TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN – ĐHQG TP. HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN

MÔN HỌC
MẪU THIẾT KẾ HƯỚNG ĐỐI TƯỢNG VÀ ỨNG DỤNG

BÁO CÁO CUỐI KỲ

DATABASE ACCESS MANAGEMENT FRAMEWORK (DAM)

Tp. Hồ Chí Minh, tháng 01 năm 2019

MỤC LỤC

Thông tin nhóm

Lớp Cử nhân Tài năng khóa 2015

MSSV	Họ và tên	Email	Số điện thoại
1512159	Hoàng Trung Hiếu	1512159@student.hcmus.edu.vn	01228759002
1512387	Đỗ Thành Nhơn	1512387@student.hcmus.edu.vn	0981662937
1512292	Châu Hoàng Long	1512292@student.hcmus.edu.vn	

1. Bảng tổng hợp các tính năng đã hoàn thành

STT	Tên tính năng	Mô tả	Mức độ hoàn thành
1	Hỗ trợ Code First	Sử dụng framework thông qua việc tạo ra các lớp theo định dạng chuẩn do framework quy định	Hoàn thành 100%
2	Hỗ trợ Model First	Phần mềm sẽ tự động tạo kết nối tới CSDL và tự động sinh ra các tệp tin mã nguồn mô tả lớp đối tượng theo định dạng chuẩn tương ứng với từng bảng (Có giao diện đồ họa)	Hoàn thành 100%
3	Kết nối cơ sở dữ liệu	Cho phép mở và tạo kết nối đến cơ sở dữ liệu thông qua Connection string	Hoàn thành 100%
4	Select All, Select First	Lấy tất cả hay một dòng dữ liệu có trong một table tương ứng với class và tạo ra các đối tượng tương ứng.	Hoàn thành 100%
5	Mệnh đề Where	Hỗ trợ Select với mệnh đề Where, biểu thức điều kiện được truyền vào dưới dạng Lambda Expression	Hoàn thành 100%
6	Mệnh đề Having	Hỗ trợ Select với mệnh đề Having, biểu thức điều kiện được truyền vào dưới dạng Lambda Expression	Hoàn thành 100%
7	Mệnh đề Group by	Hỗ trợ Select với mệnh đề Group by, biểu thức điều kiện được truyền vào dưới dạng Lambda Expression	Hoàn thành 100%
8	Mệnh đề limit	Hỗ trợ Select với mệnh đề limit	Hoàn thành 100%

9	Mệnh đề offset	Hỗ trợ Select với mệnh đề offset	Hoàn thành 100%
10	Thao tác Insert	Cho phép thêm mới một đối tượng được tạo lập vào CSDL	Hoàn thành 100%
11	Thao tác Update	Cho phép cập nhật các sự thay đổi cho một đối tượng sẵn có trong CSDL	Hoàn thành 100%
12	Thao tác Delete	Chp phép xóa một đối tượng khỏi CSDL	Hoàn thành 100%
13	Unit of work	Theo dõi thường xuyên hoạt động của đối tượng như thêm mới, thay đổi, xoá. Phân biệt được khi nào cần Update và khi nào cần Insert đối tượng. Khi người dùng gọi lệnh Commit thì mới thực hiện các câu SQL thay đổi xuống CSDL	Hoàn thành 100%
14	Thể hiện được quan hệ giữa các bảng (quan hệ khóa ngoại) thông qua các lớp đối tượng	Các quan hệ 1 – N hay N – 1 đều được thể hiện giữa các đối tượng (Ví dụ lớp `HocSinh` sẽ chứa 1 đối tượng `LopHoc`, ngược lại 1 đối tượng `LopHoc` sẽ chưa một `List<HocSinh>`)	Hoàn thành 100%
15	Lazing loading	Khi thực hiện việc tạo đối tượng, chỉ một phần đối tượng được tải lên bộ nhớ, phần đầy đủ của đối tượng sẽ được tải khi có nhu cầu truy xuất các thuộc tính của chúng	Hoàn thành 100%
16	Identity Map	Đảm bảo mỗi một dòng trong CSDL chỉ được ánh xạ với	Hoàn thành 100%

		một đối tượng duy nhất trong một Session	
17	Hỗ trợ PostgreSQL	Các hàm cần thiết được cài đặt để minh họa hoạt động, tính dễ mở rộng của framework thông qua tạo kết nối PostgreSQL.	Hoàn thành 100%
18	Hỗ trợ MySQL	Các hàm cần thiết được cài đặt để minh họa hoạt động, tính dễ mở rộng của framework thông qua tạo kết nối MySQL.	Hoàn thành 100%
19	Có khả năng thay đổi hệ quản trị CDSL	Có thể dễ dàng hỗ trợ thêm một loại HQT CSDL mới mà không phải chỉnh sửa mã nguồn cũ. Việc thay đổi này bao gồm đối tượng quản lý kết nối với CSDL và các đối tượng sinh ra câu truy vấn SQL ứng với cú pháp của hệ quản trị CSDL đó	Hoàn thành 100%

2. Sơ đồ lớp tổng hợp của framework

3. Các kỹ thuật đã sử dụng trong framework:

3.1. Reflection – generic Type

Trong framework này, nhằm mục đích đảm bảo tính tổng quát, các kỹ thuật Generic Type được sử dụng triệt để. Các lớp `Table<T>` lưu trữ các đối tượng được tạo ra, trong quá trình sử dụng ta không thể biết trước được đối tượng được tạo ra có kiểu dữ liệu gì, do đó để sử dụng được các thuộc tính trang bị bởi `Table<T>` ta cần dùng kỹ thuật Reflection.

Reflection được hiểu là một chức năng trong .Net cho phép đọc thông tin từ các siêu dữ liệu (metadata) của assembly để tạo ra một đối tượng (có kiểu là Type) bao gói các thông tin đó lại. Với reflection, bạn có thể trích xuất để gọi và tạo ra các phương thức, truy cập và thay đổi các thuộc tính của đối tượng một cách linh động trong quá trình runtime.

Một ví dụ về quá trình sử dụng Reflection trong Framework:

Đầu tiên, dựa vào thông tin các bảng được lưu từ `session`, phương thức có tên **"Table"** sẽ được gọi cho phép truy cập đến đối tượng `Table<T>` phù hợp với kiểu dữ liệu của đối tượng `obj` đang giữ. Kết quả trả về kiểu `Table<T>.`

```
MethodInfo method =
session.GetType().GetMethod("Table").MakeGenericMethod(obj
.GetType().GetProperty(tmp.Key).PropertyType);

var table = method.Invoke(session, null);
```

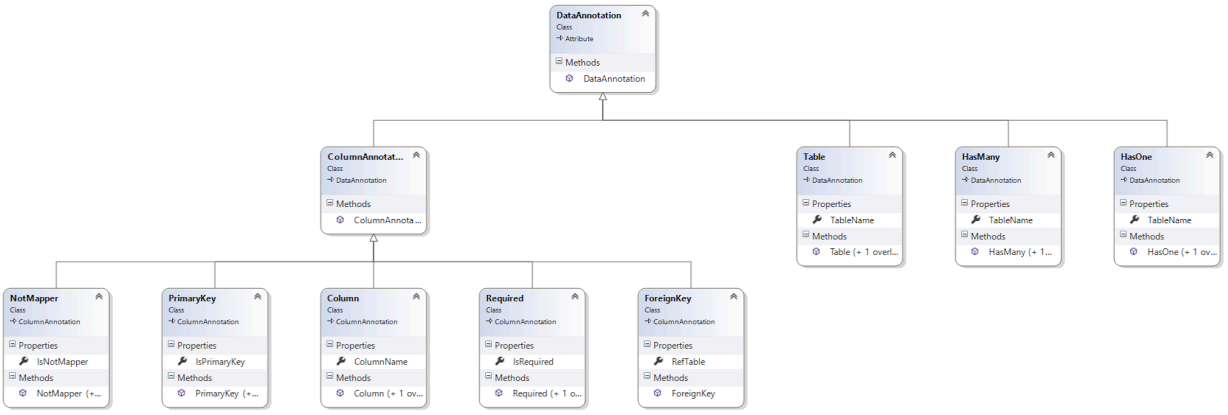
Như vậy, kết thúc quá trình ta có thể lấy được khóa chính của bảng lưu trong đối tượng `Table<T>`, phương thức có tên **"GetPrimaryKeyName"** sẽ được gọi khi biết kiểu dữ liệu của `obj`.

```
method = table.GetType().GetMethod("GetPrimaryKeyName");
var priCol = method.Invoke(table,null);
```

3.2. Data annotation

Bước đầu tiên trong việc xây dựng ORM Framework là xác định được cách định nghĩa mối liên hệ - tương quan giữa các thuộc tính của một đối tượng với các cột và dòng trên bảng CSDL. Có nhiều kĩ thuật để định nghĩa và mô tả dữ liệu của bảng trong CSDL với thuộc tính của lớp như dùng file xml, json…

Framework mà nhóm xây dựng sẽ sử dụng kĩ thuật *Data annotation* bằng cách định nghĩa lại một interface chung là `Data Annotation` kế thừa lại lớp `Attribute`



Lần lượt định nghĩa các lớp kế thừa mô tả lại các cột ở CSDL tương ứng với các thuộc tính của lớp. Khi thực thi chương trình, các *Data Annotation* sẽ được đọc lên bằng *reflection* và *attributes,* các ánh xạ tương ứng sẽ được thiết lập xuống các cột ở database. Nếu thuộc tính nào của lớp không có phần *Data Annotation* thì xem như sử dụng thông tin mặc định (cùng tên với thuộc tính hoặc lớp).

Các loại Data Annotation mà framework hỗ trợ:

- `Table:` Thông tin tên của bảng
- `Column:` Thông tin về tên của cột dữ liệu
- `PrimaryKey:` Thông tin về khoá chính
- `ForeignKey:` Thông tin về khoá ngoại
- `Required:` Cột này không được phép bỏ trống
- `HasMany:` Thể hiện quan hệ 1-n
- `HasOne:` Thể hiện quan hệ 1-1
- `NotMapper:` Không ánh xạ xuống CSDL

Ví dụ trong lớp `Student`, thuộc tính `Name` sẽ được ánh xạ tới cột `ten` thông tin mô tả được mô tả như đoạn chương trình bên dưới đây:

```
private string _name = null;
        [Column("ten")]
```

```
        public string Name
        {
            get
            {
                Load<Student>();
                return _name;
            }
            set
            {
                CheckDirty<Student>();
                _name = value;
            }
        }
```

Cài đặt của lớp `Primary Key` như sau:

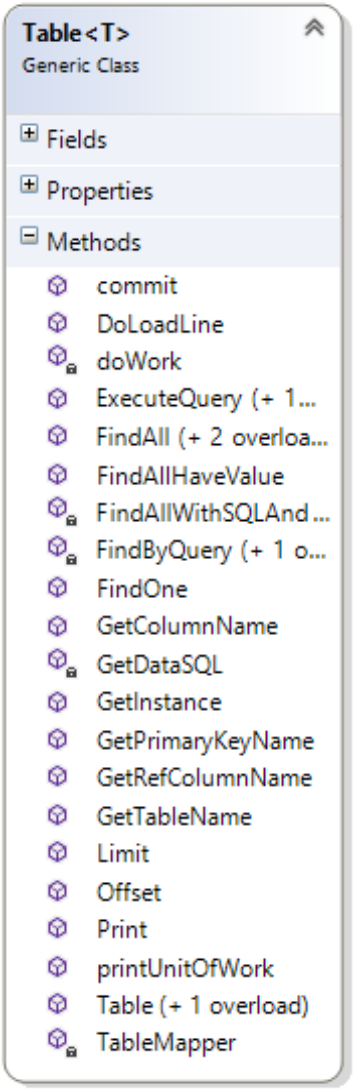
```
[AttributeUsage(AttributeTargets.All)]
    public class PrimaryKey : ColumnAnnotation
    {
        public bool IsPrimaryKey { get; set; }

        public PrimaryKey()
        {
            IsPrimaryKey = true;
        }

        public PrimaryKey(bool isPrimaryKey)
        {
            IsPrimaryKey = isPrimaryKey;
        }
    }
```

3.3. Data Table Gateway

Kỹ thuật quản lý Data Table Gateway được hiểu là xây dựng một đối tượng sẽ quản lý việc truy cập đến các bảng trong cơ sở dữ liệu. Nói cách khác, đối tượng này sẽ là "cửa ngõ" để thực hiện tất cả các thao tác trên tất cả các dòng trong bảng.



Kỹ thuật này được giới thiệu nhằm hạn chế tối đa việc trộn lẫn các câu truy vấn SQL trong tầng ứng dụng – một trong những nhân tố sẽ ảnh hưởng đến mức độ "thoải mái" của người lập trình. *Table Data Gateway* ra đời sẽ hỗ trợ việc kiểm soát truy cập và tổng hợp các câu truy vấn SQL phù hợp có thể truy xuất thay đổi và chỉnh sửa các dòng trong cơ sở dữ liệu. Nó cũng đồng thời cung cấp một interface đơn giản, thông thường bao gồm một vài các phương thức tìm kiếm để lấy dữ liệu và cập nhật, xóa hoặc thêm mới. Mỗi phương thức sẽ thực hiện việc ánh xạ các tham số đầu vào tạo thành câu truy vấn SQL và thực thi các câu truy vấn SQL đó.

Table Data Gateway được cài đặt trong framework này có thể trả về một danh sách các đối tượng tương ứng với lớp đối tượng quản lý bởi Table đó hoặc có thể trả về một danh sách, với mỗi phần tử là các `object []` `(C#)` cho người dùng, trong trường hợp các giá trị trả về không phải là một đối tượng tương ứng mà là một danh sách các đối tượng nào đó (Ví dụ như tính điểm trung bình của các học sinh trong cùng một lớp).

Data Mapper thường được sử dụng chung với *Table Data Gateway*, các thông tin về từng cột như khóa chính – khóa ngoại đều được quản lý bởi đối tượng này. Nhờ việc quản lý một cách tập trung, framework này hỗ trợ biểu diễn quan hệ giữa các bảng dưới dạng các lớp đối tượng. Ví dụ lớp `HocSinh` sẽ chứa 1 đối tượng `LopHoc`, ngược lại 1 đối tượng `LopHoc` sẽ chưa một `List<HocSinh>`.

Kỹ thuật *Generic Type* được sử dụng để quản lý các bảng. Tùy theo số lượng bảng trên CSDL mà các đối tượng này sẽ được tạo tương ứng và linh động. Mỗi một bảng trong CSDL sẽ có duy nhất một bảng duy nhất ứng với đối tượng đó trong một phiên làm việc. Khi người dùng cần lấy thông tin, hay cập nhật các đối tượng thuộc một lớp nào, họ sẽ làm việc với `Table<T>` của lớp tương ứng. Trong đó T chính là lớp của đối tượng.

Ví dụ cho việc thực hiện một số truy vấn đơn giản như:

```
List<Dummy.Student> students =
          session.Table<Dummy.Student>().FindAll()
foreach (Dummy.Student std in students)
            {
                std.Print();
            }
```

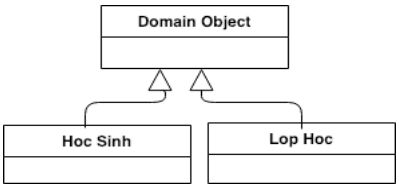
3.4. Identity Map

Kỹ thuật *Identity Map* sẽ đảm bảo một dòng trong cơ sở dữ liệu chỉ có thể ánh xạ với thành một đối tượng cụ thể trong một phiên làm việc. Kỹ thuật này rất quan trọng trong việc đảm bảo tính "nhất quán" của dữ liệu. Cùng một dòng dữ liệu nhưng lại tham chiếu đến hai đối tượng khác nhau thì thực sự rất nguy hiểm, ta sẽ không biết được đâu mới là đối tượng chính xác. Thêm nữa, việc người dùng yêu cầu đọc dữ liệu từ một dòng có thể lặp lại rất nhiều lần trong chương trình, và với mỗi lần như vậy thay vì phải lên CSDL để đọc lại, ta có thể trả về chính đối tượng đó. Điều này giảm thiểu việc thực hiện các câu truy vấn SQL đồng thời tránh việc tạo thành một vòng lặp vô hạn trong trường hợp hai đối tượng chứa lẫn nhau, đọc đối tượng này sẽ dẫn đến việc đọc và tạo mới đối tượng còn lại.

Trong framework này, mỗi một bảng (`Table<T>`) sẽ có một kiểu Dictionary (C#) mapping giữa khóa chính (key) và object. *Identity Mapping* sẽ thường đi chung với Unit of work được trình bày trong phần sau.

Quá trình thực hiện việc "đăng ký" khi một object mới được ra đời được thực hiện như sau (Trong framework, để dễ quản lý *Identity Map* được cài chung với *Unit of Work*):

Domain Object



Lớp đối tượng Domain Object sẽ đóng vai trò là lớp cha của tất cả các đối tượng dữ liệu được tạo ra trong framework. Domain Object sẽ cung cấp các phương thức và thuộc tính cho mọi đối tượng trong framework, giúp quản lý *Identity Map và LazyLoading, Unit of Work (sẽ được trình bày ở phần sau)*

Ở mỗi lớp đối tượng quản lý bảng bất kỳ, ta sẽ luôn có phương thức sau dùng để một đối tương "tự đăng ký" chính nó vào *Identity Map*

```
public bool RegisterFromDatabase(ORMFramework.DomainObject o
bj)
```

Trong hàm đó, đơn giản ta chỉ cần đưa Key và đối tượng đó vào Dictionary. Lưu ý Key sẽ được lưu ở *Domain Object* dưới dạng một thuộc tính protected và có hàm `GetKey()` tương ứng

```
public bool RegisterFromDatabase(Dummy.DomainObject ob
j)
        {
            if (obj == null)
            {
                return false;
            }
            identityMap.Add(obj.getKey(), obj);
            return true;
        }
```

Chúng ta đồng thời cũng cung cấp 2 hàm: Kiểm tra một đối tượng có key tương ứng đã được định danh trong Dictionary hay chưa và "lấy" một đối tượng có key tương ứng ra khỏi Dictionary này.

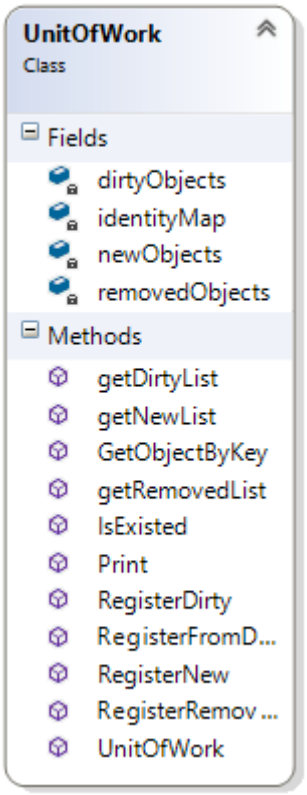
```
public bool IsExisted(object key)
public object GetObjectByKey(object key)
```

Tất cả việc yêu cầu một đối tượng nào đó sẽ đều thông qua phương thức `GetInstance(key)`

```
public T GetInstance(object key)
        {
            if (unitOfWork.IsExisted(key))
            {
                return (T)unitOfWork.GetObjectByKey(ke
y);
            }
            T res = new T();
            PropertyInfo propertyInfo = res.GetType()
                   .GetProperty(primaryKeyField);
            propertyInfo.SetValue(res, key);
            return res;
        }
```

Chúng ta sẽ kiểm tra đối tượng này đã tồn tại trong bộ nhớ hay chưa, nếu có trả về chính nó, nếu không mới thực hiện việc tạo mới đối tượng.

3.5. *Unit of work*



Unit of work là một đối tượng được xây dựng nhằm duy trì một danh sách các đối tượng bị "ảnh hưởng" trong suốt quá trình thực hiện các thao tác của người dùng. Nó đồng thời thực hiện việc định hướng các thao tác cập nhật các sự thay đổi lên phía CSDL.

Khi một đối tượng được lấy về từ CSDL, điều quan trọng là ghi nhận được các sự thay đổi của đối tượng. Việc không ghi nhận sẽ dẫn đến hàng loạt các lệnh SQL được gọi một cách không cần thiết. Thay vào đó, hãy giao việc này cho *Unit of work* quản lý và nó sẽ cho ta biết đến cuối cùng, những gì cần phải làm để lưu các thao tác của người dùng vào Database.

Có ba thứ chúng ta cần quan tâm khi xử lý các vấn đề liên quan đến sự thay đổi trong CSDL, bao gồm: đối tượng mới được tạo, các đối tượng đã có sẵn và cần cập nhật, các đối tượng cần phải bị loại bỏ khỏi CSDL. Do đó, trong framework này, *Unit of Work* sẽ được cài đặt dưới dạng 3 danh sách:

```
private Dictionary<object, object> newObjects =
new Dictionary<object, object>();
private Dictionary<object, object> dirtyObjects =
new Dictionary<object, object>();
private Dictionary<object, object> removedObjects =
new Dictionary<object, object>();
```

Một loạt các phương thức như `RegisterNew`, `RegisterRemoved` , `RegisterDirty` với tham số truyền vào là `DomainObject obj` sẽ được cài đặt, các phương thức `set` của từng đối tượng sẽ được cài lại và đối tượng sẽ "chủ động" gọi các hàm tương ứng nhằm ghi nhận chúng vào từng danh sách.

Sau khi đã hoàn thành, người dùng gọi lệnh commit sẽ thực hiện việc việc "cập nhật" các thao tác như thêm mới, chỉnh sửa và xóa lên CSDL.

Khi người dùng gọi lệnh `Commit` các thao tác sẽ được thực hiện như mã nguồn sau đây (các thao tác khác tương tự - việc thực hiện truy vấn SQL sẽ giao cho đối tượng `Action` thực hiện)

```
updateAction = new TableModifier.UpdateAction(_sqlDriver,
columns, tableName, GetPrimaryKeyName());

updateAction.Execute(unitOfWork.getDirtyList())
```

3.6. Lazy loading

Có nhiều kỹ thuật có thể được sử dụng để cài đặt *Lazy Loading* như Lazy *Initialization* (các trường sẽ gán bằng null trừ Key và khi truy cập đến thuộc tính nếu trường nào bằng null sẽ được yêu cầu đọc Database và điền vào trường còn thiếu – nhược điểm phải phân biệt được đâu là null thật, đâu là null do chưa tải dữ liệu), Kỹ thuật Virtual Proxy (tạo đối tượng ảo - mẫu Proxy trong GoF) hay sử dụng Value Holder. Các cách tiếp cận này sẽ có hạn chế là người dùng chỉ có thể tiếp cận với "phiên bản giả" của đối tượng chứ không phải đối tượng gốc. Do đó nhóm đã thực hiện cài đặt theo cách sử dụng GHOST.

Tất cả các đối tượng sử dụng framework đều phải kế thừa lớp `Domain Object`. Các lớp con này sẽ được thừa hưởng các phương thức và thuộc tính quản lý tình trạng, đọc và ghi dữ liệu của lớp cha này.

Một đối tượng sẽ được quản lý bằng key (phân biệt đối với mỗi đối tượng) và status (trạng thái của đối tượng) gồm GHOST (chỉ chứa giá trị key, các trường còn lại không có giá trị) và LOADED (các trường được load đầy đủ dữ liệu từ Database).

Như vậy, mỗi đối tượng được sinh ra đều ở trạng thái GHOST. Khi có bất kỳ một thao tác nào truy xuất đến các thuộc tính của đối tượng, framework sẽ kiểm tra đối tượng có cần thực hiện việc tải dữ liệu từ Database hay không. Nếu cần thiết thì framework sẽ thực hiện việc các công việc đó và đổi trạng thái thành LOADED.Lưu ý rằng đối tượng sẽ là người "chủ động" thực hiện việc tải dữ liệu cho chính nó khi được gọi đến (override lại các phương thức Get trong các trường thuộc tính)

Lazy Loading trong framework được cài đặt như sau:

Đầu tiên, các phương thức `get` và `set` của các thuộc tính trong đối tượng sẽ được cài đặt lại, ví dụ như:

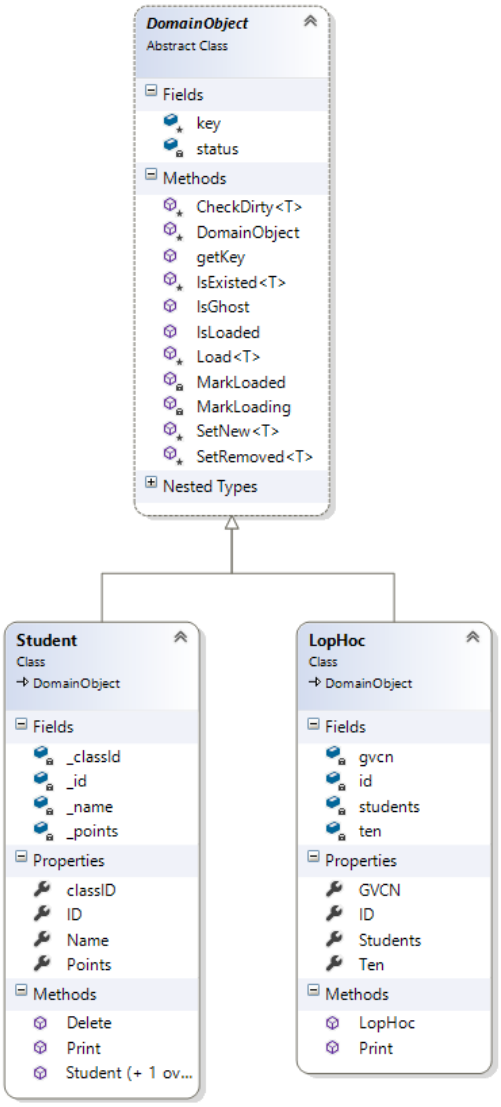
```
public float Points
   {
       get
       {
           Load<Student>();
           return _points;
       }
       set
       {
           CheckDirty<Student>();
           _points = value; ;
       }
   }
```

Để ý rằng, hàm `Load<Student>()` sẽ được thực hiện mỗi khi có bất kỳ truy cập nào đến thuộc tính của đối tượng.

Hàm `Load` ứng với bảng chứa đối tượng sẽ được gọi để thực hiện việc đưa dữ liệu vào các trường của đối tượng. Khi đó:

```
protected void Load<T>() where T : class, new()
        {
            if (IsGhost())
            {
                MarkLoading();
                ORMFramework.Session.getCurSession()
                 .Table<T>().DoLoadLine(this);
                MarkLoaded();
              }
           }
```

Nếu đối tượng này là "Ghost", nó sẽ gọi hàm `DoLoadLine(this)` với ý nghĩa đọc các trường từ CSDL ứng với đối tượng `this` này, truyền các giá trị tương ứng cho chúng, gán đối tượng này ở trạng thái `LOADED` và kết thúc. Người dùng sau đó có thể ghi nhận giá trị tương ứng của đối tượng này mà không cần thực hiện việc đọc lại từ CSDL.



3.7. Tổng hợp câu truy vấn SQL

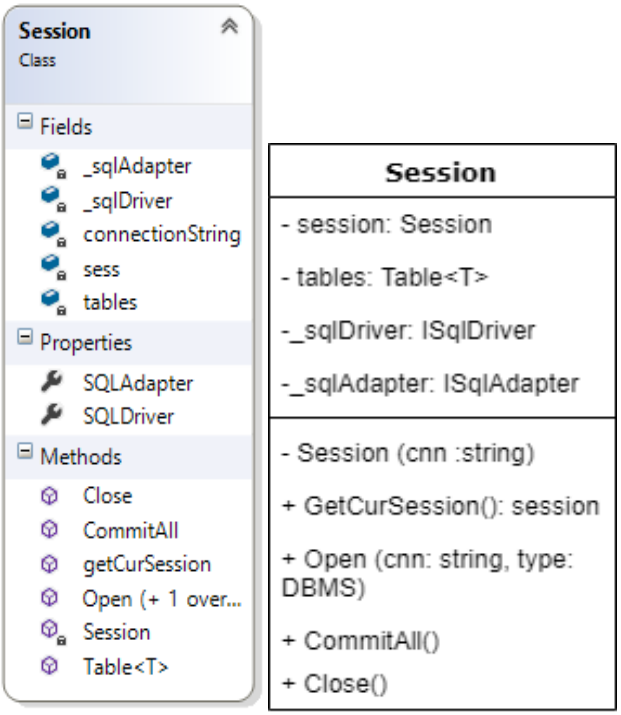
Lớp đối tượng `SQL Builder` với các lớp kế thừa tương ứng sẽ phụ trách việc tổng hợp tổng hợp câu truy vấn SQL, các câu lệnh được tổng hợp sẽ là các câu lệnh query chứ không phải các câu lệnh yêu cầu chỉnh sửa CSDL. Bên trong lớp `SQL Builder` chứa một danh sách các component với bản chất là các chuỗi ký tự, khi `SqlLam` cần sẽ thêm vào thông tin (ví dụ như lấy các trường nào trong CSDL hay thêm vào các điều kiện, các câu lệnh having, join), ...

Một đối tượng thuộc ISqlAdapter cũng được lưu trữ để đảm bảo việc tổng hợp đúng với cú pháp của hệ quản trị CSDL SQL.

Sau khi thực hiện xong, câu lệnh tổng hợp sẽ được gọi và các thành phần được tổng hợp theo đúng thứ tự.

4. Các mẫu thiết kế hướng đối tượng đã sử dụng trong framework

4.1. Mẫu Singleton – quản lý đối tượng Session

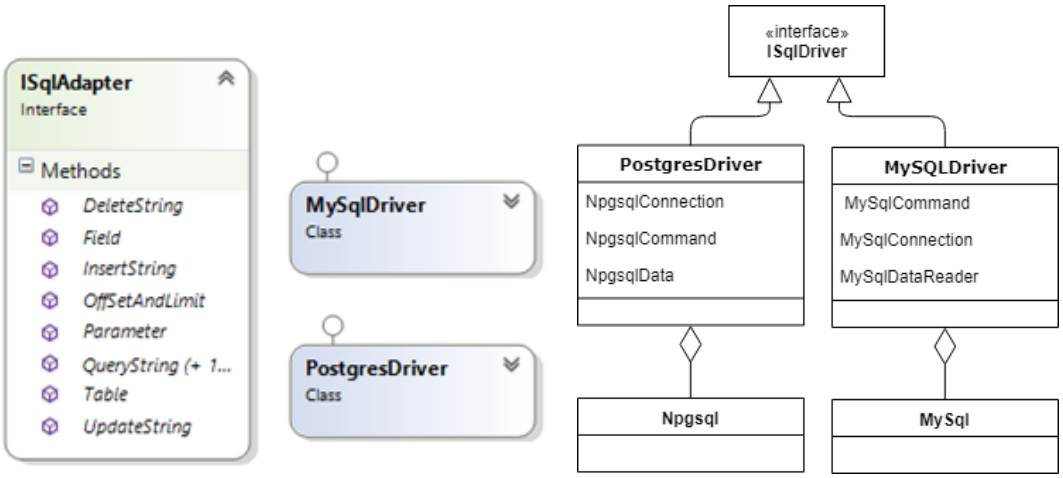


Đối tượng `Session` sẽ quản lý các bảng dữ liệu, các đối tượng kết nối với `Database`, đối tượng tổng hợp tạo câu truy vấn SQL ứng với từng Session.
Vì các đối tượng sẽ là người "chủ động - trực tiếp" yêu cầu lấy dữ liệu từ CSDL, các đối tượng do đó chúng cần được cung cấp một đối tượng Session duy nhất và thống nhất trong toàn bộ chương trình.
Các đối tượng quản lý việc kết nối với CSDL, tổng hợp câu truy vấn SQL cũng phụ thuộc nhiều vào hệ quản trị CSDL đang sử dụng và cần sự thống nhất.
Do đó, lựa chọn mẫu Singleton nhằm kiểm soát việc truy cập vào đối tượng, đảm bảo tính thống nhất và nhất. quán trong toàn bộ chương trình.

Mẫu Singleton được cài đặt thông qua việc đặt private cho phương thức khởi tạo, người dùng sử dụng Session thông qua phương thức Open và Close.
Ngoài ra, sđể kiểm soát việc một lớp đối tượng chỉ có tồn tại duy nhất một đối tượng *Table Gateway* tương ứng, một Dictionary cũng được cài đặt để quản lý điều này:

```
public Table<T> Table<T>() where T : class, new()
        {
            if (tables.ContainsKey(typeof(T))) {
                return (Table<T>) tables[typeof(T)];
            }
            Table<T> newTable = new Table<T>();
            tables.Add(typeof(T), newTable);
            return newTable;
        }
```

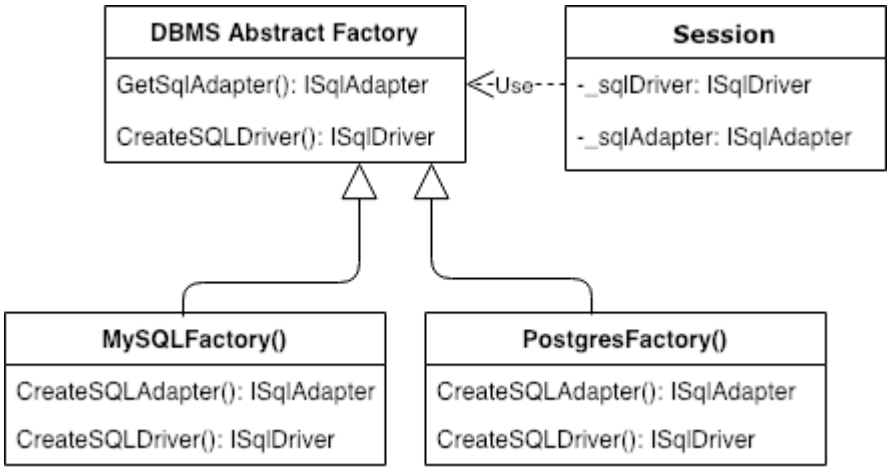
4.2. *Mẫu Adapter – quản lý các đối tượng thực hiện kết nối với Database*



Các SQL Driver sẽ quản lý việc thực thi các câu lệnh SQL (gửi tới CSDL, nhận kết quả trả về, ...) Mẫu *Object Adapter* được sử dụng để tránh việc chương trình phụ thuộc vào những lớp đối tượng cụ thể này, thay vào đó chỉ làm việc với interface `ISqlDriver`. Điều này cũng đảm bảo khả năng mở rộng về sau khi muốn thêm nhiều đối tượng quản lý mới phục vụ cho nhiều hệ quản trị CSDL khác nhau.

Các đối tượng `Npgsql` hay `MySql` là các đối tượng được cung cấp sẵn, ta sẽ tạo ra các lớp `PostgresDriver` và `MySQLDriver` để chứa các đối tượng đó và tạo ra các phương thức tuân theo interface chung.

4.3. *Mẫu Abstract Factory – tạo các đối tượng thực hiện việc kết nối với CSDL và tạo các câu truy vấn SQL*

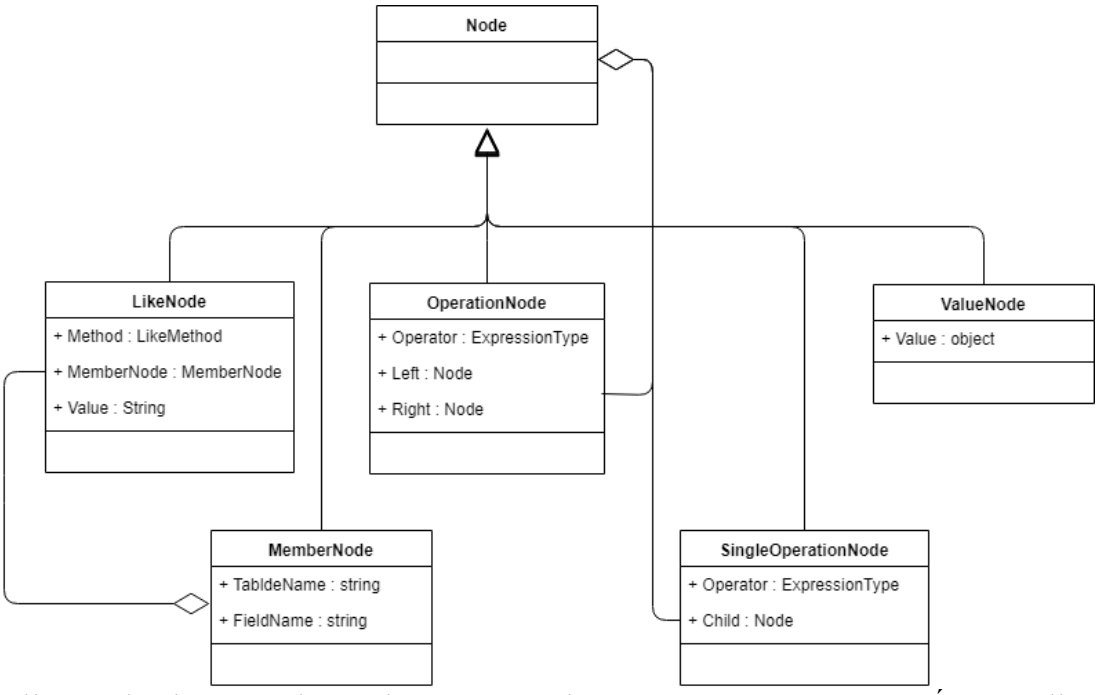


Mẫu Abstract Factory được sử để đảm bảo tính đồng bộ khi muốn tạo ra một "bộ" gồm các đối tượng thuộc nhiều loại tương ứng. Hai đối tượng cần phải có khi sử dụng framework là `ISqlAdapter` giúp tạo các câu truy vấn SQL và `ISqlDriver` quản lý việc kết nối với CSDL. Hai đối tượng này phụ thuộc vào riêng từng hệ quản trị CSDL khác nhau, và yêu cầu chúng phải luôn "đồng bộ".

4.4. Mẫu Composite – phân giải các Lambda Expression trong việc tạo lập câu truy vấn:

Việc phân giải, đọc ngữ nghĩa của Lambda Expression trong C# nằm ngoài phạm vi của đồ án này do đó thiết kế này được tham khảo từ project Lambda-SQL-Builder của tác giả Domany Dousan (https://github.com/DomanyDusan/lambda-sql-builder/tree/master/LambdaSqlBuilder)

Việc phân giải các Lambda Expression tương tự như việc phân giải biểu thức toán học, các biểu thức lambda sẽ được phân giải thành các Node tương ứng và từ nhóm sẽ tiếp tục xây dựng các câu truy vấn SQL dựa trên expression Tree (các node liên kết với nhau theo hình thức composition)



Like Node chứa các thông tin mô tả cú pháp LIKE trong câu truy vấn SQL liên quan đến tìm kiếm chuỗi

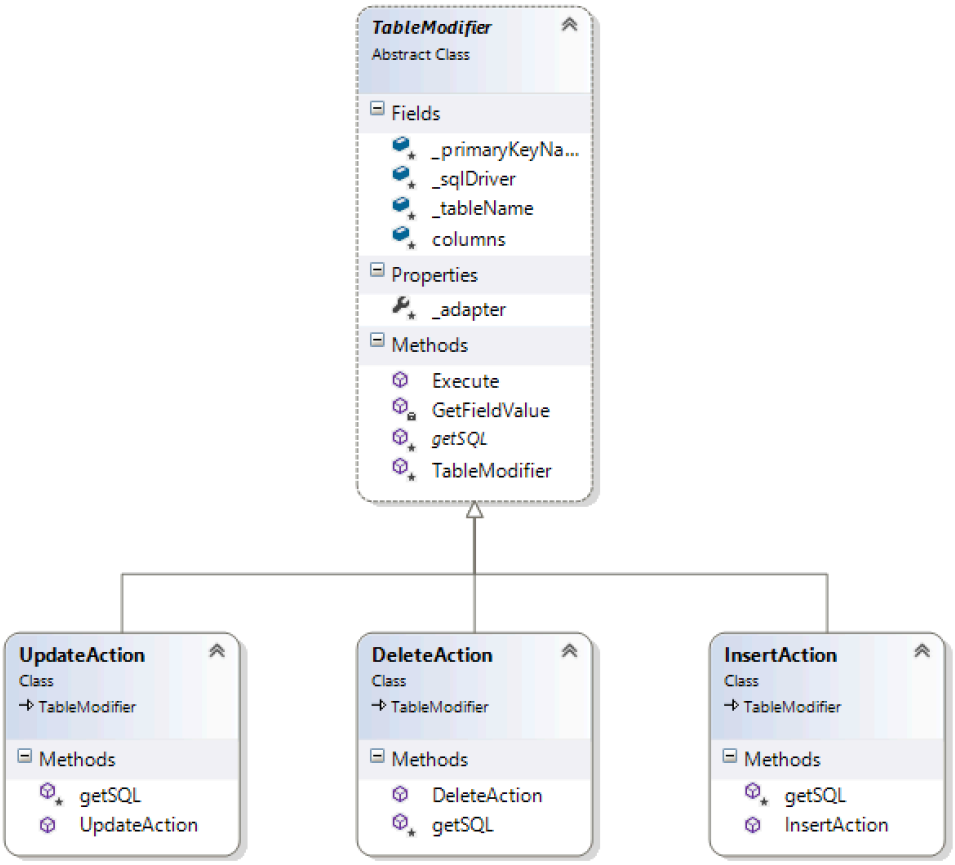
MemberNode quan tâm đến việc truy xuất các cột và bảng trong câu truy vấn SQL

Operation Node bao gồm Operator và hai Node trái – phải, ví dụ như các phép toán, phép so sánh, ...

ValueNode được dùng để lưu dữ liệu gồm các giá trị (số, chuỗi, ...) được truyền dưới dạng params trong câu truy vấn SQL.

4.5. Mẫu Template Method – tạo các đối tượng thực hiện thao tác chỉnh sửa trên bảng CSDL:

Các thao tác như Insert, Update và Delete có trình tự thực hiện gần như tương tự nhau, chỉ khác nhau về cú pháp của câu lệnh SQL. Hơn nữa chúng đều được tổ chức đồng nhất dưới dạng các danh sách của Unit of work nên chúng có rất nhiều bước thực hiện giống nhau và tương đồng.



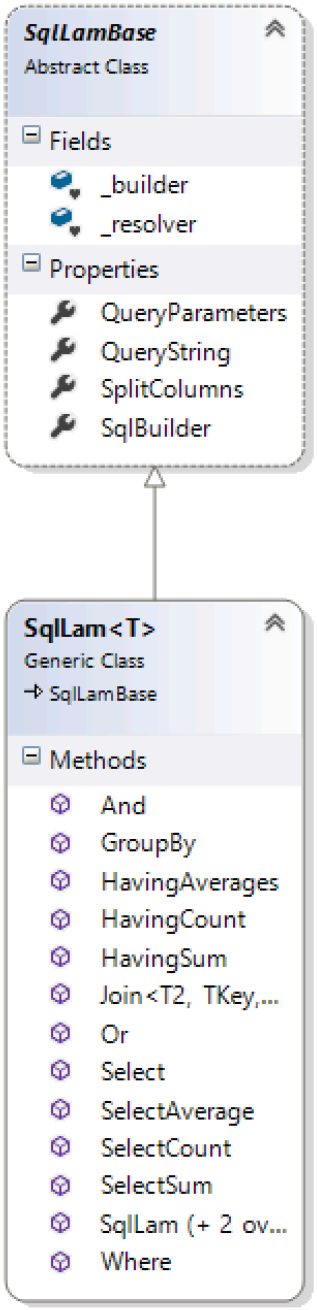
Hàm Execute thực hiện việc gửi câu lệnh tương ứng lên CSDL để thực hiện có nội dung như sau:

```
public bool Execute(object obj)
        {
            if (obj == null)
            {
                return false;
            }
            Dictionary<string, string> fieldValues = GetFieldVa
lue(obj);

            if (fieldValues == null)
            {
                return false;
            }
            string sql = getSQL(fieldValues);
            return _sqlDriver.ExecuteNonQuery(sql);
        }
```

Các thao tác tuân theo một "trình tự cụ thể": gồm kiểm tra đối tượng khác null sau đó đọc các trường thuộc tính và giá trị tương ứng của chúng, cuối cùng tạo câu truy vấn SQL.

Thao tác tạo câu truy vấn SQL là khác nhau đối với mỗi thao tác (Update, Delete hay Insert), do đó phương thức `getSQL(filedValues)` sẽ là phương thức cho phép các lớp con có thể ghi đè lên một cài đặt khác, nhưng vẫn đảm bảo quy trình là không đổi. Mẫu Template Method được sử dụng và phát huy tác dụng trong trường hợp này.



4.6. Mẫu Builder – tạo các câu truy vấn SQL

Khi người dùng muốn tạo một câu truy vấn với việc sử dụng sql-lambda-builder, họ có thể tạo ra một câu truy vấn phức tạp thông qua các câu lệnh đơn giản tác động lên thuộc tính của chúng. Người dùng có thể thêm vào thao tác như select, having, where, ...

Mẫu Builder được sử dụng trong đối tượng `SqlLam`

Một số ví dụ về cài đặt mẫu builder trong đối tượng:

```
public SqlLam<T> GroupBy
(Expression<Func<T, object>> expression)
        {
            _resolver.GroupBy(expression);
            return this;
        }

 public SqlLam<T> SelectCount
(Expression<Func<T, object>>
 expression)
        {
            _resolver.SelectWithFunction(e
xpression, SelectFunction.COUNT);
            return this;
        }
```

5. Tài liệu tham khảo:

[1] **Pattern of Enterprise Application Architecture**, Martin Fowler With contributions from David Rice, Matthew Foemmel, Edward Hieatt, Robert Mee, and Randy Stafford.

[2] **Các hướng dẫn về framework Entity, Dapper** từ https://www.tutorialspoint.com

[3] **Project SQL Lambda Builder,** DomanyDusan https://github.com/DomanyDusan/lambda-sql-builder/tree/master/LambdaSqlBuilder

[4] **Kỹ thuật Reflection trong C#**, https://yinyangit.wordpress.com/2011/01/12/c-ki-thuat-reflection-trong-net/?fbclid=IwAR3a2RGg0rw5ILH8JYBQnPUteScOdrmC9pluYnQsk42P0Tls9RtcWGXFl3o