РЕШЕНИЕ ПРОБЛЕМЫ ВЗАИМОДЕЙСТВИЯ ПРИЛОЖЕНИЯ С БАЗОЙ ДАННЫХ, ИМЕЮЩЕЙ ДИНАМИЧЕСКУЮ СХЕМУ

К.А. Гаврилов
(г. Томск, Томский политехнический университет)
E-mail: kgavrilov_kz@mail.ru

INTERACTION BETWEEN .NET APPLICATION AND DATABASE WITH DYNAMIC SCHEMA

K.A. Gavrilov
(Tomsk, Tomsk Polytechnic University)

Ways to create mapping between .Net application and Database with dynamic schema
Keywords: Dynamic database schema, LINQ, LINQ to SQL, ORM, .Net.

Схема реляционной базы данных (от англ. Database schema) — структура базы данных, описанная на формальном языке, поддерживаемом СУБД, определяющая таблицы, поля в каждой таблице (обычно с указанием их названия, типа, обязательности) и ограничения целостности (первичный, потенциальные и внешние ключи и другие ограничения) [1].

Обычно при разработке приложений схема базы данных постоянна во времени и определяется на этапе проектирования. Традиционным способом взаимодействия клиентского приложения с базой является создание сущностей, являющихся отражением таблиц реляционной СУБД в объектно-ориентированной среде. Этот способ очень удобен, так как существует большое количество ORM фрэймворков, автоматически преобразующих данные объектно-ориентированной среды в сущности таблиц БД.

Однако существует класс задач, для решения которых необходимо изменять схему БД (например, добавлять новые таблицы). При этом требуется вносить изменения и в структуру приложения (добавлять новые сущности), иначе взаимодействие с базой будет некорректным. Ввиду того что изменение клиента может оказаться нетривиальной задачей, которая потребует больших временных и материальных затрат, необходимо использовать такой подход к проектированию приложения, при котором разрабатываемое приложение не будет зависеть от заранее спроектированной схемы. Для решения данной проблемы необходима такая ORM, которая позволяет взаимодействовать со схемой, созданной не на момент компиляции приложения, а во время его выполнения.

Если рассматривать уже существующие решения, то нельзя не отметить, что изначально эти ORM проектировались под статическую схему базы данных. Применение динамических схем в большинстве ORM либо невозможно, либо достаточно тяжело в использовании. При этом нельзя забывать, что это накладывает дополнительные временные расходы, на и без того не самую быструю работу ORM. При этом нельзя с уверенностью сказать, что ORM поддерживает LINQ полностью, что может стать дополнительной неприятной неожиданностью.

Рассмотрим ведущие свободно распространяемые ORM для платформы .NET (Entity FrameWork, NHibernate и Telerik Data Access) и проанализируем, насколько их функционал подходит для решения поставленной задачи [2].

При использовании NHibernate для решения данной задачи, необходимо выполнить следующие действия:

1. Создать таблицу в БД.
2. Создать класс в приложении.
3. Создать XML файл, в котором требуется явно указать маппинг объектов таблицы БД в бизнес-объекты.

Таким образом, данная ORM не способна легко решить поставленную задачу. Для её корректной работы необходимо все изменения БД вносить в приложение. Возможно, есть

вариант «на лету» создавать динамический класс и переписывать маппинг в XML-файле, но тогда возникает проблема актуальности записей (ведь после создания такой записи таблица может быть неоднократно отредактирована).

Entity Framework предоставляет 3 способа проектирования базы данных: Database-First, Model-First, Code-First.

При Database-First подходе с помощью различных инструментов создаётся БД, а затем генерируется EDMX-модель БД, которая предоставляет удобный графический интерфейс для взаимодействия с базой данных в виде диаграмм и объектную модель в виде классов C#.

В Model-First сначала создаётся графическая модель EDMX в Visual Studio (в фоновом режиме создаются классы C# модели), а затем, на основе этой модели, генерируется БД.

Code-First заключается в ручной настройке классов объектной модели (данный подход поддерживает как генерацию сущностных классов из существующей базы данных, так и создание базы данных из созданной вручную модели объектов C#).

Однако ни один из подходов также не предоставляет возможности привязывать таблицы БД к динамическим сущностям объектной модели. К тому же EF имеет несколько крайне неприятных особенностей поведения: простая вставка 500 записей в таблицу БД может занимать порядка 30 секунд.

Telerik Data Access поддерживает добавление типов или полей для привязывания даже во время выполнения. Существует возможность добавлять новые классы, новые свойства существующих классов, сопоставлять их со столбцами базы данных и начать использовать их сразу без каких-либо дополнительных настроек [3].

Однако для каждого столбца необходимо прописать собственный маппинг, так как в основе работы TDA всё также лежит работа с классами, созданными на момент компиляции, и для добавления новой сущности потребуется переписать класс маппинга, а значит – пересобрать приложение. Таким образом, данная ORM не подходит для решения поставленной задачи.

Разработка собственной библиотеки, реализующей работу с динамической схемой базы данных. Ещё одним вариантом реализации необходимого функционала является динамическое создание класса в момент запроса приложения к БД. Такой подход является достаточно нетривиальным в разработке, однако позволяет создать динамическую схему с минимальной привязкой к клиенту.

Основная логика реализации состоит из следующих шагов:

1. Создать одну достаточно простую по своей структуре таблицу, содержащую схемы доступа, имена и «Заглавный атрибут» - имя колонки, из которой будут браться данные при использовании динамической таблицы в качестве внешнего ключа. Соответственно, при добавлении новой таблицы в БД достаточно просто добавить новую строку в «dbo.Таблицы». Клиентское приложение будет знать только об этой таблице, схема которой не будет меняться со временем.

	ID	Schema	Name	TitleAttribute
1	7FD25417-5051-4446-9483-66911C2C986C	dbo	Справочник	Название
2	647D5821-B8CE-4553-A00A-D53F1CAE72A2	dbo	Объект1	ID

Рисунок 1. Контент таблицы «dbo.Таблицы»

2. На основе данных о таблице (схема доступа-имя) получить полную схему нужной таблицы.
3. Исходя из полученной схемы, в момент запроса сгенерировать динамический класс, являющийся сущностью .Net, связанной с сущностью БД.
4. Использовать интерфейс IQueryable в связке с Dynamic LINQ to SQL для серверной фильтрации данных динамического типа из БД.

```
DynamicData = dc.GetTable(tableType).AsQueryable();
```

На данный момент разработан прототип данного решения.

Пользователь может выбрать таблицу БД, а приложение выведет её на экран. Это достигается за счёт крайне простого кода, приведённого ниже, где в выпадающем списке (ComboBox) представлен список записей таблицы «dbo.Таблицы», а для отображения данных используется привязка к IQueryable объекту DynamicData.

```
<StackPanel Grid.Row="0">
            <ComboBox Width="100" HorizontalAlignment="Left" Margin="5"
                      ItemsSource="{Binding TableTypes}"
                      DisplayMemberPath="Name"
                      SelectedItem="{Binding SelectedTableType,
                      UpdateSourceTrigger=PropertyChanged}"/>
        </StackPanel>
        <telerik:RadGridView Grid.Row="1" Margin="5"
                             ShowGroupPanel="False"
                             AutoGenerateColumns="True"
                             ItemsSource="{Binding DynamicData,
                             UpdateSourceTrigger=PropertyChanged}"/>
```

Созданный прототип полностью работоспособен и показывает отличную скорость работы, однако на данный момент есть список нерешённых задач, которые хотелось бы разрешить:

1. Не реализована работа с колонкой «Заглавный атрибут», поэтому при использовании внешних ключей отображается список GUID-значений, которые являются неинформативными для пользователя. Решением данной задачи является использование операторов join при формировании запросов. Существует 2 варианта использования таких операторов. Первый сводится к тому, чтобы строить динамический класс не для таблицы, а непосредственно для запроса. Однако в данном случае возникает проблема одинаковых названий колонок для разных таблиц. Второй способ – это написание собственного LINQ провайдера, способного добавлять в исходных запрос динамические LINQ операторы.
2. Существует проблема с составными именами таблиц и колонок (из-за ограничений среды .Net для имён переменных/классов невозможно создать класс, состоящий из 2 слов).
3. Необходимо реализовать больше LINQ запросов (в перспективе реализовать все наиболее востребованные).

ЛИТЕРАТУРА

1. Object-relational mapping. [Электронный ресурс]. Режим доступа: *https://en.wikipedia.org/wiki/Database_schema*, свободный.

2. SlideShare. [Электронный ресурс]. Режим доступа: *http://www.slideshare.net/ptsukanov/orm-net-nhibernate-linq-to-sql-entity-framework*, свободный.

3. Telerik Data Access. Powerful Runtime Capabilities. [Электронный ресурс]. Режим доступа: *http://www.telerik.com/data-access/runtime-features*, свободный.