**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**  
**Database Access Management Framework**  
**Mẫu thiết kế hướng đối tượng và ứng dụng– 18KTPM1**

| thực hiện bởi |

**18127006 – Hồ Nguyễn Huy Hoàng**

**18127092 – Phạm Vũ Duy**

**18127099 – Nguyễn Đức Hiếu**

**18127118 - Mai Đăng Khánh**

# **MỤC LỤC**

[**MỤC LỤC** 1](#_Toc92489170)

[**CLASS DIAGRAM** 2](#_Toc92489171)

[**a)** **Ý nghĩa các lớp đối tượng** 2](#_Toc92489172)

[**b)** **Các mẫu thiết kế được sử dụng** 3](#_Toc92489173)

[1) Bridge 3](#_Toc92489174)

[2) Builder 3](#_Toc92489175)

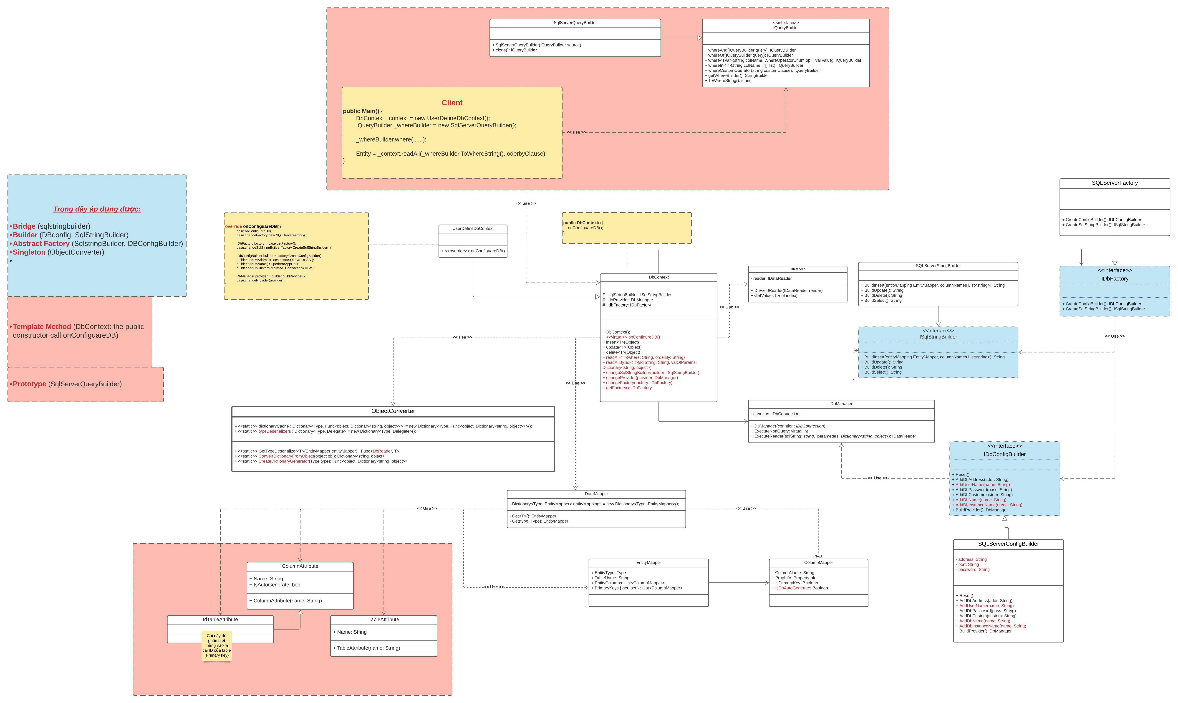
[3) Abstract Factory 5](#_Toc92489176)

[4) Singleton 5](#_Toc92489177)

[5) Template Method 5](#_Toc92489178)

[6) Prototype 6](#_Toc92489179)

# **CLASS DIAGRAM**

****

**1. Sơ đồ lớp của Framework (" ORM Class Diagram.jpeg")**

# **Ý nghĩa các lớp đối tượng**

* **DBContext:** đóng vai trò giao diện của framework cho người dùng. Chứa các hàm dùng để khai báo kết nối đến cơ sở dữ liệu và các thao tác chính trên cơ sở dữ liệu như: select, insert, update, delete, ….Có chứa thuộc tính DBManager dùng để xác định cơ sở dữ liệu được truy cập, chứa thuộc tính SQLStringBuilder dùng để tạo chuỗi câu lệnh SQL thích hợp cho cơ sở dữ liệu được truy cập.
* **DBManager:** chứa thông tin kết nối đến một cơ sở dữ liệu nhất định và các chức năng cho phép tương tác với cơ sở dữ liệu nêu trên như thực hiện query được cung cấp bởi các SQLStringBuilder.
* **SQLServerStringBuilder:** xây dựng và cung cấp các query theo cú pháp của SQL Server.
* **SQLServerConfigBuilder:** xây dựng một kết nối đến SQL server, có chứa các hàm để thiết lập các thuộc tính của kết nối như: đường dẫn, tài khoản, mật khẩu, port, …
* **DBReader:** cung cấp các hàm cho phép DBContext đọc dữ liệu từ cơ sở dữ liệu.
* **ObjectConverter:** lớp này thực hiện các chức năng chuyển đổi các trường dữ liệu SQL thành các đối tượng C#.
* **DataMapper:** cung cấp các chức năng cho phép DBContext chuyển đổi các lớp C# thành các bảng sử dụng EntityMapper và chuyển đổi các thuộc tính của lớp C# nêu trên thành các cột sử dụng ColumnMapper.
* **EntityMapper:** lớp này đóng vai trò là giao diện của một bảng cơ sở dữ liệu cho ứng dụng. Có chứa các thông tin của một bảng như tên bảng, danh sách các thuộc tính (cột) và các khóa chính của bảng.
* **ColumnMapper:** lớp này đóng vai trò là giao diện của một thuộc tính (cột) cơ sở dữ liệu cho ứng dụng. Có chứa các thông tin của một cột như tên cột, đặc tính của cột (kiểu dữ liệu, ràng buộc), nhận dạng khóa chính, nhận dạng được khởi tạo tự động.

# **Các mẫu thiết kế được sử dụng**

## **Bridge**

* **Class diagram:**

Diagram

Description automatically generated

* **Source code**

Text

Description automatically generated

* **Ý nghĩa:**

+ Giảm sự phụ thuộc giữa interface và implement (có thể thay đổi implementation lúc runtime).

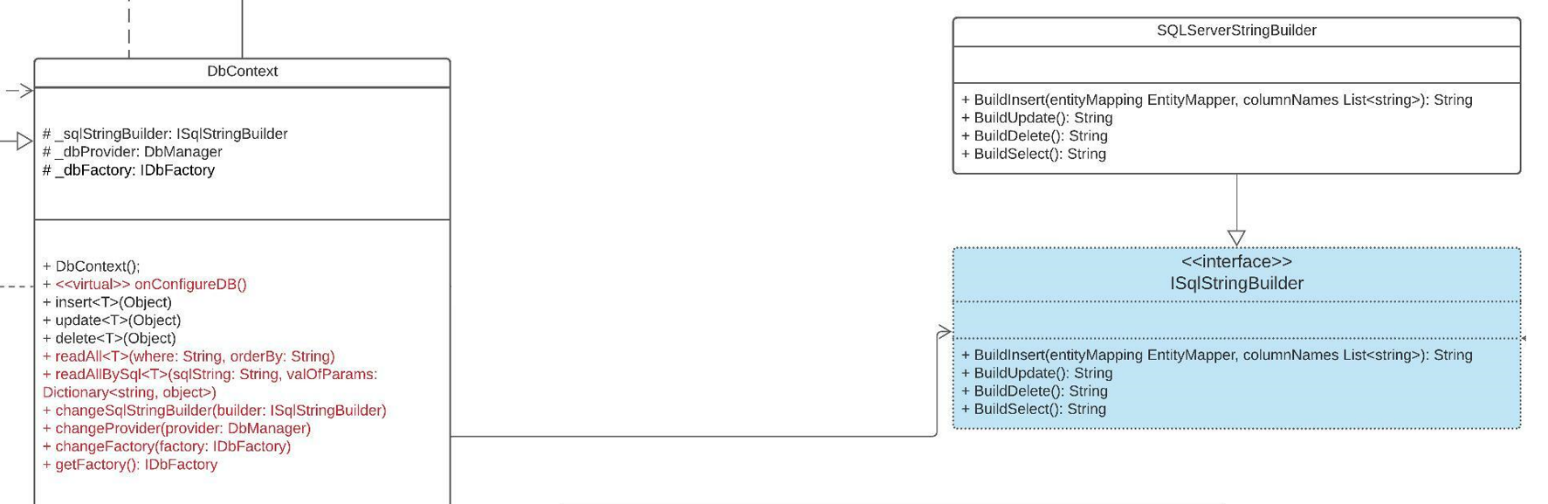
+ Dễ dàng mở rộng về sau. Sau này nếu Framework có hỗ trợ thêm SQLite, MySQL,… thì chỉ cần implement interface IsqlStringBuilder

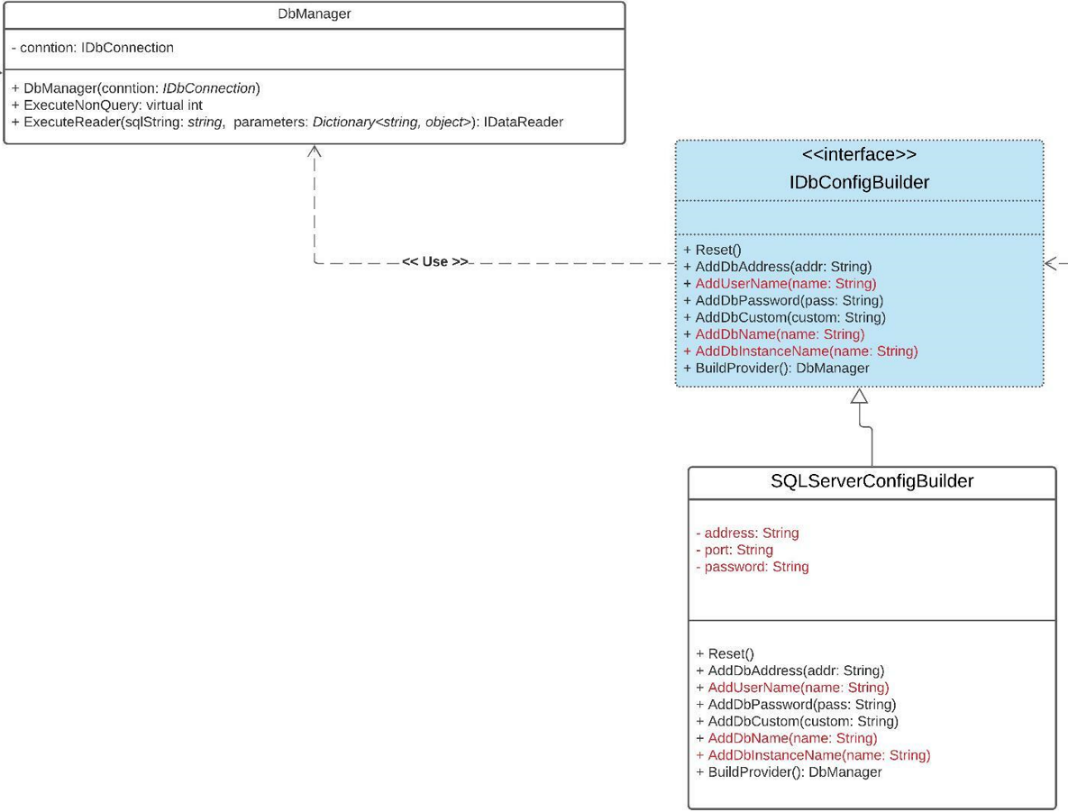
+ Code sạch sẽ, gọn gàng

+ Dễ dàng nâng cấp mở rộng và bảo trì về sau.

## **Builder**

* **Class diagram:**



****

* **Source code:**

var entityMapper = DataMapper.Get<T>();

var valOfParams = ObjectConverter.ConvertDictionaryFromObject(data);

var sql = \_sqlStringBuilder.BuildInsert(entityMapper, valOfParams.Keys.ToList());

return \_dbProvider.ExecuteNonQuery(sql, valOfParams);

public string BuildInsert(EntityMapper entityMapper, List<string> columnNames)

{

StringBuilder sqlString = new StringBuilder();

sqlString.Append("INSERT INTO ").Append(entityMapper.TableName).Append(" (");

//Convert columnNames to string

var columns = entityMapper.EntityColumns

.Where(m => m.IsDbAutoGenerate == false &&

columnNames.Contains(m.ColumnName, StringComparer.OrdinalIgnoreCase))

.ToArray();

for (int i = 0; i < columns.Length; i++)

{

var member = columns[i];

if (i > 0)

sqlString.Append(", ");

sqlString.Append(member.ColumnName);

}

//Pass param

sqlString.Append(") VALUES (");

for (int i = 0; i < columns.Length; i++)

{

var member = columns[i];

if (i > 0)

sqlString.Append(", ");

sqlString.Append("@").Append(member.ColumnName);

}

sqlString.Append(")");

return sqlString.ToString();

}

* **Ý nghĩa:**

**+** Đối với SQL String Builder: Mô hình hóa các câu lệnh SQL. Cho phép người dùng (DBContext) tạo các chuỗi ký tự mô tả câu lệnh SQL tương ứng với loại cơ sở dữ liệu và nhu cầu. Đơn giản hóa viết các câu lệnh query.

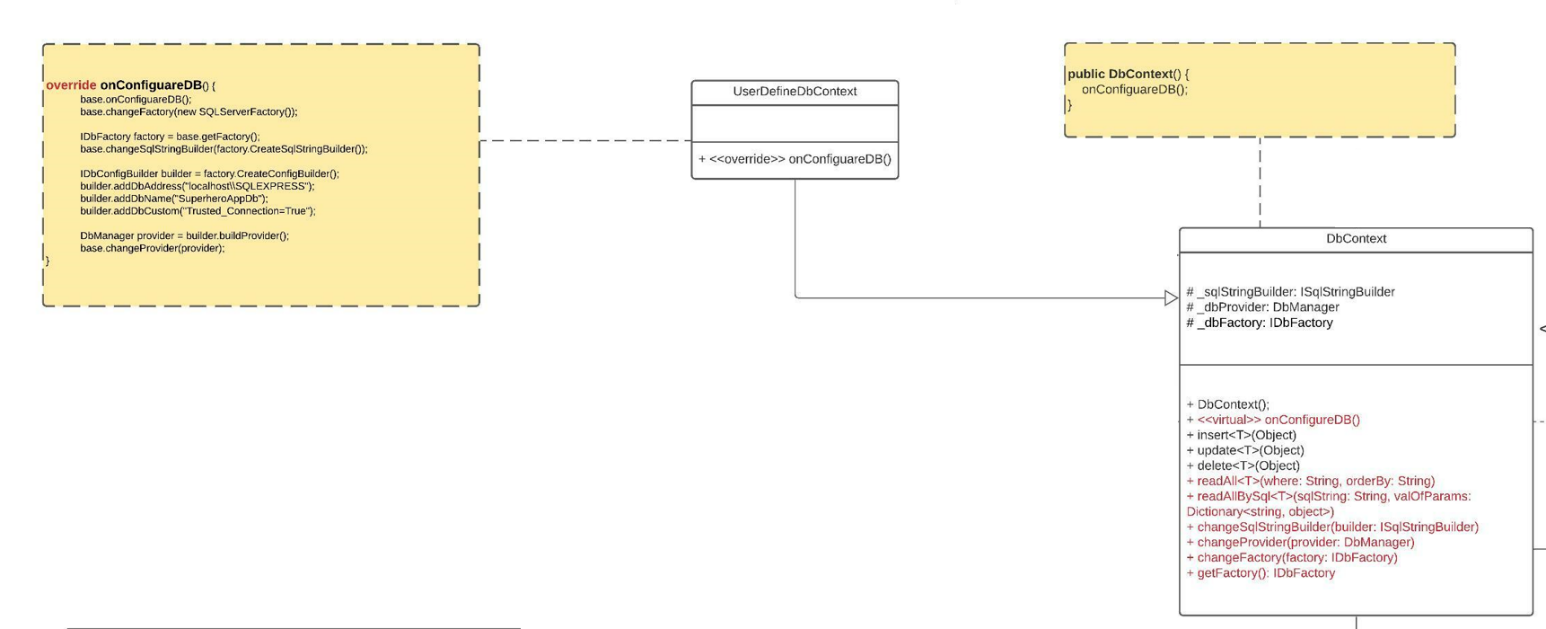
**+** Đối với DB Config Builder: Cho phép DBManager tạo một chuỗi ký tự chứa các thông tin về một kết nối cơ sở dữ liệu. Do một đường đẫn kết nối đến database khá dài và có thể chứa nhiều thông tin (mật khẩu, port, …) nên khả năng cao sẽ phát sinh lỗi, vì vậy khi sử dụng mẫu thiết kế Builder ta có thể rời rạc hóa việc tạo một kết nối đến Database.

## **Abstract Factory**

## **Singleton**

## **Template Method**

* **Class diagram:**

****

* **Source code:**

void onConfiguareDB()

{

base.onConfiguareDB();

base.changeFactory(new SQLServerFactory());

IDbFactory factory = base.getFactory();

base.changeSqlStringBuilder(factory.CreateSqlStringBuilder());

IDbConfigBuilder builder = factory.CreateConfigBuilder();

builder.addDbAddress("localhost\\SQLEXPRESS");

builder.addDbName("SuperheroAppDb");

builder.addDbCustom("Trusted\_Connection=True");

DbManager provider = builder.buildProvider();

base.changeProvider(provider);

}

* **Ý nghĩa:** Template method được sử dụng ở class DBContext, cụ thể ở hàm onConfigureDB() cho phép các class người dùng tự định nghĩa kế thừa từ DBContext có thể override hàm onConfigureDB(). Việc này cho phép người dùng có thể tự chủ trong việc thiết lập kết nối đến cơ sở dữ liệu của họ và hạn chế việc lập lại code đã được cài đặt ở DBContext

## **Prototype**

* **Class diagram:**

**Graphical user interface, diagram

Description automatically generated**

* **Source code:**

IQueryBuilder queryWhere1 = new SqlServerQueryBuilder();

queryWhere1.where("Type", WhereOperatorEnum.Equals, 2);

IQueryBuilder queryWhere2 = new SqlServerQueryBuilder();

queryWhere2.where("Id", WhereOperatorEnum.GreaterThan, 4);

IQueryBuilder queryWhere3 = new SqlServerQueryBuilder();

queryWhere3.whereIn("HeroId", new int[] { 1, 3 });

string whereString = queryWhere1.whereAnd(queryWhere2)

.whereOr(queryWhere3)

.ToWhereString();

public IQueryBuilder whereOr(IQueryBuilder query)

{

if (this.\_whereStringBuilder.Length > 0)

{

var leftWhere = new StringBuilder();

leftWhere.Append(\_whereStringBuilder);

this.\_whereStringBuilder.Clear();

this.\_whereStringBuilder.Append("(");

this.\_whereStringBuilder.Append(leftWhere);

this.\_whereStringBuilder.Append(")");

this.\_whereStringBuilder.Append(" OR ");

var rightWhere = query.getWhereBuilder();

this.\_whereStringBuilder.Append("(");

this.\_whereStringBuilder.Append(rightWhere);

this.\_whereStringBuilder.Append(")");

} else

{

this.\_whereStringBuilder.Append(query.getWhereBuilder());

}

return clone();

}

* **Ý nghĩa:** Mẫu Prototype được dùng để thực thi lồng các điều kiện where SQL. Cho phép một khởi tạo một chuỗi chứa nhiều điều kiện kép từ các chuỗi chứa điều kiện đơn. Chuỗi kép sẽ sao chép từ các chuỗi đơn.