УДК 681.3

СРЕДСТВА КОДОГЕНЕРАЦИИ ДЛЯ ВЗАИМОДЕЙСТВИЯ С БАЗОЙ ДАННЫХ ЧЕРЕЗ ОБЪЕКТЫ

И.А. Лихацкий

Институт программных систем НАН Украина
03680, Киев, проспект Академика Глушкова,40, корп. 5
igor_md@ukr.net

Предложена методика нахождения эффективного способа хранения объектных данных, направленная на достижение наибольшей производительности генерируемого кода и труда программиста. Описана реализация этой методики в созданной автором системе C-Gen. Приведена краткая характеристика и основные возможности системы C-Gen.

There was suggested the most effective method of storing object data, that is aimed at reaching the best performance of generated code and programmer work. There was described implementation of this methods in the system C-Gen, created by the author. There was given short description and main features of the C-Gen system.

Введение

В объектно-ориентированном программировании (ООП) [1] объекты в программе описывают объекты из реального мира. В качестве примера можно рассмотреть список работников некого предприятия, который содержит список сотрудников с такими данными как имя, фамилия, адрес проживания, телефон, отдел, заработная плата. В терминах ООП они будут представляться объектами класса «Сотрудник», которые будут содержать следующий список полей: имя, фамилия, адрес проживания, телефон, заработная плата (рис. 1).

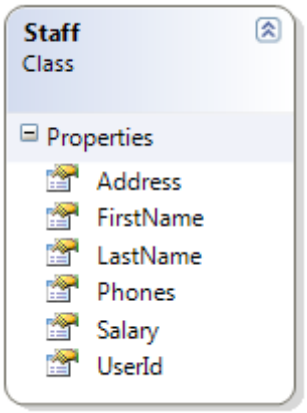

Рис. 1. Класс "Сотрудник"

Суть проблемы состоит в преобразовании таких объектов в форму, в которой они могут быть сохранены в файлах или базах данных, и которые легко могут быть извлечены в последующем, с сохранением свойств объектов и отношений между ними. Эти объекты называются «хранимыми». Существует несколько подходов к решению этой задачи.

В качестве решения проблемы хранения данных может быть использована реляционная система управления базами данных (РСУБД) [2]. Но использование РСУБД для хранения объектно-ориентированных данных (ООД) приводит к значительным накладным расходам по написанию программного обеспечения [3], которое должно обеспечить обработку ООД в таком виде, чтобы уметь сохранить эти данные в реляционной форме.

РСУБД используют набор таблиц, для представления данных. Связная информация хранится в других таблицах. Между этими таблицами, как правило, существуют связи (один-ко-многим, один-к-одному, многие-ко-многим). Один объект может быть представлен в РСУБД как несколько связных таблиц. Например, в вышерассмотренном примере, для хранения данных могут быть использованы две таблицы, таблица с сотрудниками и таблица с телефонами (рис. 2) (допустим, что у сотрудников может быть несколько телефонных номеров).

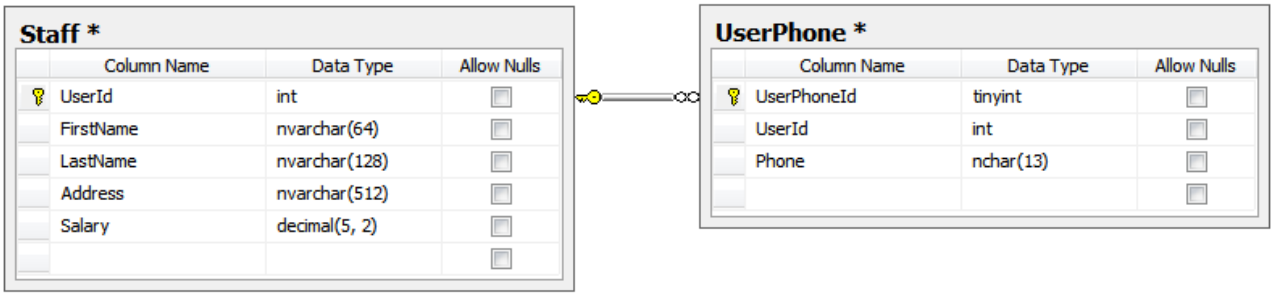

Рис. 2. Реляционное представление объекта "Сотрудники" в БД

Так как системы РСУБД обычно не реализуют реляционного представления физического уровня связей, выполнение нескольких последовательных запросов (относящихся к одной «объектно-ориентированной» структуре данных) может быть слишком ресурсоемко.

Представление сущностей в БД сильно отличается от представления сущностей в объектно-ориентированных языках программирования (ООЯП). Это явление называют несоответствием парадигмы (paradigm mismatch) [4]. Несоответствие порождает ряд проблем таких как:

1. **Проблема гранулярности**. Допустим, что в примере с заказами адрес покупателя нужно хранить не как простую строку, а как отдельный объект со своими свойствами и методами (рис. 3). У адреса может быть выделен индекс, страна, город, улица.

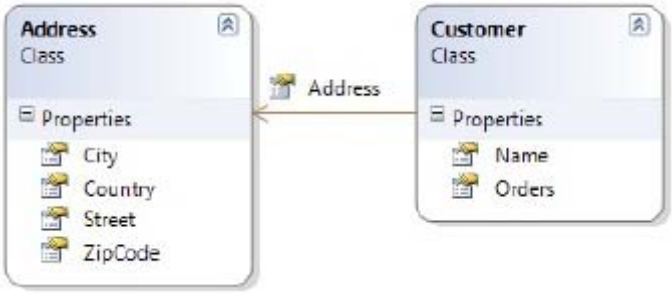

Рис. 3. Диаграмма класса "Покупатель и его адрес"

2. **Проблема поддержки наследования и полиморфизма**. ООЯП поддерживают такое понятие, как наследование, а в РСУБД его нет. В нашем примере может существовать несколько видов покупателей: физическое лицо и юридическое лицо (рис. 4).

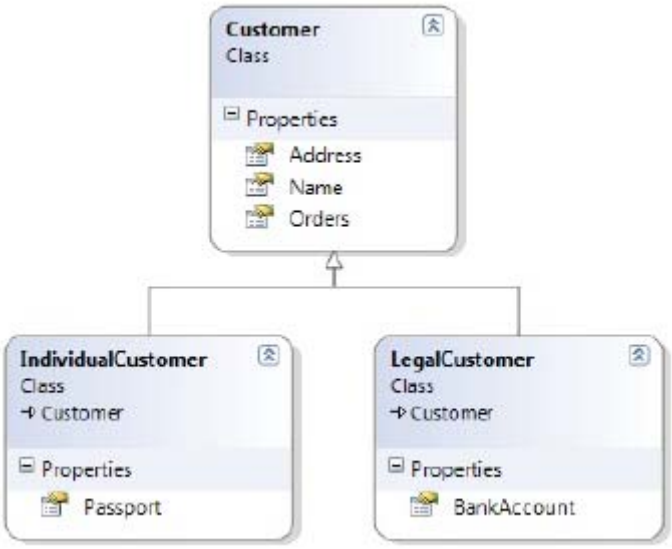

Рис. 4. Иерархия класса Customer

На стороне сервера БД мы должны выполнять полиморфные запросы, т. е. при запросе объектов класса Customer должны быть найдены и объекты его наследников IndividualCustomer и LegalCustomer соответственно.

3. **Проблема идентичности.** Создаваемые объекты предметной области необходимо идентифицировать. Идентичность в БД задается с помощью первичного ключа, в то время как идентичность объекта в памяти не является эквивалентом первичного ключа.

4. **Проблемы связанные с ассоциациями**. В ООЯП ассоциация представлена в виде объектных ссылок, а в РСУБД в виде внешних ключей. Проблема заключается в соединении полностью открытой модели данных, которая не зависит от нашего приложения с навигационной моделью данных нашего специфического приложения.

Одним из путей решения проблемы ООД к данным является использование подхода "**Ручного кодирования слоя сохраняемости**". Данный способ позволяет получить полный контроль над процессом сохранения/загрузки объектов. Главным недостатком способа является большой объем рутинной работы, как на этапе первичной реализации, так и на этапе сопровождения. Однако здесь можно добиться максимального уровня производительности и задействовать все необходимые возможности конкретной СУБД.

Другим путем решения проблемы является использование **объектно-реляционного отображения (ORM)** – это автоматическое сохранение объектов языка программирования в таблицы реляционной базы данных с использованием метаданных, которые описывают это отображение [4].

ORM решения обладают богатыми возможностями по сохранению и загрузке объектов. Допускается работа, как с одиночными объектами, так и целыми графами связанных объектов. Существует большое количество различных ORM решений, более 60 [5, 6]. Различные решения накладывают различные ограничения на классы нашей предметной области. К таким ограничениям можно отнести:

- требования наследования от определенного базового класса;
- реализация определенного интерфейса;
- ограничения на типы коллекций и ассоциаций;
- использование сгенерированных классов.

Большинство ORM средств самостоятельно генерируют SQL-запросы к БД. Пользователь максимально изолирован от подробностей работы с БД. ORM средства учитывают особенности конкретной СУБД. Многие ORM-средства позволяют выполнять отображение классов предметной области на различные СУБД с учетом индивидуальных особенностей.

Таким образом использование ORM технологий дает нам следующие преимущества:

- простота использования. Мы оперируем объектами, сгенерированными ORM системой, не заботясь о написании SQL запросов;
- независимость от производителя. Существует возможность смены поставщика РСУБД с минимальными издержками;
- удобство сопровождения. Стандартные решения проще поддерживать в долгосрочной перспективе.

Из недостатков стоит отметить следующее:

- неоптимальный SQL. Поскольку ORM системы сами генерируют SQL он не всегда является оптимальным;
- неиспользуемый код. ORM системы генерируют функциональность, которая может быть не использована в приложении.

Еще один путь решения проблемы это использование объектных и объектно-ориентированных БД(ООБД), таких как Cache [7]. Cache предлагает способ интеграции реляционного и объектного представлений на основе единой архитектуры данных. В рамках этой архитектуры существует единое описание объектов и таблиц, отображаемых непосредственно в многомерные структуры ядра БД, ориентированного на обработку транзакций. Таким образом, любой класс Cache может быть представлен как таблица, и любая таблица – как класс. Встроенные в Cache ООЯП Cache ObjectScript и Cache Basic полностью поддерживают объектный доступ к данным. В Cache поддерживаются наследование (в том числе и множественное), инкапсуляция и полиморфизм, встраиваемые объекты, ссылки, коллекции, отношения и т. д.

Однако в результате применения объектно-ориентированного подхода разработчики сильно ограничены в выборе средств проектирования и разработки. Таким образом современные РСУБД отличает совершенство реализаций, обеспечивающее высокую надежность и производительность, в отличие от ООБД, находящихся сегодня на ранней стадии своего развития. Более того, существуют задачи [8], для которых объектный подход не дает преимуществ.

Рассмотрев основные подходы решения задачи сохранения объектных данных можно поставить основную задачу: нахождения эффективного способа хранения объектных данных направленную на достижение максимальной производительности и автоматизацию процесса, а также минимальные затраты. Исходя из поставленной задачи определим методологию ее решения.

1. Для реализации "слоя сохраняемости нашего приложения" необходимо использовать средства кодогенерации, и свести долю написания повторяющегося кода в ручную к минимуму.

2. В качестве источника хранения данных использовать РСУБД, так как они сочетают в себе достаточно высокую надежность и производительность.

3. На основе метаданных БД, необходимо генерировать объекты и классы для доступа к данным из кода приложения.
4. Обеспечения возможности переопределения/дополнения логики работы сгенерированного кода без внесения изменений в структуру приложения.
5. Обеспечение безопасного взаимодействия с БД. Подразумевает наличие встроенных механизмов проверки данных перед отправкой их на выполнение.
6. Обеспечить взаимодействия с БД через строго типизированные объекты, чтобы исключить возможность возникновения ошибок выполнения приложения, связанных с запросами к БД, к нулю.
7. Обеспечить возможность частичной генерации, предоставив разработчику возможность выбора, для каких сущностей БД должны быть сгенерированы объекты, и какие именно.
8. Обеспечить возможность генерации кода на разных выходных языках программирования.

На основе описанной методологии автором предложена реализация системы C-Gen, которая решает задачу нахождения эффективного способа хранения объектных данных ориентированную на максимальную производительность и автоматизацию процесса, а также минимальные затраты.

Существует ряд программных систем, таких как .netTiers Application Framework [11], которые частично решают поставленную задачу. Однако данная система не лишена недостатков. Из основных недостатков можно выделить:

- система не обеспечивает взаимодействия с базой данных только через типизированные объекты. Для ряда хранимых процедур генерируются не типизированные объекты DataReader или DataSet;
- нет возможности генерировать код на разных языках программирования.

Система кодогенерации C-Gen

В данной статье, в качестве решения проблемы кодогенерации, автором предложена система автоматического генерирования программного кода C-Gen.

В общем, систему C-Gen можно классифицировать как систему генерирующую программный код, которая на входе получает структуру БД. Система является однонаправленной, т. е. генерация программного кода происходит только на основе структуры БД.

Кодогенерация – это процесс создания кода, будь то человеком или машиной. Та форма кодогенерации, про которую будет идти речь, лучше назвать **автоматическим программирование**[9] т. е. использованием неких механизмов, которые создают компьютерную программу при минимальном вмешательстве человека.

Использование технологии автоматическим программирование уместно там где используется большого количества кода с повторяющимися структурами. В нашем случае, в качестве входных данных мы получаем БД с набором таблиц, представлений, и хранимых процедур, которые по своей структуре, по сути, являются однотипными объектами.

Преимущества использования технологии автоматического программирования по сравнению с программированием вручную дает следующие преимущества:

1. Экономия времени, и следовательно денежных затрат на разработку.
2. Сгенерированный код более надежный, в нем легче производить глобальные изменения, легче его тестировать.
3. Автоматическое программирование также позволяет лучше управлять изменениями. Чтобы исправить глобальную ошибку, или расширить логику работы приложения, вам достаточно внести изменения в одном месте.
4. С использование автоматического программирования легче контролировать стиль кодирования, т.к. он задается централизованно.

В качестве инструмента для кодогенерации будем использовать шаблоны T4 [10]. Текстовый шаблон T4 представляет собой сочетание блоков текста и логики управления, которое может создать текстовый файл. Логика управления представляет собой фрагменты программного кода в Visual C# или Visual Basic. Созданный файл может представлять собой текст любого вида, например веб-страницу, файл ресурсов или исходный программный код на любом языке.

В нашем случае мы будем использовать текстовые шаблоны T4 времени разработки [10], которые позволяют создавать программный код и другие файлы. Как правило, шаблоны создаются для того, чтобы менять код, генерируемый на основе данных из модели. Модель – это файл или БД, которая содержит ключевые сведения о требованиях используемого приложения. В нашем случае в качестве модели используются метаданные полученные из БД, которая поступает на вход нашей системе. Например, шаблон хранимой процедуры Delete (рис. 5).

Взаимодействие с БД

Вернемся к модели данных на основе которой генерируется программный код. Как уже было сказано, в нашем случае в качестве модели у нас выступают метаданные полученные из БД. Давайте поближе рассмотрим взаимодействие системы C-Gen с пользовательской БД.

БД, которая предоставляется системе для генерации, должна быть спроектирована разработчиком. То есть вся ответственность за оптимальность построения БД, правильности выбранных индексов, связей между таблицами ложится на разработчика. Так как наша система ориентирована на максимальную

производительность и минимальные затраты был выбран именно этот подход. Преимуществами данного подхода является тот факт, что разработчик имеет полный доступ к средствам оптимизации БД, а также полностью контролирует процесс проектирования и создания БД, используя оптимальные средства и методы, что на выходе дает более качественных sql-код. Также такая система потребляет меньше серверных ресурсов.

Что касается хранимых процедур, то здесь дела обстоят следующим образом: для достижения оптимальной работы сервера БД, а также максимальной автоматизации программирования, в системе C-Gen было принято решение разделить хранимые процедуры на два типа:

- простые хранимые процедуры. Это те хранимые процедуры, структура которых является одинаковой для каждой сущности базы данных. К таким хранимым процедурам относятся CRUD хранимые процедуры. Такие процедуры можно сгенерировать автоматически;
- сложные хранимые процедуры. К данному классу можно отнести те хранимые процедуры, автоматическая генерация которых не всегда является оптимальной. Разработка таких хранимых процедур полностью контролируется программистом.

```
create procedure <#= string.Format("[{0}].[{1}_Delete]", SchemaName, Tbl.Name)#>
<#
    PushIndent("\t");
    List<string> pk = new List<string>();
    foreach (Column column in Tbl.Columns)
    {
        if (column.InPrimaryKey)
        {
            if (pk.Count == 0)
            {
                Write(String.Format("@{0} {1}", column.Name, column.DataType.Name));
                pk.Add(column.Name);
            }
            else
            {
                WriteLine(",");
                WriteLine(String.Format("@{0} {1}", column.Name, column.DataType.Name));
                pk.Add(column.Name);
            }
        }
    }
    PopIndent();
#>

AS
SET NOCOUNT ON

delete from <#= string.Format("[{0}].[{1}]", SchemaName, Tbl.Name) #>
where
<#
    PushIndent("\t");
    bool isFirst = true;
    foreach (string str in pk)
    {
        if (isFirst)
        {
            WriteLine(String.Format(" {0} = @{0}", str));
            isFirst = false;
        }
        else
        {
            WriteLine(String.Format("and {0} = @{0}", str));
        }
    }
    PopIndent();

    return this.GenerationEnvironment.ToString();
#>
```

Рис. 5. T4 шаблон хранимой процедуры Delete

Таким образом, в следствии автоматической генерации простые хранимых процедур, мы избавляем разработчика от выполнения рутинной работы по написанию их вручную. В то же время, мы оставляем за разработчиком возможность реализации сложных хранимых процедур, позволяя оптимизировать производительность сервера БД.

В качестве примера создадим БД, с таблицей User, которая описывает пользователей некой системы. После генерации в системе C-Gen мы получим набор простых хранимых процедур (рис. 6)

Такой набор хранимых процедур будет сгенерирован для каждой таблицы в БД. Для представлений генерируется всего две простые хранимые процедуры: GetTop и GetPage соответственно.

Что касается хранимых процедур, которые пишет разработчик, то для каждой такой процедуры, система кодогенерации C-Gen сгенерирует типизированный объект, который будет отображать результирующий набор данных, возвращаемых хранимой процедурой и метод, по которому можно будет взаимодействовать с этой хранимой процедурой из кода.

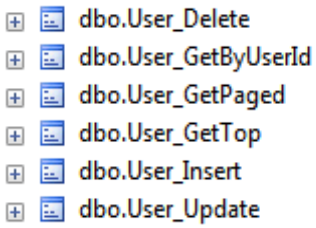

Рис. 6. Простые хранимые процедуры

На примере таблиц Staff и UserPhone создадим хранимую процедуру, которая будет выбирать всех пользователей и их телефоны, и отображать следующие данные (FirstName, LastName, Phone) (рис. 7).

```
create procedure [_Staff_UserPhoneList]
select
    f.FirstName,
    f.LastName,
    p.Phone
from Staff as f
inner join UserPhone p on f.UserId = p.UserId
```

Рис. 7. Листинг хранимой процедуры

После перегенерации проекта система C-Gen сгенерирует класс UserPhoneList (рис. 8) и метод UserPhoneList() для получения данных. Таким образом, взаимодействие с объектами из базы данных осуществляется через типизированные объекты, которые доступны сразу после генерации. Также существует возможность получить результат выполнения хранимой процедуры в виде объекта DataReader.

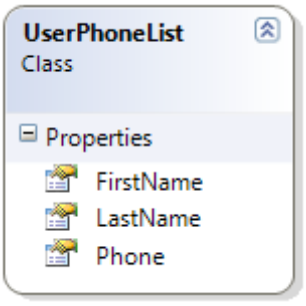

Рис. 8. Структура класса UserPhoneList

Следует также отметить, что наличие встроенных механизмов проверки данных на sql-инекции в системе C-Gen, перед отправкой их на выполнение обеспечивает безопасное взаимодействие с БД.

Частичная генерация

Частичная генерация является решением проблемы создания генератором неиспользуемого кода. Рассмотрим пример, когда в БД существует таблица логирования, в которой должны фиксироваться данные о

входе пользователя в систему. Мы знаем, что в таблицу нужно только записывать данные, а также предоставлять данные из нее пользователям для статистики. То есть потенциально будет использоваться всего две хранимые процедуры Insert – добавления данных в таблицу и GetPaged() – получение данных из таблицы, с возможностью постраничного отображения и накладывания фильтров. Остальные четыре процедуры использоваться не будут, а вместе с ними не будут и использоваться методы, сгенерированные в коде.

Частичная генерация, позволяет произвести тонкую настройку процесса генерации, определив лишь те сущности и тот функционал, который необходим разработчику в данном конкретном случае. Таким образом решается проблема генерации неиспользуемого кода.

Переопределение дополнительной логики в сгенерированных классах

Как мы уже говорили системе C-Gen генерирует классы и методы для взаимодействия с базой данных. Однако в некоторых случаях возникает необходимость определить дополнительный функционал для сгенерированных классов либо переопределить логику одного из сгенерированных методов. В системе C-Gen предусмотрена возможность генерации классов прослоек, которые наследуются от сгенерированного системой класса (рис. 9).

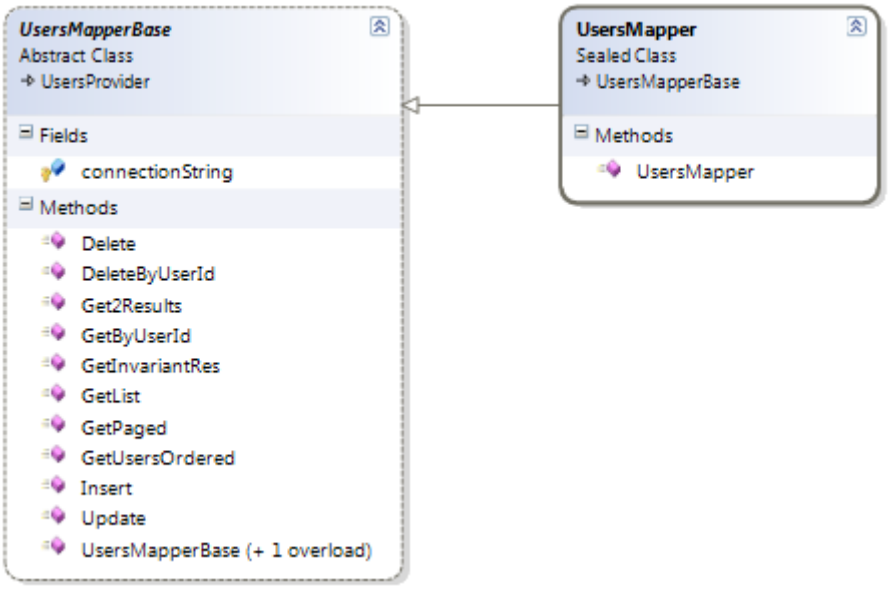

Рис. 9. Структура класса UserMapperBase и UserMapper

Класс UserMapperBase содержит реализацию методов для доступа к БД, и перезаписывается после каждой пере генерации. Класс UserMapper наследуется от класса UserMapperBase. Именно в классе UserMapper можно определить дополнительный функционал, который не будет перезаписан при перегенерации. Таким образом система является очень гибкой и масштабируемой.

Выводы

Рассмотрены существующие подходы к решению задачи обеспечения взаимодействия с базой данных в ООП. Первый подход использование ORM технологий, избавляет разработчика от написания какого-либо кода по обеспечению взаимодействие с базой данных, предоставляя средства для взаимодействия на уровне типизированных объектов, генерируя нужные запросы в базу данных и следя за правильность их выполнения. Однако, за такую "простоту" в использовании приходится платить накладными расходами связанными с генерацией неоптимального sql-кода, что снижает производительность приложения в целом, и требует больших серверных ресурсов для обслуживания приложения. Также система не позволяет разработчику самостоятельно определять набор компонентов необходимых для генерации, что приводит к генерации "лишнего" кода, который никогда не будет исполнен.

Использование объектных и объектно-ориентированных СУБД позволяет решить проблему представления объектов в реляционной модели, т. к. реализует принципы ООП. Однако, такой подход, сильно ограничивает разработчика в выборе средств и технологий, а также сами объектно-ориентированных СУБД сегодня находятся на ранней стадии своего развития, и в сравнении с современные реляционные СУБД не отличаются совершенством реализаций, и не обеспечивают достаточно высокую надежность и производительность.

На основе рассмотренных подходов и методологий решения задачи сохранения объектных данных, была поставлена основная задача нахождения эффективного способа хранения объектных данных ориентированную на максимальную производительность и автоматизацию процесса, а также минимальные затраты. Исходя из чего была разработана и предложена методология решения поставленной задачи.

В результате, на основе разработанной методологии, была спроектирована система кодогенерации C-Gen, которая на входе получает спроектированную разработчиком базу данных, а на выходе генерирует объекты и классы, с помощью которых можно взаимодействовать с базой данных. Система решает такие проблемы.

1. Система C-Gen, использует механизмы кодогенерации на основе текстовых шаблонов, что позволяет автоматизировать процесс генерации.
2. Использование текстовых шаблонов, позволяет получить любой выходной язык программирования в качестве результата генерации.
3. В качестве входных данных система получает базу данных, на основе которой генерируются объекты и классы, а также стандартный набор CRUD хранимых процедур, на стороне сервера. В результате мы избавляем программиста от нужды реализовывать однотипные хранимые процедуры (Insert, Update, Delete, GetByPrimaryId) для таблиц, сохраняя при этом, за разработчиком право реализовывать сложную логику выборки по своему усмотрению, самым оптимальным способом. Таким образом решается проблема неоптимального sql-кода в приложении.
4. Система обладает встроенными механизмами проверки валидности передаваемой информации на sql сервер в качестве параметров, что исключает возможность проведению sql-инъекций.
5. Система обеспечивает возможность взаимодействия с базой данных через строго типизированные объекты, что позволяет избежать появления ошибок исполнения, при взаимодействии приложения с базой данных.
6. Система является масштабируемой. Система может быть расширена путем добавления нового шаблона генерации.
7. Система обладает гибкой системой настроек, что позволяет, определить какие именно объекты необходимо сгенерировать системе, чтобы уменьшить количество неиспользуемого кода до минимума.

1. *Роганов Е.А*. Основы информатики и программирования. – М.: 2001 – 587с.
2. *Буч Г*. Объектно-ориентированный анализ и проектирование с примерами приложений, 3-е изд.: Пер. с англ. – М.: Издательский дом "Вильямс", 2008 – 987с.
3. *Гуйдо А. В*. Технологии программирования // Электронное издание – издательский центр Южно-Уральского государственного университета.
4. *Bauer C., King G*. Java Persistence with Hibernate. – New York: Manning, 2007. – 876 p.
5. *Object-relational* mapping vendors. – http://www.service-architecture.com/products/object-relational_mapping.html.
6. *List* of object-relational mapping software. – http://en.wikipedia.org/wiki/List_of_object-relational_mapping_software.
7. *Кирстен В., Иринге М., Рёриг В., Шульте П.* СУБД Cache, объектно-ориентированная разработка приложений. – СПб.: Питер, 2001. – 377 с.
8. *Moynihan T.* Objects Versus Functions in User-Validation of Requirements: Which Paradigm Works Best? Proceedings of the International Conference on Object Oriented Information Systems, 1994.
9. *Automatic* Inductive Programming ICML 2006 Tutorial http://www.evannai.inf.uc3m.es/et/icml06/aiptutorial.htm.
10. *Создание* кода и текстовые шаблоны T4 http://msdn.microsoft.com/ru-ru/library/bb126445.aspx.
11. *.netTiers* Application Framework – http://nettiers.com/