УДК 51:681.3

И.А. Лихацкий

О ТЕХНИКЕ РЕЛЯЦИОННО-ОБЪЕКТНОГО ПРЕОБРАЗОВАНИЯ

Данная статья посвящена методам отображения прикладных реляционных данных в объектную модель. В ней проводится анализ существующих решений, основанных на методике объектно-реляционного преобразования, а также выделения основных недостатков в использовании данной техники. Предложена техника реляционно-объектного преобразования, главная цель – это достижения высокой производительности и достаточного уровня автоматизации генерируемого кода, при взаимодействии с базой данных. В качестве примера реализации методики реляционно-объектного преобразования описана система кодогенерации C-Gen предложенная автором. Приведена краткая характеристика и основные возможности данной системы.

Введение

В условиях бурного развития информационных технологий и стремительной эволюции систем управления данными одной из актуальных задач, возникающей перед разработчиками приложений, является выбор системы управления базой данных (СУБД), в соответствии с особенностями решаемых прикладных задач хранения данных и эффективного управления ими. С одной стороны, СУБД должна поддерживать необходимую степень абстракции данных, а с другой – она должна быть ориентирована на структурные особенности их организации и характер использования.

В настоящее время широкое распространение получила объектно-ориентированная методология разработки прикладных систем, а в мире хранения данных доминируют реляционные СУБД. Учитывая целесообразное решение, оказывается использование промежуточного слоя программного обеспечения, предоставляющего необходимые объектно-ориентированные интерфейсы для доступа и управления данными, хранимые под управлением реляционной СУБД [1, 2]. Актуальность использования такого подхода сочетается с известными преимуществами реляционных СУБД, таких как:

- возможность поддержки наследуемых (legacy) систем, использующих традиционные решения;
- простота использования технологии, основанной на понятной табличной модели и математически строгой теории реляционной алгебры;
- широкое распространение и тщательная многолетняя апробация предлагаемых на рынке продуктов СУБД;
- предоставления естественных для приложений объектно-ориентированных интерфейсов реализованных для популярных языков программирования.

Существенно, что значительная часть функций по реализации указанных свойств ложится на промежуточный слой, а его создание сопряжено с комплексом проектных решений, затрагивающих сопровождаемость, производительность, простоту применения между клиентским приложением и сервером.

Сокрытие базы данных

С появлением на свет реляционных баз данных программистам стал доступен структурированный язык запросов SQL для доступа к данным, тем самым избавив их от необходимости знания ненужных деталей организации физического хранения данных. Наряду с этим оказалось, что большинство форматов данных, можно легко описать в виде таблиц и связей между ними, а наличие строгой математической теории, которая легла в основу технологии сложно переоценить. Таким образом, эти два фактора предопределили успех реляционных баз данных.

Реляционная и объектная модели ортогональны. Это значит, что они моде-

лируют одну и ту же сущность, но с разных сторон: реляционная модель акцентирует свое внимание на структуре и связях сущностей, объектная на их свойствах и поведении. Реляционная модель используется для информационного моделирования, выделения существенных для нас атрибутов, сохранения их значений и последующего поиска, обработки и анализа. Объектная модель, в большей степени, используется для моделирования поведения, выделения существенных для нас функций и последующего их использования [3]. На практике сложилась ситуация, когда информационные системы разрабатываются в основном с использованием ООП, тогда как данные хранятся в реляционном виде. Таким образом, существует необходимость в отображении объектов на реляционные структуры и наоборот. Данные методы активно развиваются в конце восьмидесятых годов и отражены в многочисленных публикациях [4–8].

Компонент программной системы, отвечающий за преобразования данных из объектного в реляционный вид, называется объектно-реляционным проектором (ORM).

В технологии отображения объектов на РСУБД существует очень важный момент, от понимания которого зависит успех разработки того или иного приложения. Бытует мнение о том, что для слоя сохраняемости (Persistence Layer), который генерируется по средствам ORM, генерируемый SQL код является аналогом трансляции языка высокого уровня в машинный код. Данное утверждение является ошибочным, а также может привести к созданию трудно-сопровождаемых систем, с потенциальными проблемами производительности.

Дело в том, что SQL – это высокоуровневый декларативный специализированный язык четвертого поколения, в отличие от Java и C#, относящихся к третьему поколению императивных языков. Так, один оператор SQL, выполняющий нечто посложнее нежели выборка по ключу, потребует для достижения того же результата на порядок больше строк на языке C# или Java [9].

Такая ситуация приводит разработчиков ORM к необходимости создавать собственный SQL-подобный язык для манипулирования объектами и уже его транслировать в SQL. К числу таких языков можно отнести HQL – Hibernate Query Language – SQL-подобный язык запросов, используемый в Hibernate/NHibernate [10], а также компонент .Net Framework – LINQ to SQL, предоставляющий инфраструктуру времени выполнения для управления реляционными данными как объектами [11]. Также существует возможность использования динамического преобразования результата SQL запроса в коллекцию объектов.

В противном случае пришлось бы извлекать большие массивы данных из базы данных и впоследствии обрабатывать их непосредственно в своем приложении. Примерно так же обрабатывались данные при отсутствии встроенного SQL языка. Такой подход называется навигационным подходом к манипулированию данными, а также является типичным для сетевых и иерархических СУБД [12]. Тем не менее, получив в распоряжение ORM, мы в какой-то мере возвращаемся к навигационным подходам обработки данных.

Таким образом, сложилась тенденция, в которой разработчики пытаются скрыть недостаток знаний РСУБД за дополнительным уровнем абстракций. Не смотря на предоставляемый ORM уровень абстракции, и сокращении времени разработки, заставить приложение эффективно работать с РСУБД, таким способом, без знания основ SQL практически невозможно.

Недостатки ORM

Используя ORM в качестве инструмента для обеспечения взаимодействия приложения с сервером баз данных, разработчики зачастую сталкиваются со следующими проблемами.

Как только разработчики реализовывают CRUD-логику, использовать SQL напрямую становится затруднительно. Это касается стратегий отображения данных и проблем переносимости приложения между СУБД. По сути, каждый запрос SQL к

СУБД является некой проекцией результата выборки на конкретный класс. Учитывая это этим разработчикам зачастую приходится использовать язык запросов ORM (если он есть). Зачастую такие языки не стандартны и не обладают средствами для отладки и профилирования. Например, в .NET начиная с версии 3.5, есть возможность использовать LINQ, который позволяет обнаружить некоторые ошибки на уровне компиляции.

В результате оказывается, что язык запросов ORM генерирует далеко не самый оптимальный SQL код. Чтобы решить данные проблемы, разработчики зачастую прибегают к частичной обработки данных внутри приложения: выбирают коллекции объектов и в циклах фильтруют или, используя тот же LINQ над обрабатываемым массивом, порождая новые запросы. Количество таких запросов к СУБД при такой обработке может исчисляться тысячами.

Техника реляционно-объектное представление как альтернатива ORM

Исходя из поставленных задач, была предложена концепция, основанная на построении высокопроизводительной, оптимальной структуры базы данных, которая описывает модель разрабатываемой информационной системы. Предоставляя разработчику полный доступ к управлению базой данных (построению индексов, написанию сложных SQL запросов, использованию представлений), мы решаем один из важнейших вопросов, связанных с производительностью. Таким образом, использование высокоуровневого декларативного специализированного языка SQL, позволит оптимизировать процесс выборки данных и производительность системы в целом, по сравнению с обработкой больших массивов данных в приложении, отчасти возвращаясь к навигационным подходам обработки данных. Естественно, что производительность системы будет зависеть от квалификации специалиста, отвечающего за проектирования структуры БД и написания запросов, но с другой стороны мы получаем полный контроль над этим процессом.

Исходя из вышесказанного, можно заключить, что для разработки высокопроизводительного приложения, без знания SQL не обойтись, а реализация сложных структур, таких как нетривиальные SQL запросы, или построение правильных индексов, плохо поддается автоматизации.

Однако существует один процесс, который все же может быть автоматизированным, это процесс написания CRUD хранимых процедур. Генерация CRUD хранимых процедур осуществляется автоматически для каждого объекта БД (таблицы, представления). Особенность данного типа хранимых процедур заключается в том, что они имеют четкую структуру, а процесс их генерации может быть легко автоматизирован [13].

В качестве установления соответствия между реляционной и объектной моделями используется методика «трех проекций» [12]. Данная методика описывает правила, по которым осуществляется преобразование из реляционной в объектно-ориентированную модель данных. Четкое выделение реляционных объектов, а также присущих им свойств, позволяет однозначно представить их в объектном виде.

Таким образом, процесс генерации кода уровня доступа к данным, является полностью автоматизированным. Разработчик получает полный набор средств API для доступа и управления данными, хранимыми под управлением реляционной СУБД.

Система кодогенерации C-Gen

Изложенная концепция реляционно-объектного представления была реализована в системе кодогенерации C-Gen. В данном разделе мы представим основные характеристики данной системы.

Система C-Gen генерирует промежуточный уровень (уровень сохраняемости), приложения используют в качестве входных параметров структуру базы данных. Входные данные включат в себя набор таблиц, представлений, хранимых процедур, которые описывают объекты предметной области, таким образом, они

могут быть использованы для создания бизнес логики приложения. Система является однонаправленной, т. е. генерация программного кода происходит только на основе структуры базы данных.

В качестве инструмента для кодогенерации используются шаблоны T4 [14]. 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, Web-страницу, файл ресурсов или исходный программный код на любом языке.

В случае с нашей системой используются текстовые шаблоны T4 времени разработки [14], позволяющие 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 база данных, которая содержит ключевые сведения о требованиях используемого приложения. В нашем случае в качестве модели используются метаданные, полученные из базы данных, которая поступает на вход нашей системы.

Процесс генерации кода

Процесс генерации начинается с проектирования базы данных. Выше мы подробно описали причины, по которым была выбрана именно концепция, в которой за основу берется спроектированная разработчиком база данных.

Когда подготовительные работы завершены и база данных спроектирована, мы можем, использовать систему кодогенерации C-Gen для генерации уровня сохраняемости. Процесс кодогенерации подразумевает создание CRUD хранимых процедур на сервере баз данных, а также объектов и методов в программном коде. Для достижения высокой производительности сервера баз данных, наряду с достаточным уровнем автоматизации решено поддерживать два вида хранимых процедур.

- **«Простые хранимые процедуры».** Структура таких процедур одинакова для каждой сущности базы данных. К таким хранимым процедурам относятся CRUD хранимые процедуры. Такие процедуры можно сгенерировать автоматически.

- **«Пользовательские хранимые процедуры».** Отображают нетривиальные выборки данных, специфические для конкретной ситуации, где важна производительность. Разработка таких хранимых процедур полностью контролируется программистом.

Таким образом, автоматизация написания CRUD хранимых процедур в сочетании, с одной стороны, с ручным написанием сложных запросов и контроль над базой данных в целом, с другой стороны, позволяет достигнуть высокой производительности и достаточного уровня автоматизации при работе с базой данных.

Следующим шагом после генерации хранимых процедур является создание непосредственно уровня сохраняемости, который является агентом между базой данных и приложением. Для каждого объекта в базе данных система C-Gen генерирует соответствующий класс. Для каждой хранимой процедуры генерируется метод, с помощь которого осуществляется взаимодействие. Параметры и тип возвращаемого значения сгенерированного метода зависит от сигнатуры хранимой процедуры. Также стоит обратить внимание на тот факт, что код, генерируемый C-Gen, инкапсулирует в себе все необходимые свойства и методы, для осуществления взаимодействия с базой данных.

После того, как система C-Gen завершит процесс генерации, разработчик получает API для работы с базой данных. Логика взаимодействия приложения с базой данных полностью скрыта за сгенерированными объектами. После внесения изменений в структуру базы данных, процесс генерации необходимо повторить, чтобы внести изменения в API.

Оценка эффективности

Чтобы оценить преимущества нашего подхода, мы сравнили производительность кода генерируемого системой кодогенирации C-Gen, с производительностью кода генерируемого системами NetTiers [15] и Entity Framework 4.0 соответственно. За основу была взята база данных небольшого интернет-магазина размером 18 mb. Затем мы измерили производительность на операциях выборки и вставить на различных объемах данных. Измерения проводились на сервере с процессором Core i5-2500K 3,3 ГГц, 8 Гб оперативной памяти, работающий под управлением Windows 7 x64 SP1 и Microsoft SQL Server 2008 R2 x64 Express. Результаты представлены на рисунке.

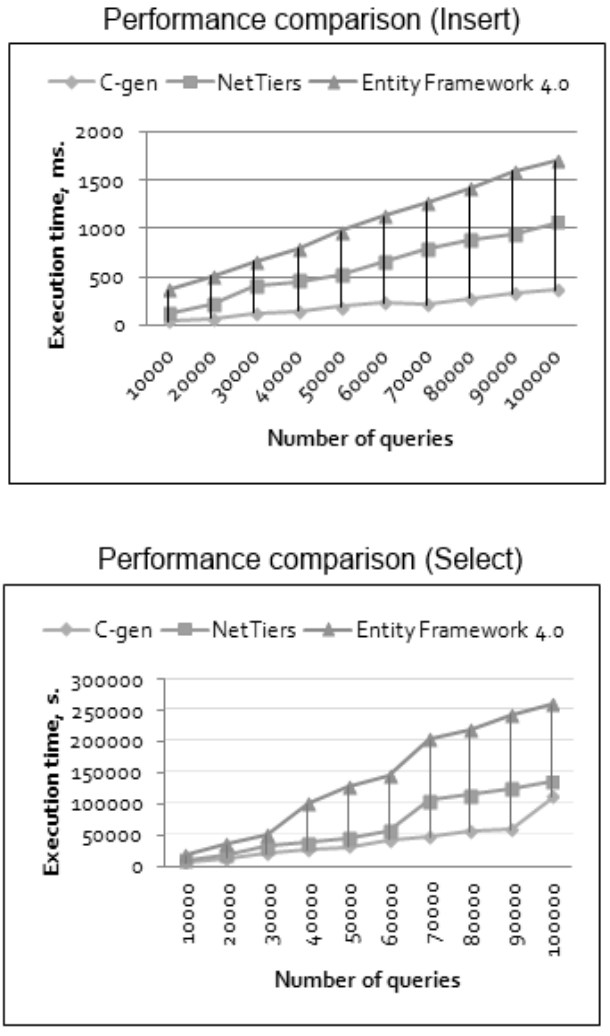

Рисунок. Сравнение производительности системы кодогенерации C-Gen с NetTiers и Entity Framework 4.0

На графике видно, что C-Gen генерирует более эффективный код. На операциях выборки это примерно в 3 раза быстрее по сравнению с NetTiers и примерно в 5 раз по сравнению с Entity Framework. На операциях вставки – 1,66 раза быстрее, по сравнению с NetTiers и около 3,4 раза по сравнению с Entity Framework.

Выводы

В данной работе определена актуальность использования систем обеспечивающих взаимодействие между реляционным и объектным представлением данных. Рассмотрен один из наиболее распространенных подходов реализации такого взаимодействия: использование ORM систем. Также проанализированы преимущества и недостатки использования систем подобного рода, на основе чего поставлены задачи, нахождения эффективного способа преобразования данных из объектного представления в реляционное и наоборот.

Исходя из этого, предложена методика реляционно-объектного представления, которая базируется на принципах построения информационных систем на основе модели в базе данных. Данная методика реляционно-объектного отображения решает две критические проблемы, характерные для систем подобного рода:

- производительность. Оптимальные запросы, написанные SQL разработчиком, оказываются на много более эффективными чем генерируемый ORM SQL код;
- автоматизация. Алгоритм генерации программного кода, основанный на методике «трех проекций» позволяет установить соответствие между объектным и реляционным представлением данных, а также предоставить разработчику API для доступа к данным.

Данная методика, положена в основу системы кодогенерации C-Gen [13]. Принципы построения, а также результаты сравнения производительности данной системы детально описаны в [12, 13, 16].

Одной из сфер применения данной методики может стать сфера поддержки и модернизации наследуемых (legacy) систем. Например, трудно найти корпорацию

с возрастом больше 25 лет, в которой не использовались бы информационные подсистемы, созданные на основе ранних аппаратно-программных платформ компании IBM. Базы данных таких подсистем содержат громадные объемы ценной информации и корпорация просто не может обойтись без их использования.

С другой стороны, данный подход может быть использован системами, для которых критично время выполнения приложения, либо есть ограничение в аппаратных ресурсах используемых системой. Например, неоптимальные SQL запросы, или дополнительная обработка данных в приложении может привести к увеличению потребляемых ресурсов, необходимых для обслуживания информационной системы. Для небольших систем этот факт может показаться незначительным, но когда дело доходит до промышленных масштабов, а принцип облачного хостинга строится на тарификации процессорного времени и расходуемой оперативной памяти, экономия может оказаться существенной.

1. *Keller W.* Object/Relational Access Layers — A Roadmap, Missing Links and More Patterns // Proceedings of the 3rd European Conference on Pattern Languages of Programming and Computing (EuroPLoP). – 1998, http://www.objectarchitects.de/ObjectArchitects/papers/Published/ZippedPapers/or06_proceedings.pdf.
2. *Иванников В.П., Гайсарян С.С., Антипин К.В., Рубанов В.В.* Объектно-ориентированное окружение, обеспечивающее доступ к реляционным СУБД // Труды Института системного программирования РАН. – 2001. – Том 2. – С. 89–114.
3. *Обзор* средств объектно-реляционной проекции (ORM) для платформы .NET http://arbinada.com/main/node/33
4. *Eggers J.* Implementing EXPRESS in SQL. Document ISO TC184/SC4/WG1/N292, October 1988.
5. *Mead M., Thomas D.* Proposed Mapping from EXPRESS to SQL. Technical Report, Rutherford Appleton Laboratory, May 1989.
6. *Morris K.C.* Translating EXPRESS to SQL: A User's Guide. Technical Report NISTIR 4341, National Institute of Standards and Technology, Gaithersburg, Maryland, May 1990.
7. *Sanderson D., Spooner D.* Mapping between EXPRESS and Traditional DBMS Models // Proceedings of EUG‘93 – The Third EXPRESS Users Group Conference, Berlin, October 2–3, 1993.
8. *Klein L., Stonis A., Jancauskas D.* EXPRESS/SQL white paper. Document ISO TC184/SC4/WG11/N144, February 2001.
9. *ORM* или объектно-реляционный проектор http://habrahabr.ru/company/piter/blog/165327/
10. *HQL* examples http://docs.jboss.org/hibernate/orm/3.3/reference/en/html/queryhql.html#queryhql-examples
11. *LINQ* to SQL [LINQ to SQL] http://msdn.-msdn.microsoft.com/ru/library-library /bb386976.aspx
12. *Лихацкий И.А.* Об одной методике формирования объектного представления реляционных данных // Проблеми програмування. – 2013. – № 3. – С. 79–85.
13. *Лихацкий И.А.* Средства кодогенерации для взаимодействия с базой данных через объекты // Проблеми програмування. – 2012. – № 2–3. – С. 384–385.
14. *Создание* кода и текстовые шаблоны T4 http://msdn.microsoft.com/ru-ru/library/bb126445.aspx.
15. *.netTiers* Application Framework – http://nettiers.com/
16. *Igor Lihatsky, Anatoliy Doroshenko, Kostiantyn Zhereb.* A Template-Based Method to Create Efficient and Customizable Object-Relational Transformation Components // Information Systems: Methods, Models, and Applications. 4th International United Information System Conference UNISCON 2012, Yalta, Ukraine, June 2012. Revised Selected Papers.

Получено 13.08.2013

Об авторе:

Лихацкий Игорь Александрович,
младший научный сотрудник Института программных систем НАН Украины.

***Место работы автора*:**

Институт программных систем
НАН Украины,
03680, Киев-187,
проспект Академика Глушкова, 40.
Тел.: +38(095)361 0330.
E-mail: igor_md@ukr.net