antara dunia basis data relasional dan object oriented memiliki struktur penyimpanan yang berbeda. Basis data relational mengembalikan kumpulan record (*set of row*), bukan nya kumpulan objek (*collection of object*). Masalah ini biasanya disebut "*Impedence mismatch*". Untuk mengatasi ini data access layer framework harus mampu mengubah kumpulan baris (row) di tabel menjadi kumpulan objek. Fitur Data Mapper merupakan salah satu bagian kecil dari sebuah ORM (*Object Relational Mapping*) framework.

EntityMap tidak memposisikan diri sebagai ORM framework, namun bisa menerapkan salah satu fitur nya, yaitu Data Mapper. Perbedaanya, mapping di EntityMap dilakukan secara programatic (*hand code*), sedangkan pada ORM framework umumnya dilakukan secara otomatis dengan menggunakan metadata baik berupa custom attribute atau XML. Contoh ORM Framework populer adalah NHibernate, dan Entity Framework.

Dengan menggunakan data mapper, cara menampilkan data akan sesuai dengan gaya object oriented. Perhatikan perbedaan nya :

Tanpa mapping (menggunakan ExecuteReader)

```
IDataReader rdr=em.ExecuteReader("SELECT * FROM Customers");

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

Dengan mapping (menggunakan ExecuteList)

```
List<Customer> custs = em.ExecuteList<Customer>(sql,new Customer Mapper());
foreach (Customer cust in custs)
{
    Console.WriteLine(cust.CustomerId);
    Console.WriteLine(cust.CompanyName);
}
```

Method yang mengimplementasikan data mapper adalah ExecuteObject(), dan ExecuteList(). ExecuteObject() digunakan jika yang dikembalikan adalah single object (satu row), sedangkan ExecuteList() mengembalikan object collection (kumpulan row).

Mapping dilakukan disebuah class tersediri yang mengimplementasikan interface IDataMapper<T>

```
public class CustomerMapper : IDataMapper<Customer>
{
    public Customer Map(IDataReader rdr)
    {
        Customer cust = new Customer();
```

```
            cust.CustomerId = rdr["CustomerId"].ToString();
            cust.CompanyName = rdr["CompanyName"].ToString();
            cust.ContactName = rdr["ContactName"].ToString();
            cust.Address = rdr["Address"].ToString();
            cust.Phone = rdr["Phone"].ToString();

            return cust;
        }
}
```

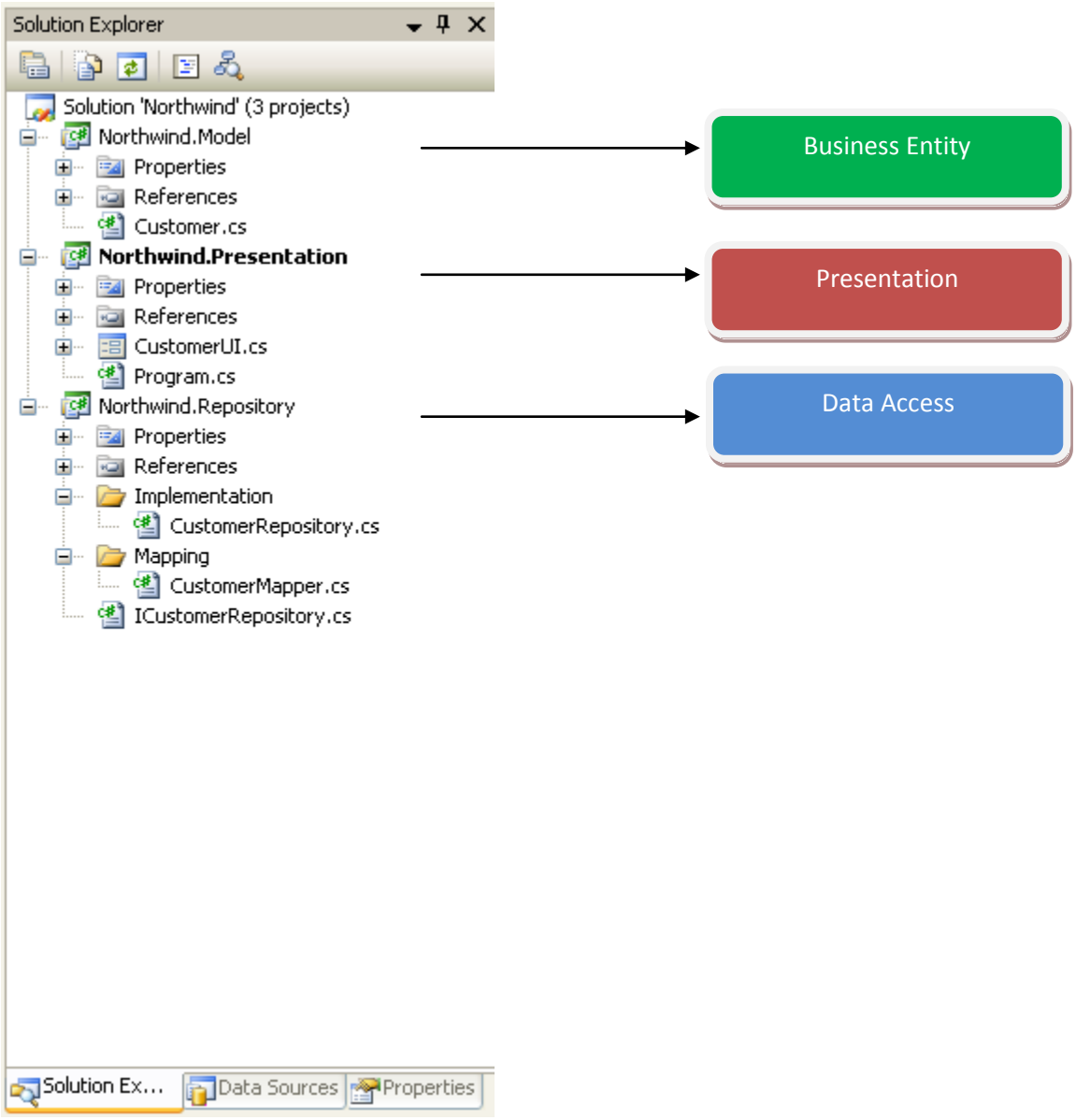
Jika salah satu field dalam tabel yang dimapping memperbolehkan NULL value, maka lakukan mapping seperti berikut :

```
namespace Northwind.Repository.Mapping
{
    public class CustomerMapper : IDataMapper<Customer>
    {
        public Customer Map(IDataReader rdr)
        {
            Customer customer = new Customer();

            customer.CustomerId = rdr["CustomerId"] is DBNull ?
                string.Empty : (string)rdr["CustomerId"];
            customer.CompanyName = rdr["CompanyName"] is DBNull ?
                string.Empty : (string)rdr["CompanyName"];
            customer.ContactName = rdr["ContactName"] is DBNull ?
                string.Empty : (string)rdr["ContactName"];
            customer.Address = rdr["Address"] is DBNull ?
                string.Empty : (string)rdr["Address"];
            customer.Phone = rdr["Phone"] is DBNull ?
                string.Empty : (string)rdr["Phone"];

            return customer;
        }
    }
}
```

Selanjutnya class CustomerMapper() dipanggil di method **Read** class CustomerRepository

```
public Customer FindById(object id)
{
   Query q = new Query().From(tableName).Where("CustomerId").Equal(id);

   return em.ExecuteObject<Customer>(q.GetSql(), new CustomerMapper ());
}

public List<Customer> FindAll()
{
   Query q = new Query().From(tableName);
   return em.ExecuteList<Customer>(q.GetSql(), new CustomerMapper());
}
```

Project structure



Disisi client class-class Repository diakses lewat interface nya. Pemrograman melaui interface ini menjadikan client tidak tergantung dengan layer data access, sehingga jika terjadi perubahan pada metode akses data di repsository, layer presentation sama sekali tidak perlu diubah. Desain struktur seperti ini disebut *lousley coupled*.

Relationship

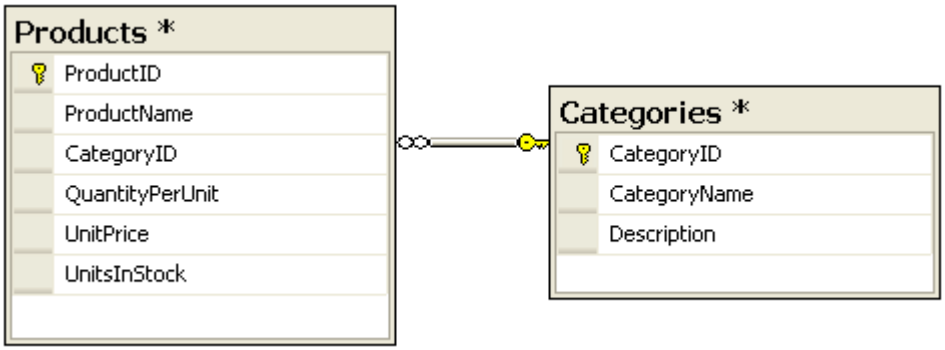
Relational DBMS memiliki tabel-tabel yang saling beralasi. Relational DBMS yang berbasis aljabar relasi memiliki tipe relasi yang disebut Asosiasi. Hubungan ini terbagi menjadi 4 jenis, yaitu : hubungan one-to-one, one-to-many, many-to-one dan many-to-many. Sebelum merelasikan tabel harus dilakukan proses normalisasi terlebih dahulu untuk menjamin konsistensi, integritas data dan meminimalisir redudansi.

Tidak seperti Relational DBMS yang hanya memiliki 1 jenis relasi, class memiliki 5 macam jenis relasi yang mungkin yaitu : Asosiasi, Agregasi, Generalisasi, Depedensi, dan Realisasi .

Berikut ini penjelasan masing-masing relasi :

Jenis Relasi	Keterangan
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
Aggregasi	Relasi antara "keseluruhan" dengan "bagian"
Generalisasi	Relasi pewarisan antara dua class. Super class (class induk) mewarisi seluruh property dan method ke sub class. Sub class bisa jadi memiliki property dan method spesifik yang tidak dimiliki oleh super class nya Generalisasi dalam OOP disebut juga Inheritance
Depedency	Relasi yang menunjukkan sebuah class mengacu ke class lainnya. Depedency adalah sebuah relasi yang bersifat tighty-coupled. Perubahan pada class yang diacu bisa jadi menyebabkan perubahan di class pengguna
Realisasi	Relasi antara interface dengan class yang menjadi implemetasi dari interface tersebut. Dengan menggunakan inteface, struktur kode kita menjadi loosely-coupled, karena memungkinkan secara dinamis mengganti implementasi

Contoh relasi tabel Products dengan tabel Categories di Database Northwind



Berikut ini adalah representasi Business Entity dari kedua tabel

Class Category

```
using System.Collections.Generic;

namespace Northwind.Model
{
    public class Category
    {
        public int CategoryId { get; set; }

        public string CategoryName { get; set; }

        public List<Product> Products { get; set; }
    }
}
```

Hubungan antara tabel Categories dan Product adalah one-to-many, satu kategori mengandung banyak produk.Representasi one-to-many di class menggunakan asosiasi dalam bentuk collection. Implementasi collection di C# biasanya menggunakan List<T> generic, dimana T bisa diisi oleh object apa saja (dalam hal ini T diisi oleh objek Category). Sebaliknya, di class Product berarti asosiasi nya many-to-one.

Class Product

```
namespace Northwind.Model
{
    public class Product
    {
        public int ProductId { get; set; }

        public string ProductName { get; set; }

        public string CategoryId { get; set; }

        public Category Category { get; set; }

        public string QuantityPerUnit { get; set; }

        public decimal UnitPrice { get; set; }

        public int UnitsInStock { get; set; }

    }
}
```

Data mapping untuk kedua class tersebut adalah sebagai berikut :

```
public class CategoryMapper : IDataMapper<Category>
{
    public Category Map(IDataReader rdr)
    {
         Category category = new Category();
```

```
            category.CategoryId = rdr["CategoryId"] is DBNull ?
                  0 : (int)rdr["CategoryId"];
            category.CategoryName = rdr["CategoryName"] is DBNull ?
                  string.Empty : (string)rdr["CategoryName"];

            return category;
      }
}


public class ProductMapper : IDataMapper<Product>
{
      public Product Map(IDataReader rdr)
      {
            Product product = new Product();

            product.ProductId = rdr["ProductId"] is DBNull ?
                0 : (int)rdr["ProductId"];
            product.ProductName = rdr["ProductName"] is DBNull ?
                string.Empty : (string)rdr["ProductName"];
            product.CategoryId = rdr["CategoryId"] is DBNull ?
                0 : (int)rdr["CategoryId"];

            CategoryMapper categoryMapper=new CategoryMapper();
            product.Category = categoryMapper.Map(rdr);

            product.QuantityPerUnit = rdr["QuantityPerUnit"] is DBNull ?
               string.Empty : (string)rdr["QuantityPerUnit"];
            product.UnitPrice = rdr["UnitPrice"] is DBNull ?
               0 : (decimal)rdr["UnitPrice"];
            product.UnitsInStock = rdr["UnitsInStock"] is DBNull ?
               0 : (int)rdr["UnitsInStock"];

            return product;
       }
 }
```

Jika tabel yang dimap memiliki relasi (seperti tabel produk), terlebih dahulu tabel tersebut di map di class terpisah (class CategoryMap), selanjutnya di ProductMap class CategoryMap ini dinstantiasi

```
CategoryMapper categoryMapper=new CategoryMapper();
product.Category = categoryMapper.Map(rdr);
```

Class ProductRepository

```
public class ProductRepository : IProductRepository
{
    private IEntityManager em;
    private string tableName = "Products";

    public ProductRepository(IEntityManager em)
    {
        this.em = em;
    }

    public Product FindById(object id)
    {
       Query q=new Query().From(tableName)
          .InnerJoin("Categories","CategoryId","CategoryId")
          .Where("ProductId").Equal(id);

       return em.ExecuteObject<Product>(q.GetSql(),new ProductMapper());
    }

    public List<Product> FindAll()
    {
       Query q = new Query().From(tableName)
          .InnerJoin("Categories", "CategoryId", "CategoryId")

       return em.ExecuteList<Product>(q.GetSql(), new ProductMapper());
    }

    public int Save(Product entity){
    }

    public int Update(Product entity){
    }

    public int Delete(Product entity){
    }

 }
```

Fluent interface query pada method FindById() dan FindAll() menggunakan statement InnerJoin untuk merelasikan tabel Products dengan tabel Categories. Relasi dilakukan melalui foreign key CategoryId ditabel Products. Fluent interface query pada method FindAll() akan mengenerate perintah Sql berikut :

```
SELECT * FROM Products INNER JOIN Categories ON Products.CategoryId =
Categories.CategoryId
```

One To Many Relationship

Seperti yang telah dijelaskan sebelumnya, hubungan antara tabel Categories dan Products adalah one-to-many. Representasi one-to-many di entity class menggunakan asosiasi dalam bentuk collection seperti yang terlihat dibawah ini :

```
using System.Collections.Generic;

namespace Northwind.Model
{
    public class Category
    {
        public int CategoryId { get; set; }

        public string CategoryName { get; set; }

        public List<Product> Products { get; set; }
    }
}
```

Sedangkan class CategoryRepository sebagai berikut :

```
public class CategoryRepository : ICategoryRepository
{
    private IEntityManager em;
    private string tableName = "Categories";
    private IProductRepository productRepository;

    public CategoryRepository(IEntityManager em)
    {
        this.em = em;
        productRepository = RepositoryFactory
             .GetObject<IProductRepository>();
    }

    public Category FindById(object id)
    {
        Query q = new Query().From(tableName).Where("CategoryId").Equal(id);

        Category category=em.ExecuteObject<Category>(q.GetSql(),
             new CategoryMapper());

        category.Products = productRepository.FindByCategoryId(id);

        return c;
    }

    public List<Category> FindAll()
    {
        Query q = new Query().From(tableName);
        List<Category> categories=em.ExecuteList<Category>(q.GetSql(),
              new CategoryMapper());
```

```
            foreach (Category category in categories)
            {
                category.Products = productRepository
                          .FindByCategoryId(c.CategoryId);
             }
            return categories;
        }

        public int Save(Category category) {
        }

        public int Update(Category category){
        }

        public int Delete(Category category) {
        }
}
```

Dari contoh diatas terlihat jelas untuk mengimplementasikan hubungan one to many class CategoryRepository perlu menginstantiate class yang berelasi yaitu ProductRepository. Perhatikan baik-baik method FindById() dan FindAll() milik class CategoryRepository. Object productRepository memanggil method FindByCategoryId().

Method ini bertugas untuk memanggil semua produk berdasarkan category id tertentu dan memiliki return value object collection (dalam hal ini yang dikembalikan adalah List<Product>) yang kemudian di inject ke property Products di object category.

Selanjutnya di interface IProductRepository tambahkan method FindByCategoryId()

```
public interface IProductRepository : IRepository<Product>
{
    List<Product> FindByCategoryId(object id);
}
```

Jangan lupa buatkan implementasi nya di class ProductRepository

```
public List<Product> FindByCategoryId(object id)
{
     Query q = new Query().From(tableName)
          .InnerJoin("Categories", "CategoryId", "CategoryId")
          .Where("Categories.CategoryId").Equal(id);

     return em.ExecuteList<Product>(q.GetSql(),
          new ProductMapper());
}
```

Lazy Loading

Tujuan dari lazy loading adalah untuk mengoptimasi memory dengan memprioritaskan komponen apa saja yang di load ke memory pada saat program berjalan. Lazy loading erat sekali hubungannya dengan ORM (*Object Relational Mapping*). Dalam dunia ORM lazy load adalah menunda me-load data dari database, membuat object atau object collection sampai ia dibutuhkan. Contoh ketika me load object Category maka objek-objek yang berelasi dengan nya akan ikut di load juga. Jika ada banyak sekali objek yang saling berelasi akan menimbulkan *performance hurt*, padahal belum tentu kita ingin menggunakan objek yang terhubung tersebut.

Menurut Martin Fowler, ada 4 pendekatan yang bisa ditempuh untuk menerapkan lazy load, yaitu : lazy initialization, virtual proxy, value holder, dan ghost. Berikut ini contoh bagaimana menggunakan pendekatan virtual proxy.

```
public class Category
{
     public int CategoryId { get; set; }

     public string CategoryName { get; set; }

     public virtual List<Product> Products { get; set; }
}
```

Virtual proxy mengharuskan property objek yang berelasi di set menjadi virtual. Tujuannya agar bisa di override di proxy class nya.

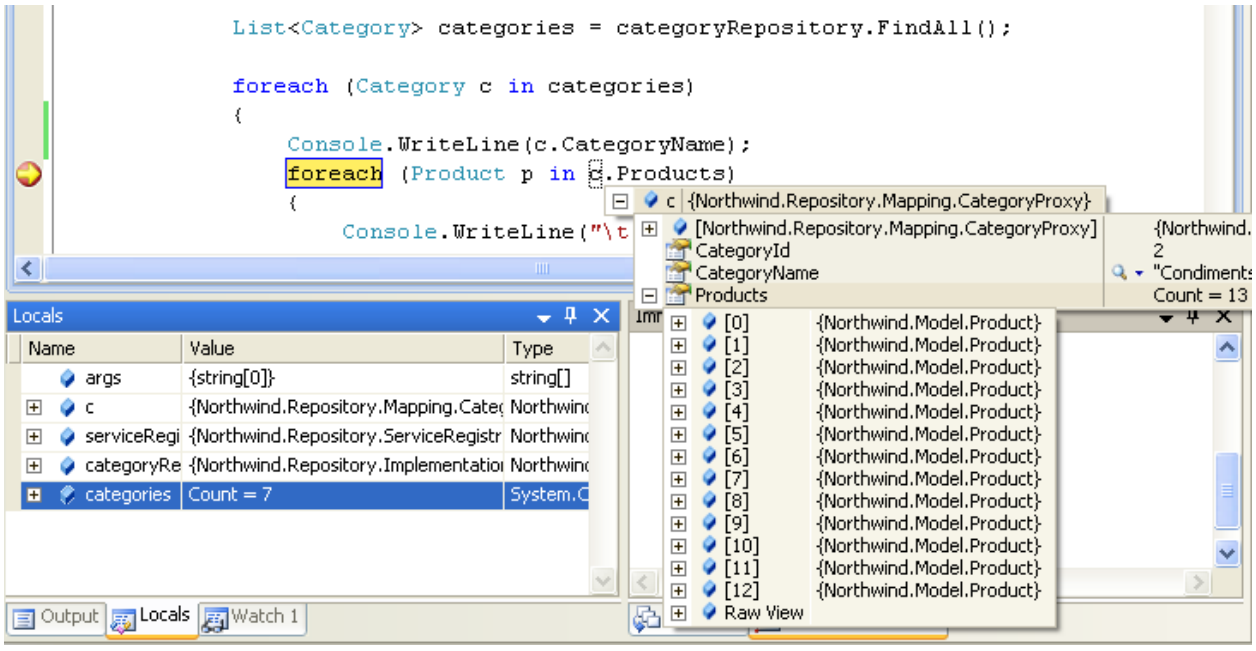
```
public class CategoryProxy : Category
{
     private bool productLoaded = false;

     public override List<Product> Products
     {
          get
          {
               List<Product> products;
               if (!productLoaded)
               {
                    IProductRepository productRepository = RepositoryFactory
                           .GetObject<IProductRepository>();
                    products = productRepository
                          .FindByCategoryId(this.CategoryId);
                    base.Products = products;

                    productLoaded = true;

               }
               else
               {
                    products=base.Products;
               }
```

```
                return products;
            }
             set
             {
                 base.Products = value;
                 productLoaded = true;
             }
        }
    }
```

Bagian yang perlu di refactoring lagi adalah CategoryMapper. Gantilah class Category menjadi CategoryProxy

```
public class CategoryMapper : IDataMapper<Category>
{
     public Category Map(IDataReader rdr)
     {
          Category category = new CategoryProxy();

          category.CategoryId = rdr["CategoryId"] is DBNull ?
              0 : (int)rdr["CategoryId"];
          category.CategoryName = rdr["CategoryName"] is DBNull ?
              string.Empty : (string)rdr["CategoryName"];

          return category;

     }
}
```

Hasil Debug



Pada gambar diatas jelas bahwa property Products pada objek c (category) baru diisi pada saat dipanggil/dibutuhkan. Dengan demikian penggunaan virtual proxy ini dapat meningkatkan performance.

Service Registry

Semua class repository beserta dependency nya di register pada sebuah class yang bernama ServiceRegistry. Dependency object class-class repository bisa lebih dari satu, misalnya selain objek em (IEntityManager) bisa juga terdapat sebuah object logger. Semua constructor dependency ini di resolve menggunakan constructor injection.

```
public class ServiceRegistry : IRegistry ,IDisposable
{
    private IEntityManager em;

    public void Configure()
    {
       string provider = ConfigurationManager.AppSettings["Provider"];
       string connStr = ConfigurationManager.AppSettings["ConnectionString"];

       DataSource dataSource=new DataSource(provider,connStr);
       em = EntityManagerFactory.CreateInstance(dataSource);

       object[] dependency = { em };

       RepositoryFactory.RegisterObject<ICategoryRepository,
            CategoryRepository>(dependency);

       RepositoryFactory.RegisterObject<IProductRepository,
            ProductRepository>(dependency);
    }

    public void Dispose()
    {
        em.Dispose();
    }
}
```

Dependency objek DataSource, value dari provider dan connectionString nya diambil dari file konfigurasi XML App.config.

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

Cara mengakses ProductRepository

```
//panggil service registry untuk meregister semua reposiotory
//berikut dependency nya

ServiceRegistry serviceRegistry = new ServiceRegistry();
RepositoryFactory.AddRegistry(serviceRegistry);

//instiantiate repository class melalui interface
//yang di register di service registry.

//proses instantiation dilakukan oleh RepositoryFactory

IProductRepository productRepository = RepositoryFactory
    .GetObject<IProductRepository>();

//panggil method FindAll() masukan ke List<T>

List<Product> products = productRepository.FindAll();

foreach (Product p in products)
{
     Console.WriteLine(p.ProductName);
     Console.WriteLine(p.Category.CategoryName);
}

//dispose object aktif jika selesai

serviceRegistry.Dispose();

Console.ReadLine();
```

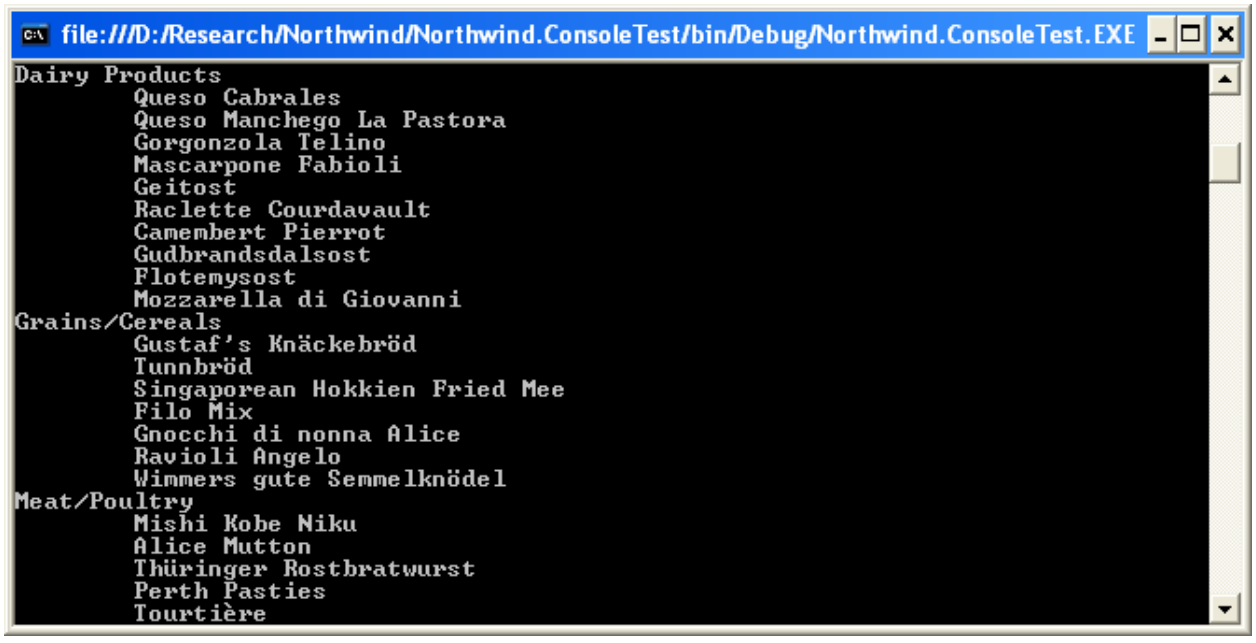
Berikut ini contoh Jika object dependency lebih dari satu

```
public class ServiceRegistry : IRegistry ,IDisposable
{
    private IEntityManager em;

    public void Configure()
    {
       string provider = ConfigurationManager.AppSettings["Provider"];
       string connStr = ConfigurationManager.AppSettings["ConnectionString"];

       DataSource dataSource=new DataSource(provider,connStr);
       em = EntityManagerFactory.CreateInstance(dataSource);

       ILogger logger = LoggerFactory.CreateFileLogger("c:\\log.txt");

       object[] dependency = { em,logger };

       //register semua repository class

     }
}
```

Mengakses one to many relationship

```
ServiceRegistry serviceRegistry = new ServiceRegistry();
RepositoryFactory.AddRegistry(serviceRegistry);

ICategoryRepository categoryRepository = RepositoryFactory
    .GetObject< ICategoryRepository >();

List<Category> categories = categoryRepository.FindAll();

foreach (Category c in categories)
{
     Console.WriteLine(c.CategoryName);

     foreach (Product p in c.Products)
     {
         Console.WriteLine("\t" + p.ProductName);
     }
}

serviceRegistry.Dispose();

Console.ReadLine();
```

Screen shoot



Command Wrapper

Selain menggunakan fluent query, EntityMap masih mendukung pemanggilan syntax SQL biasa atau stored procedure. EntityMap menyediakan wrapper (pembungkus) bagi objek Command di ADO.NET sehingga pengaksesannya bisa lebih mudah.

```
IEntityManager em = EntityManagerFactory.CreateInstance(dataSource);

string sql = "SELECT * FROM Employees WHERE EmployeeId=@EmployeeId";
DbCommandWrapper cmd = em.CreateCommand(CommandWrapperType.Text, sql);
cmd.SetParameter("@EmployeeId",1);

IDataReader rdr=em.ExecuteReader(cmd);

while (rdr.Read())
{
   Console.WriteLine(rdr["FirstName"]);
}
```

Contoh lain

```
string sql = "INSERT INTO Categories (CategoryName) VALUES (@name)";
DbCommandWrapper cmd = em.CreateCommand(CommandWrapperType.Text, sql);
cmd.SetParameter("@name","Chinese Food");

em.ExecuteNonQuery(cmd);
```

ExecuteObject()

```
string sql = "SELECT * FROM Customers WHERE CustomerId=@Id";
DbCommandWrapper cmd = em.CreateCommand(CommandWrapperType.Text, sql);
cmd.SetParameter("@Id","ALFKI");

Customer cust=em.ExecuteObject<Customer>(cmd,new CustomerMapper());
Console.WriteLine(cust.CompanyName);
Console.WriteLine(cust.ContactName);
```

ExecuteList()

```
string sql = "SELECT * FROM Products INNER JOIN Categories ON "
           + "Products.CategoryId = Categories.CategoryId";

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

List<Product> products=em.ExecuteList<Product>(cmd,new ProductMapper());
foreach (Product p in products)
{
     Console.WriteLine(p.ProductName);
     Console.WriteLine(p.Category.CategoryName);
}
```

Stored Procedure

Salah satu fitur default DBMS bertipe client server adalah 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. Maintenence juga dapat dilakuakan dengan mudah karena business process yang mudah berubah ditulis di stored procedure, bukan pada kode program.

Contoh stored procedure

spFindCustomerById

```
CREATE PROCEDURE [dbo].[spFindCustomerById]
  @CustomerId varchar(50)
AS
BEGIN
   SELECT * FROM Customers WHERE CustomerId=@CustomerId
END
```

Cara mengakses nya :

```
DbCommandWrapper cmd = em.CreateCommand(CommandWrapperType.StoredProcedure,
        "spFindCustomerById");
cmd.SetParameter("@CustomerId", "ALFKI");

Customer cust = em.ExecuteObject<Customer>(cmd, new CustomerMapper());
Console.WriteLine(cust.CompanyName);
```

spInsertCustomer

```
CREATE PROCEDURE [dbo].[spInsertCustomer]
  @CustomerId varchar(50),
  @CompanyName varchar(50),
  @ContactName varchar(50)
AS
BEGIN
  INSERT INTO Customers (CustomerId,CompanyName,ContactName) VALUES
  (@CustomerId,@CompanyName,@ContactName)
END
```

```
DbCommandWrapper cmd = em.CreateCommand(CommandWrapperType.StoredProcedure,
        "[spInsertCustomer]");

cmd.SetParameter("@CustomerId", "MSFT");
cmd.SetParameter("@CompanyName", "Microsoft");
cmd.SetParameter("@ContactName", "Bill Gates");

em.ExecuteNonQuery(cmd);
```

Mengakses stored procedure di Repository

```
public class CustomerRepository : ICustomerRepository
{
    private IEntityManager em;

    public CustomerRepository(IEntityManager em)
    {
        this.em = em;
    }

    public Customer FindById(object id)
    {
        DbCommandWrapper cmd = em.CreateCommand(
             CommandWrapperType.StoredProcedure,"spFindCustomerById");
        cmd.SetParameter("@CustomerId", id);

        return em.ExecuteObject<Customer>(cmd, new CustomerMapper());
    }

    public List<Customer> FindAll()
    {
        DbCommandWrapper cmd = em.CreateCommand(
             CommandWrapperType.StoredProcedure, "spFindAll");

        return em.ExecuteList<Customer>(cmd, new CustomerMapper());
    }

    public int Save(Customer cust)
    {
        DbCommandWrapper cmd = em.CreateCommand(
             CommandWrapperType.StoredProcedure, "spInsertCustomer");

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

        return em.ExecuteNonQuery(cmd);
    }

    public int Update(Customer cust)
    {

    }

    public int Delete(Customer cust)j
    {

    }

}
```

Transaction

Transaksi didefenisikan sebagai himpunan satu atau lebih pernyataan yang diekseskusi sebagai satu unit (unit of work), 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.

Berikut ini contoh penggunaan transaction :

```
string provider = ConfigurationManager.AppSettings["Provider"];
string connStr = ConfigurationManager.AppSettings["ConnectionString"];

DataSource dataSource = new DataSource(provider, connStr);
IEntityManager em=EntityManagerFactory.CreateInstance(dataSource);

Transaction tx = em.BeginTransaction();

try
{
    string[] fields1 = { "CustomerId", "OrderDate", "ShippedDate" };
    object[] values1 = { 1, DateTime.Now, DateTime.Now };

    Query q1 = new Query().Select(fields1).From("Orders")
          .Insert(values1);

    em.ExecuteNonQuery(q1.GetSql(), tx);

    string[] fields2 = { "OrderId", "ProductId", "Quantity" };
    object[] values2 = { 1, 10, 2 };

    Query q2 = new Query().Select(fields2).From("OrderDetails")
           .Insert(values2);

    em.ExecuteNonQuery(q2.GetSql(), tx);

    tx.Commit();

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

     Console.WriteLine(ex.ToString());
}
```

Logging

Sebuah aplikasi yang masuk kategori skala enterprise seharusnya ditambahkan kemampuan application logging untuk memonitor kondisi aplikasi yang sedang berjalan. Logging ini sangat berperan jika suatu saat terjadi kesalahan/failure pada aplikasi, 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.

EntityMap menyediakan fitur logging yang cukup lengkap untuk monitoring aplikasi, diantaranya :

1. Mendukung banyak tipe logging seperti : console, file, database, event log, SMTP, dan trace
2. Mendukung penggunaan lebih dari satu tipe logger secara bersamaan
3. Tipe logger dapat diubah dengan hanya dengan mengedit konfigurasinya

Untuk menggunakan fitur logging di EntityMap, sebelumnya harus menambahkan terlebih dahulu namespace EntityMap.Logger. Selanjutnya tinggal menggunakan class-class Logger yang tersedia.

Logger	Keterangan
ConsoleLogger	Log output dikirim ke Console
FileLogger	Log output dikirim ke file
DbLogger	Log ditulis ke tabel di basis data, tabel tersebut harus di buat terlebih dahulu dengan nama LOGS
EventLogger	Log ditulis ke Event Log milik Windows yang bisa diakses dari Control Panel -> Administrative Tools -> Event Viewer
SmtpLogger	Log dikirim lewat e-mail melalui protokol SMTP
TraceLogger	Log dimunculkan di debug window Visual Studio.NET/Express Edition

Untuk menggunakan objek loggervgunakan static class LoggerFactory untuk membuat jenis logger yang diinginkan.

```
ILogger logger = LoggerFactory.CreateConsoleLogger();
```

Untuk menuliskan log 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 ditulis pada log

```
logger.Write(Severity.Info, "Initialize application");
logger.Write(Severity.Error, "Error initialize process");
```

Contoh lainnya :

File Logger

```
ILogger logger = LoggerFactory.CreateFileLogger("c:\\log.txt");
logger.Write(Severity.Info, "log ini disimpan ke file text");
```

Db Logger

```
IEntityManager em = EntityManagerFactory.CreateInstance(dataSource);

ILogger logger = LoggerFactory.CreateDbLogger();
logger.Write(Severity.Info, "log ini disimpan ke tabel logs");
```

Event Logger

```
ILogger logger = LoggerFactory.CreateEventLogger();

logger.EventSource = "Application";
logger.Write(Severity.Info, "Log ini ditulis ke EventLog Windows");
```

Trace Logger

```
ILogger logger = LoggerFactory.CreateTraceLogger();
logger.Write(Severity.Info, "Trace log");
```

Smtp Logger

```
string host = "smtp.gmail.com";
int port = 25;
string mail = "neonerdy@gmail.com";
string password = "secret";
string destination = "myapplog@gmail.com";

ILogger logger = LoggerFactory.CreateSmtpLogger(host, port, mail,
       password, destination);

logger.Write(Severity.Info, "log ini dikirim ke email");
```

Repository Logging

Pasanglah Logger pada bagian yang paling rentan terjadi kesalahan. Pada aplikasi yang menggunakan database, operasi yang paling banyak dilakukan adalah CRUD (Create, Read, Update, Delete) yang diimplementasikan di class repository. Karena itu paling logis memasang logger di repository class.

Untuk mengakses logger tambahkan namespace EntityMap.Logger.

Cara memasang logger

```
public class EmployeeRepository : IEmployeeRepository
{
    private IEntityManager em;
    private ILogger logger;
    private string tableName = "Employees";

    public EmployeeRepository(IEntityManager em, ILogger logger)
    {
        this.em = em;
        this.logger = logger;
    }

}
```

Interface ILogger diletakkan di setiap constructor repository class, sehingga constructor memiliki dua object dependency yaitu em dan logger. Dua object ini diinstantiasi di class ServiceRegistry() dengan menggunakan constructor injection.

```
public class ServiceRegistry : IRegistry
{
   private IEntityManager em;

   public void Configure()
   {
      string provider = ConfigurationManager.AppSettings["Provider"];
      string connStr = ConfigurationManager.AppSettings["ConnectionString"];

      DataSource dataSource=new DataSource(provider,connStr);
      em = EntityManagerFactory.CreateInstance(dataSource);
      ILogger logger = LoggerFactory.CreateFileLogger("c:\\log.txt");

      object[] dependency = {em,logger };

      RepositoryFactory.RegisterObject<IEmployeeRepository,
              EmployeeRepository>(dependency);
    }
}
```

Seperti yang terlihat pada source code diatas, semua dependency diinstantiasi di ServiceRegistry(), konfigurasi object DataSource diambail dari App.config, sedangkan objek logger di konstruksi menggunakan LoggerFactory dengan tipe FileLogger. Semua log yang dipasang di repository class pada contoh diatas akan disimpan ke file log.txt.

Pada dasarnya kita bebas menuliskan apa pun kedalam log. Biasanya apa yang ditulis di log adalah "exception" atau status suatu operasi. Contoh :

```
public int Save(Employee employee)
{
    int result=0;
    try
    {
        string[] fields={"LastName","FirstName","Title",
           "BirthDate","HireDate"};
        object[] values={employee.LastName,employee.FirstName,
            employee.BirthDate,employee.HireDate};

        Query q=new Query().Select(fields).From(tableName).Insert(values);

        result=em.ExecuteNonQuery(q.GetSql());
     }
     catch(Exception ex)
     {
       logger.Write(Severity.Error,ex.Message.ToString());
     }
     return result;
 }
```

Contoh lain :

```
public int Update(Employee employee)
{
     string[] fields={"LastName","FirstName","Title","BirthDate","HireDate"};

     object[] values={employee.LastName,employee.FirstName,
                      employee.BirthDate,employee.HireDate};

     Query q=new Query().Select(fields).From(tableName).Update(values)
         .Where("EmployeeId").Equal(employee.EmployeeId);


    logger.Write(Severity.Info,"admin updated " + employee.EmployeeId);


     return em.ExecuteNonQuery(q.GetSql());

}
```

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 ditest adalah function atau procedure, sedangkan padapemrograman berorientasi objek unit terkecil adalah method, abstract class, atau class turunan.Unit testing biasanya dilakukan secara otomatis dengan menggunakan tools tertentu.

Tujuan dari testing adalah memastikan software tidak memiliki bugs/kesalahan, atau setidak-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. Software testing berhubungan erat dengan kualitas software (*software quality*)

Dalam buku "*Pragmatic Unit Testing*" karya Andrew Hunt, disebutkan prinsip-prinsip utama dalam membuat unit testing yang baik, yaitu :

1. Automatic
   Testing harus bisa dilakukan secara otomatis dengna menggunakan tools, tertentu
2. Through
   Testing harus dilakukan secara keseluruhan
3. Repeatable
   Testing hasilnya tetap sama walaupun dilakukan secara berulang-ulang
4. Independent
   Tidak tergantung pada modul lain
5. Independent
   Menulis unit test sama prioritas nya seperti menulis source code utama

Tools untuk melakukan unit testing dapat menggunakan yang open source seperti Nunit, atau fitur built ini IDE Visual Studio 2008 Professional/Team System.

Perhatikan contoh berikut :

```
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 buatkan sebuah test class nya :

```
[TestClass]
public class CalculatorTest
{
    private Calculator calculator;

    public CalculatorTest()
    {
        calculator=new Calculator();
    }

    [TestMethod]
    public void TestAdd()
    {
       Assert.AeEqual(2,calculator.Add(1,1));
    }

    [TestMethod]
    public void TestMultiply()
    {
       Assert.AreEqual(4, calculator.Add(2, 2));
    }
}
```

Setelah menandai semua method yang akan ditest dengan menggunakan atribut [TestMethod], selanjutnya panggil static class Assert , berikut method-method yang ada :

Method	Keterangan
AreEqual()	Membandingkan nilai expected (yang diharapkan) dengan nilai actual. Jika sama(equal) maka return value nya true
AreNotEqual()	Membandingkan nilai expected(yang diharapkan) dengan nilai actual. Jika tidak sama (not equal) maka return value nya true
AreSame()	Membandingkan apakah kedua objek sama
AreNotSame()	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

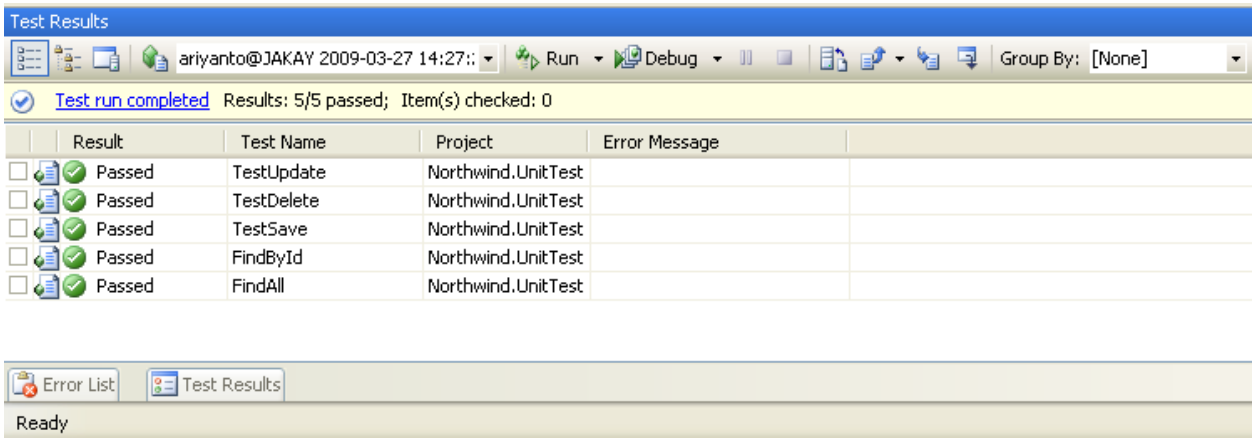
Repository Testing

Class repository adalah bagian yang wajib memiliki unit testing, karena pada class tersebut method CRUD dieksekusi. Berikut ini contoh unit testing untuk class CategoryTest. Unit testing yang baik adalah menyeluruh, artinya semua method-method yang terdapat pada class repository harus dipastikan memiliki unit testing.

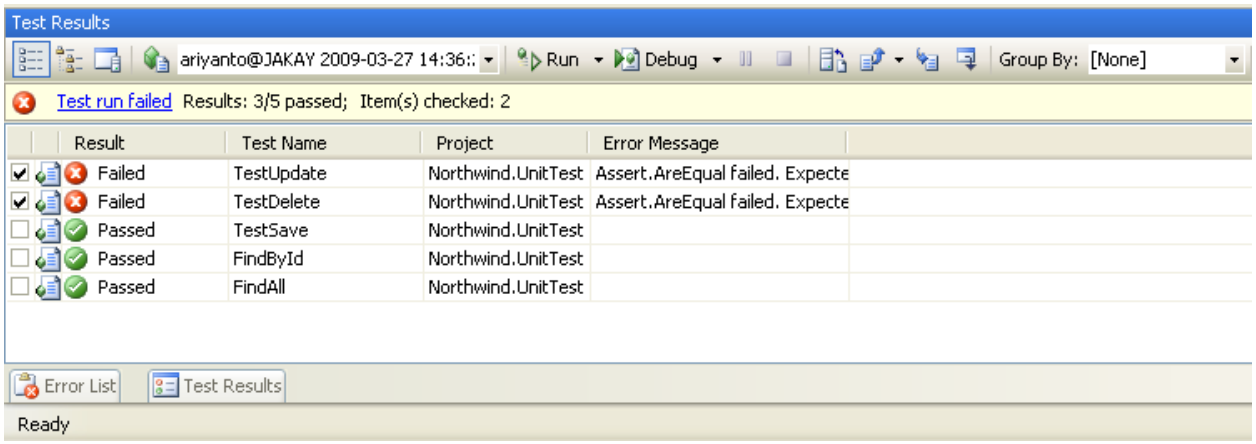
```
[TestClass]
public class CategoryTest
{
     private ICategoryRepository categoryRepository;

     public CategoryTest()
     {
         RepositoryFactory.AddRegistry(new ServiceRegistry());
         categoryRepository=RepositoryFactory
              .GetObject<ICategoryRepository>();
     }

     [TestMethod]
     public void FindById()
     {
         Category category = categoryRepository.FindById(1);
         Assert.AreEqual("Beverages", category.CategoryName);
     }

     [TestMethod]
     public void FindAll()
     {
         List<Category> categories = categoryRepository.FindAll();
         Assert.AreEqual("Beverages", categories[0].CategoryName);
     }

     [TestMethod]
     public void TestSave()
     {
         Category category = new Category();
         category.CategoryName = "Chinese Food";

         Assert.AreEqual(1,categoryRepository.Save(category));
     }

     [TestMethod]
     public void TestUpdate()
     {
         Category category = new Category();
         category.CategoryId = 1;
         category.CategoryName = "Japanese Food";

         Assert.AreEqual(1,categoryRepository.Update(category));
     }
```

```
    [TestMethod]
    public void TestDelete()
    {
        Category category = new Category();
        category.CategoryId = 1;
        Assert.AreEqual(1,categoryRepository.Delete(category));

    }
}
```

Hasil test sukses :



Jika ada test yang gagal, result nya “Failed”



Source Code Lengkap

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 akan ditunjukkan penerapan praktis pada sebuah aplikasi berbasis GUI (WinForm).

App.config

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

Customer

```
namespace Northwind.Model
{
    public class Customer
    {
        public string CustomerId { get; set; }

        public string CompanyName { get; set; }

        public string ContactName { get; set; }

        public string Address { get; set; }

        public string Phone { get; set; }

    }
}
```

Category

```
using System.Collections.Generic;

namespace Northwind.Model
{
    public class Category
    {
        public int CategoryId { get; set; }

        public string CategoryName { get; set; }

        public virtual List<Product> Products { get; set; }
    }
}
```

Product

```
using System;

namespace Northwind.Model
{
    public class Product
    {
        public int ProductId { get; set; }

        public string ProductName { get; set; }

        public int CategoryId { get; set; }

        public Category Category { get; set; }

        public string QuantityPerUnit { get; set; }

        public decimal UnitPrice { get; set; }

        public int UnitsInStock { get; set; }

    }
}
```

CustomerMapper

```
using System;
using System.Data;
using EntityMap;
using Northwind.Model;

namespace Northwind.Repository.Mapping
{
    public class CustomerMapper : IDataMapper<Customer>
    {
        public Customer Map(IDataReader rdr)
        {
            Customer customer = new Customer();

            customer.CustomerId = rdr["CustomerId"] is DBNull ?
                string.Empty : (string)rdr["CustomerId"];
            customer.CompanyName = rdr["CompanyName"] is DBNull ?
                string.Empty : (string)rdr["CompanyName"];
            customer.ContactName = rdr["ContactName"] is DBNull ?
                string.Empty : (string)rdr["ContactName"];
            customer.Address = rdr["Address"] is DBNull ?
                string.Empty : (string)rdr["Address"];
            customer.Phone = rdr["Phone"] is DBNull ?
                string.Empty : (string)rdr["Phone"];

            return customer;
        }
    }
}
```

CategoryMapper

```
using System;
using System.Data;
using EntityMap;
using Northwind.Model;

namespace Northwind.Repository.Mapping
{
    public class CategoryMapper : IDataMapper<Category>
    {
        public Category Map(IDataReader rdr)
        {
            Category category = new CategoryProxy();

            category.CategoryId = rdr["CategoryId"] is DBNull ?
                0 : (int)rdr["CategoryId"];
            category.CategoryName = rdr["CategoryName"] is DBNull ?
                string.Empty : (string)rdr["CategoryName"];

            return category;

        }
    }
}
```

CategoryProxy

```
using System;
using System.Collections.Generic;
using EntityMap;
using Northwind.Model;

namespace Northwind.Repository.Mapping
{
    public class CategoryProxy : Category
    {
        private bool productLoaded = false;

        public override List<Product> Products
        {
            get
            {
                List<Product> products;
                if (!productLoaded)
                {
                    IProductRepository productRepository = RepositoryFactory
                        .GetObject<IProductRepository>();

                    products = productRepository
                        .FindByCategoryId(this.CategoryId);
                    base.Products = products;

                    productLoaded = true;

                }
```

```
                else
                {
                    products=base.Products;
                }
                return products;
            }
            set
            {
                base.Products = value;
                productLoaded = true;
            }
        }
    }
}
```

ProductMapper

```
using System;
using System.Data;
using EntityMap;
using Northwind.Model;

namespace Northwind.Repository.Mapping
{
    public class ProductMapper : IDataMapper<Product>
    {
        public Product Map(IDataReader rdr)
        {
            Product product = new Product();

            product.ProductId = rdr["ProductId"] is DBNull ?
                0 : (int)rdr["ProductId"];
            product.ProductName = rdr["ProductName"] is DBNull ?
                string.Empty : (string)rdr["ProductName"];
            product.CategoryId = rdr["CategoryId"] is DBNull ?
                0 : (int)rdr["CategoryId"];

            CategoryMapper categoryMapper=new CategoryMapper();
            product.Category = categoryMapper.Map(rdr);

            product.QuantityPerUnit = rdr["QuantityPerUnit"] is DBNull ?
                string.Empty : (string)rdr["QuantityPerUnit"];
            product.UnitPrice = rdr["UnitPrice"] is DBNull ?
                0 : (decimal)rdr["UnitPrice"];
            product.UnitsInStock = rdr["UnitsInStock"] is DBNull ?
                0 : (int)rdr["UnitsInStock"];

            return product;
        }

    }
}
```

ICustomerRepository

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

using EntityMap;
using Northwind.Model;

namespace Northwind.Repository
{
    public interface ICustomerRepository : IRepository<Customer>
    {
        List<Customer> FindByName(string name);
    }
}
```

ICategoryRepository

```
using System;

using EntityMap;
using Northwind.Model;

namespace Northwind.Repository
{
    public interface ICategoryRepository : IRepository<Category>
    {
    }
}
```

IProductRepository

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

using EntityMap;
using Northwind.Model;

namespace Northwind.Repository
{
    public interface IProductRepository : IRepository<Product>
    {
        List<Product> FindByName(string name);
        List<Product> FindByCategory(string categoryName);
        List<Product> FindByCategoryId(object id);
    }
}
```

CustomerRepository

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

using EntityMap;
using Northwind.Model;
using Northwind.Repository.Mapping;

namespace Northwind.Repository.Implementation
{
    public class CustomerRepository : ICustomerRepository
    {
        private IEntityManager em;
        private string tableName = "Customers";

        public CustomerRepository(IEntityManager em)
        {
            this.em = em;
        }

        public Customer FindById(object id)
        {
            Query q = new Query().From(tableName)
                .Where("CustomerId").Equal(id);
            return em.ExecuteObject<Customer>(q.GetSql(),
                new CustomerMapper());
        }

        public List<Customer> FindAll()
        {
            Query q = new Query().From(tableName);
            return em.ExecuteList<Customer>(q.GetSql(),
                new CustomerMapper());
        }

        public List<Customer> FindByName(string name)
        {
            Query q = new Query().From(tableName)
                .Where("CompanyName").Like("%" + name + "%");

            return em.ExecuteList<Customer>(q.GetSql(),
                 new CustomerMapper());
        }


        public int Save(Customer cust)
        {
            string[] fields = {"CustomerId","CompanyName","ContactName",
                               "Address","Phone"};
            object[] values ={cust.CustomerId,cust.CompanyName,
                              cust.ContactName,cust.Address,
                              cust.Phone};

            Query q = new Query().Select(fields).From(tableName)
                 .Insert(values);
```

```
            return em.ExecuteNonQuery(q.GetSql());
        }


        public int Update(Customer cust)
        {
            string[] fields = { "CompanyName", "ContactName",
                                "Address", "Phone" };
            object[] values ={cust.CompanyName,cust.ContactName,cust.Address,
                               cust.Phone};

            Query q = new Query().Select(fields).From(tableName)
                .Update(values).Where("CustomerId")
                .Equal(cust.CustomerId);

            return em.ExecuteNonQuery(q.GetSql());

        }

        public int Delete(Customer cust)
        {
            Query q = new Query().From(tableName).Delete()
                .Where("CustomerId").Equal(cust.CustomerId);

            return em.ExecuteNonQuery(q.GetSql());

        }
    }
}
```

CategoryRepository

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

using EntityMap;
using Northwind.Model;
using Northwind.Repository.Mapping;

namespace Northwind.Repository.Implementation
{
    public class CategoryRepository : ICategoryRepository
    {
        private IEntityManager em;
        private string tableName = "Categories";
        private IProductRepository productRepository;

        public CategoryRepository(IEntityManager em)
        {
            this.em = em;
            productRepository = RepositoryFactory
               .GetObject<IProductRepository>();
        }
```

```
        public Category FindById(object id)
        {
            Query q = new Query().From(tableName).Where("CategoryId")
                  .Equal(id);
            Category category=em.ExecuteObject<Category>(q.GetSql(),
                  new CategoryMapper());

            return category;
        }

        public List<Category> FindAll()
        {
            Query q = new Query().From(tableName);

            List<Category> categories=em.ExecuteList<Category>(q.GetSql(),
                new CategoryMapper());
            return categories;
        }

        public int Save(Category category)
        {
            string[] fields = {"CategoryName"};
            object[] values = { category.CategoryName };

            Query q = new Query().Select(fields).From(tableName)
                .Insert(values);

            return em.ExecuteNonQuery(q.GetSql());
        }


        public int Update(Category category)
        {
            string[] fields = { "CategoryName" };
            object[] values = { category.CategoryName };

            Query q = new Query().Select(fields).From(tableName)
                .Update(values)
                .Where("CategoryId").Equal(category.CategoryId);

            return em.ExecuteNonQuery(q.GetSql());

        }

        public int Delete(Category category)
        {
            Query q = new Query().From(tableName).Delete()
                .Where("CategoryId").Equal(category.CategoryId);

            return em.ExecuteNonQuery(q.GetSql());
        }

    }
}
```

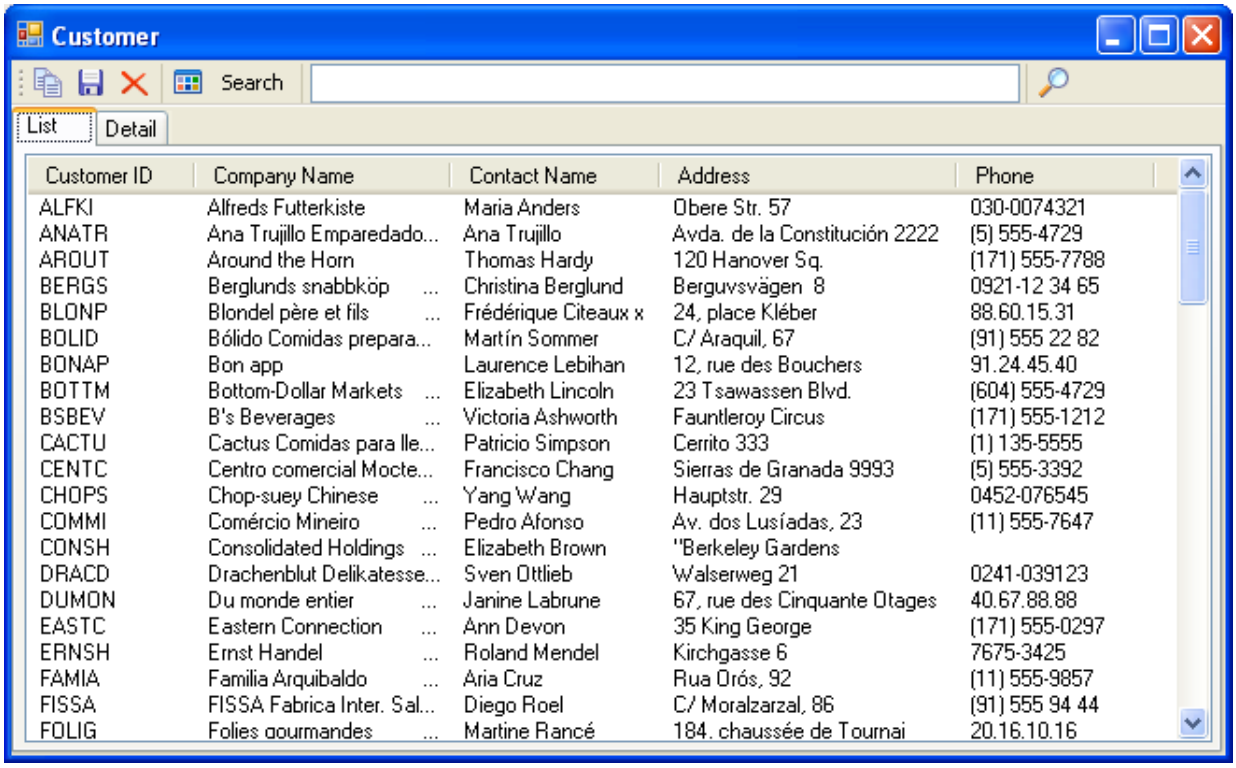
ProductRepository

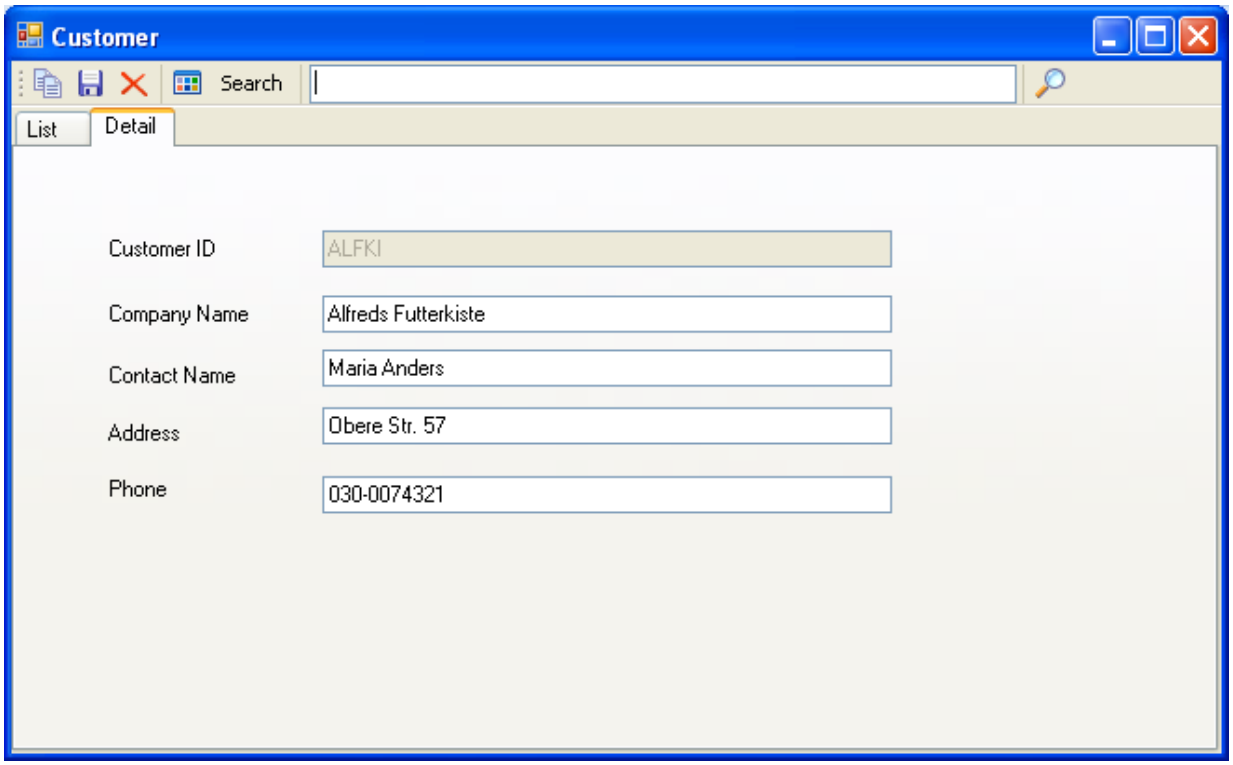
```
using System;
using System.Collections.Generic;

using EntityMap;
using Northwind.Model;
using Northwind.Repository.Mapping;

namespace Northwind.Repository.Implementation
{
   public class ProductRepository : IProductRepository
   {
        private IEntityManager em;
        private string tableName = "Products";

        public ProductRepository(IEntityManager em)
        {
            this.em = em;
        }

        public Product FindById(object id)
        {
             Query q=new Query().From(tableName)
                 .InnerJoin("Categories","CategoryId","CategoryId")
                 .Where("ProductId").Equal(id);

            return em.ExecuteObject<Product>(q.GetSql(),new ProductMapper());
        }


        public List<Product> FindAll()
        {
            Query q = new Query().From(tableName)
                .InnerJoin("Categories", "CategoryId", "CategoryId");
            return em.ExecuteList<Product>(q.GetSql(), new ProductMapper());
        }

        public List<Product> FindByName(string name)
        {
            Query q = new Query().From(tableName)
                .InnerJoin("Categories", "CategoryId", "CategoryId")
                .Where("ProductName").Like("%" + name + "%");

            return em.ExecuteList<Product>(q.GetSql(), new ProductMapper());
       }

        public List<Product> FindByCategoryId(object id)
        {
            Query q = new Query().From(tableName)
              .InnerJoin("Categories", "CategoryId", "CategoryId")
              .Where("Categories.CategoryId").Equal(id);

            return em.ExecuteList<Product>(q.GetSql(), new ProductMapper());
        }
```

```
        public List<Product> FindByCategory(string categoryName)
        {
            Query q = new Query().From(tableName)
                .InnerJoin("Categories", "CategoryId", "CategoryId")
                .Where("Categories.CategoryName").Equal(categoryName);

            return em.ExecuteList<Product>(q.GetSql(), new ProductMapper());

        }

        public int Save(Product product)
        {
            string[] fields = { "ProductName", "CategoryId",
                                "QuantityPerUnit", "UnitPrice",
                                "UnitsInStock" };
            object[] values = { product.ProductName,product.CategoryId,
                                 product.QuantityPerUnit,
                                 product.UnitPrice,product.UnitsInStock };

            Query q = new Query().Select(fields).From(tableName)
                .Insert(values);

            return em.ExecuteNonQuery(q.GetSql());
        }

        public int Update(Product product)
        {
            string[] fields = { "ProductName", "CategoryId",
                                "QuantityPerUnit", "UnitPrice",
                                "UnitsInStock" };
            object[] values = { product.ProductName,product.CategoryId,
                                 product.QuantityPerUnit,
                                 product.UnitPrice,product.UnitsInStock };

            Query q = new Query().Select(fields).From(tableName)
                .Update(values).Where("ProductId")
                .Equal(product.ProductId);

            return em.ExecuteNonQuery(q.GetSql());
        }

        public int Delete(Product product)
        {
            Query q = new Query().From(tableName).Delete()
                .Where("ProductId").Equal(product.ProductId);

            return em.ExecuteNonQuery(q.GetSql());
        }

    }
}
```

ServiceRegistry

```
using System;
using System.Configuration;

using EntityMap;
using Northwind.Repository;
using Northwind.Repository.Implementation;

namespace Northwind.Repository
{
    public class ServiceRegistry : IRegistry,IDisposable
    {
        private IEntityManager em;

        public void Configure()
        {
            string provider = ConfigurationManager
                   .AppSettings["Provider"];

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

            DataSource dataSource=new DataSource(provider,connStr);

            em = EntityManagerFactory.CreateInstance(dataSource);

            object[] dependency = { em };

            RepositoryFactory.RegisterObject<ICategoryRepository,
                CategoryRepository>(dependency);

            RepositoryFactory.RegisterObject<IProductRepository,
                ProductRepository>(dependency);

            RepositoryFactory.RegisterObject<ICustomerRepository,
                CustomerRepository>(dependency);

        }

        public void Dispose()
        {
            em.Dispose();
        }

    }
}
```


Customer
Search
List
Detail
Customer ID
Company Name
Contact Name
Address
Phone
ALFKI Alfreds Futterkiste Maria Anders Obere Str. 57 030-0074321
ANATR Ana Trujillo Emparedado... Ana Trujillo Avda. de la Constitución 2222 (5) 555-4729
AROUT Around the Horn Thomas Hardy 120 Hanover Sq. (171) 555-7788
BERGS Berglunds snabbköp ... Christina Berglund Berguvsvägen 8 0921-12 34 65
BLONP Blondel père et fils ... Frédérique Citeaux x 24, place Kléber 88.60.15.31
BOLID Bólido Comidas prepara... Martín Sommer C/ Araquil, 67 (91) 555 22 82
BONAP Bon app Laurence Lebihan 12, rue des Bouchers 91.24.45.40
BOTTM Bottom-Dollar Markets ... Elizabeth Lincoln 23 Tsawassen Blvd. (604) 555-4729
BSBEV B's Beverages ... Victoria Ashworth Fauntleroy Circus (171) 555-1212
CACTU Cactus Comidas para lle... Patricio Simpson Cerrito 333 (1) 135-5555
CENTC Centro comercial Mocte... Francisco Chang Sierras de Granada 9993 (5) 555-3392
CHOPS Chop-suey Chinese ... Yang Wang Hauptstr. 29 0452-076545
COMMI Comércio Mineiro ... Pedro Afonso Av. dos Lusíadas, 23 (11) 555-7647
CONSH Consolidated Holdings ... Elizabeth Brown "Berkeley Gardens
DRACD Drachenblut Delikatesse... Sven Ottlieb Walserweg 21 0241-039123
DUMON Du monde entier ... Janine Labrune 67, rue des Cinquante Otages 40.67.88.88
EASTC Eastern Connection ... Ann Devon 35 King George (171) 555-0297
ERNSH Ernst Handel ... Roland Mendel Kirchgasse 6 7675-3425
FAMIA Familia Arquibaldo ... Aria Cruz Rua Orós, 92 (11) 555-9857
FISSA FISSA Fabrica Inter. Sal... Diego Roel C/ Moralzarzal, 86 (91) 555 94 44
FOLIG Folies gourmandes ... Martine Rancé 184, chaussée de Tournai 20.16.10.16


Customer
Search
List
Detail
Customer ID
ALFKI
Company Name
Alfreds Futterkiste
Contact Name
Maria Anders
Address
Obere Str. 57
Phone
030-0074321

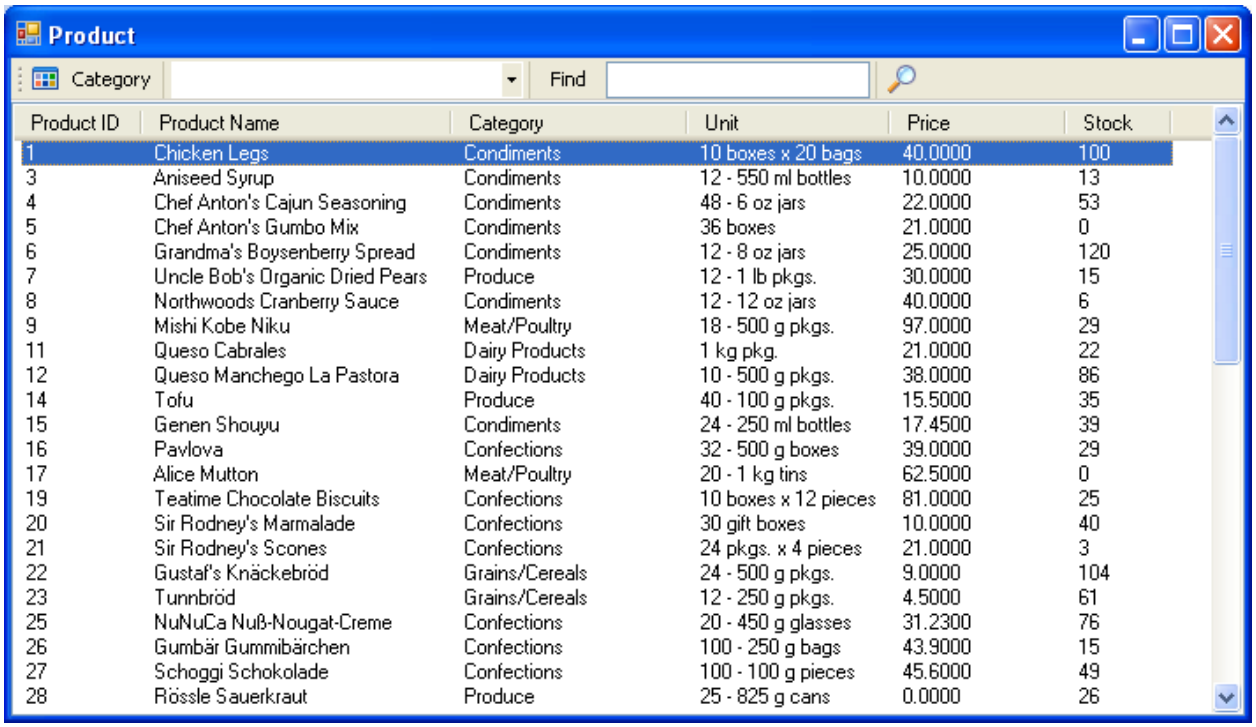
CustomerUI

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

using EntityMap;
using Northwind.Repository;
using Northwind.Model;

namespace Northwind.Presentation
{
    public partial class CustomerUI : Form
    {
        enum FormState
        {
            New,Edit
        }

        private IRegistry serviceRegistry;
        private ICustomerRepository customerRepository;
        private FormState formState;

        public CustomerUI()
        {
            InitializeComponent();

            serviceRegistry = new ServiceRegistry();
            RepositoryFactory.AddRegistry(serviceRegistry);

            customerRepository = RepositoryFactory
                .GetObject<ICustomerRepository>();
        }

        private void FillListView(Customer cust)
        {
            ListViewItem item = new ListViewItem(cust.CustomerId.ToString());

            item.SubItems.Add(cust.CompanyName);
            item.SubItems.Add(cust.ContactName);
            item.SubItems.Add(cust.Address);
            item.SubItems.Add(cust.Phone);

            lvwCustomer.Items.Add(item);

        }

        private void FillCustomer()
        {
            List<Customer> custs = customerRepository.FindAll();

            lvwCustomer.Items.Clear();
            foreach (Customer cust in custs)
```

```
        {
            FillListView(cust);
        }
    }



    private void ViewDetail()
    {
        Customer cust = customerRepository
            .FindById(lvwCustomer.FocusedItem.Text);

        txtCustomerID.Text = cust.CustomerId;
        txtCompanyName.Text = cust.CompanyName;
        txtContactName.Text = cust.ContactName;
        txtAddress.Text = cust.Address;
        txtPhone.Text = cust.Phone;
    }

    private void Clear()
    {
        txtCustomerID.Enabled = true;
        txtCustomerID.Clear();
        txtCompanyName.Clear();
        txtContactName.Clear();
        txtAddress.Clear();
        txtPhone.Clear();
    }

    private void SaveCustomer()
    {
        Customer cust = new Customer();

        cust.CustomerId = txtCustomerID.Text;
        cust.CompanyName = txtCompanyName.Text;
        cust.ContactName = txtContactName.Text;
        cust.Address = txtAddress.Text;
        cust.Phone = txtPhone.Text;

        customerRepository.Save(cust);

    }

    private void UpdateCustomer()
    {
        Customer cust = new Customer();

        cust.CustomerId = txtCustomerID.Text;
        cust.CompanyName = txtCompanyName.Text;
        cust.ContactName = txtContactName.Text;
        cust.Address = txtAddress.Text;
        cust.Phone = txtPhone.Text;

        customerRepository.Update(cust);

    }
```

```
private void Delete()
{
    Customer cust = new Customer();
    cust.CustomerId = lvwCustomer.FocusedItem.Text;
    customerRepository.Delete(cust);
}


private void FindCustomer()
{
    List<Customer> custs=customerRepository.FindByName(txtFind.Text);
    lvwCustomer.Items.Clear();
    foreach (Customer cust in custs)
    {
        FillListView(cust);
    }
}


private void Form1_Load(object sender, EventArgs e)
{
    FillCustomer();
}

private void btnFind_Click(object sender, EventArgs e)
{
    FindCustomer();
    tabControl1.SelectedTab = this.tabPage1;
}

private void lvwSupplier_DoubleClick(object sender, EventArgs e)
{
    formState = FormState.Edit;
    txtCustomerID.Enabled = false;
    ViewDetail();
    tabControl1.SelectedTab = this.tabPage2;
}

private void btnNew_Click(object sender, EventArgs e)
{
    formState = FormState.New;
    tabControl1.SelectedTab = this.tabPage2;
    Clear();
}

private void btnSave_Click(object sender, EventArgs e)
{
    if (formState == FormState.New)
    {
        SaveCustomer();
    }
    else if (formState == FormState.Edit)
    {
        UpdateCustomer();
    }
```

```
            Clear();
            FillCustomer();
            tabControl1.SelectedTab = this.tabPage1;
        }

        private void btnDelete_Click(object sender, EventArgs e)
        {
            if (MessageBox.Show("Are you sure want to delete '"
                + lvwCustomer.FocusedItem.SubItems[1].Text + "'?","Confirm",
               MessageBoxButtons.YesNo,MessageBoxIcon.Question)
               ==DialogResult.Yes)
            {
                Delete();
                FillCustomer();
            }

         }

        private void CustomerUI_FormClosed(object sender,
           FormClosedEventArgs e)
        {
            serviceRegistry.Dispose();
        }
    }
}
```



ProductUI

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;
using EntityMap;
using Northwind.Model;
using Northwind.Repository;

namespace Northwind.Presentation
{
    public partial class ProductUI : Form
    {
        private IProductRepository productRepository;
        private ICategoryRepository categoryRepsoitory;
        private ServiceRegistry serviceRegistry;

        public ProductUI()
        {
            InitializeComponent();
            serviceRegistry = new ServiceRegistry();
            RepositoryFactory.AddRegistry(serviceRegistry);

            productRepository=RepositoryFactory
                .GetObject<IProductRepository>();
            categoryRepsoitory = RepositoryFactory
                .GetObject<ICategoryRepository>();
        }


        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.QuantityPerUnit.ToString());
            item.SubItems.Add(p.UnitPrice.ToString());
            item.SubItems.Add(p.UnitsInStock.ToString());
            lvwProduct.Items.Add(item);
        }

        private void FillProduct()
        {
            List<Product> products = productRepository.FindAll();
            foreach (Product p in products)
            {
                FillListView(p);
            }
        }

        public void FillCategory()
        {
            List<Category> categories = categoryRepsoitory.FindAll();
            foreach (Category c in categories)
```

```
            {
                cbCategory.Items.Add(c.CategoryName);
            }
        }

        public void GetProductByCategory()
        {
            List<Product> products = productRepository
               .FindByCategory(cbCategory.Text);

            lvwProduct.Items.Clear();
            foreach (Product p in products)
            {
                FillListView(p);
            }
        }

        private void FindProduct()
        {
            List<Product> products = productRepository
                 FindByName(txtFind.Text);

            lvwProduct.Items.Clear();
            foreach (Product p in products)
            {
                FillListView(p);
            }
        }


        private void ProductUI_Load(object sender, EventArgs e)
        {
            FillProduct();
            FillCategory();
        }

        private void cbCategory_SelectedIndexChanged(object sender,
            EventArgs e)
        {
            GetProductByCategory();
        }

        private void btnFind_Click(object sender, EventArgs e)
        {
            FindProduct();
        }

        private void ProductUI_FormClosed(object sender,
            FormClosedEventArgs e)
        {
            serviceRegistry.Dispose();
        }

    }
}
```

Daftar Pustaka

Ariyanto, *".NET Enterprise Application Programming*", INDC, Jakarta, 2008.

Fowler, Martin, "*Patterns of Enterprise Application Architecture*", Addison Wesley, USA, 2002.

Eposito, Dino, "*Architecting Microsoft® .NET Solutions for the Enterprise*", Microsoft Press, USA,2008.

Eric, Gamma, "*Design Patterns : Element of Reusable Object-Oriented Software*", Addison Wesley, USA, 1995.

Mehta, Vijay. "*Pro LINQ Object Relational Mapping with C# 2008*", Apress, USA, 2008.

Microsoft, "*Application Architecture for .NET : Designing Applications and Services "*, Microsoft Corporation, USA, 2002.

Microsoft, "*.NET Data Access Architecture Guide "*, Microsoft Corporation, USA, 2003.

Biografi Penulis

Ariyanto, software engineer Petrolink Services Indonesia (www.petrolink.com). 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 dan Basis Data Client Server. Saat ini tertarik pada Object Oriented Technology, Software Engineering, dan Design Pattern. Informasi lebih lanjut mengenai EntityMap dan penulis bisa didapat melalui E-mail : neonerdy@yahoo.com