УДК 004.4, 004.65

А. В. Посконин, аспирант, e-mail: aposk@yandex.ru,
Московский государственный университет им. М. В. Ломоносова

Интеграция SQL-ориентированных СУБД и NoSQL-систем на уровне объектного отображения

Рассмотрены вопросы интеграции разнородных систем управления данными: традиционных SQL-ориентированных СУБД и новых NoSQL-систем. На примере реализованного прототипа программной библиотеки показаны возможности интеграции различных решений для управления данными на уровне объектного отображения, включая унифицированный язык запросов и отображение объектов языка программирования на различные модели данных. Приводятся результаты тестов производительности полученного прототипа.

Ключевые слова: *SQL, NoSQL, polyglot persistence, сервис-ориентированная архитектура, объектно-реляционное отображение, объектно-документное отображение, интеграция*

Введение

Современные приложения вынуждены работать с данными различной структуры, объема и степени важности, предъявляя к системам хранения данных достаточно жесткие (а порой и невыполнимые) требования. Отсутствие универсального решения для работы с данными привело к появлению на рынке большого числа узкоспециализированных систем, значительно отличающихся от традиционных SQL-ориентированных СУБД. Этот новый класс систем получил собирательное название "NoSQL" ("Not Only SQL"), подчеркивающее, что в ряде задач применение реляционных СУБД может быть не самым эффективным решением.

Стремительное развитие Интернет и Web-приложений вывело на первый план проблему масштабируемости при возрастающих нагрузках и объемах информации, которую NoSQL-системы стараются решить с использованием распределенных архитектур в противоположность классическим "односерверным" СУБД. Однако поддержка согласованности данных (и, особенно, поддержка ACID-транзакций) в условиях распределенности увеличивает не только сложность системы, но и время отклика из-за большого числа сетевых взаимодействий. По этой причине NoSQL-системы ослабляют гарантии согласованности данных и отказываются от ACID-транзакций в пользу масштабируемости, скорости и высокой доступности данных (см., например, [1]). Системы категории NoSQL, кроме того, основаны на нереляционных моделях данных (например, ключ-значение, документная модель и т. д.), ориентированы в основном на работу с неструктурированными или слабоструктурированными данными и используют отличные от SQL языки запросов и интерфейсы доступа к данным. Стоит также упомянуть о появлении нового поколения SQL-ориентированных СУБД, изначально создаваемых для работы в распределенной среде — так называемых NewSQL-системах. Эти СУБД поддерживают транзакционную семантику и язык SQL, но обеспечивают при этом приемлемую масштабируемость. Подробные обзоры существующих NoSQL- и NewSQL-решений можно найти, например, в работах [2—4].

Богатый выбор различных систем для управления данными обусловлен тенденцией к специализации инструментов: если раньше SQL-ориентированные системы рассматривались как универсальные СУБД, способные решать практически любые задачи, то сегодня для каждой задачи можно выбрать наиболее оптимальное и удобное решение (см., например, [5]). Например, если приложению приходится работать с сущностями, обладающими переменным набором атрибутов, то практичнее использовать NoSQL-системы с документной моделью данных, чем реализовывать модель Entity-Attribute-Value [6] средствами SQL. Документная модель данных позволяет хранить объекты с произвольным набором атрибутов, обычно представляемые в JSON и называемые "документами". При этом обычно допускаются списки и вложенные документы, поддерживаются индексы и выборки на основе полей документов. Сами документы обычно хранятся в коллекциях, не накладывающих каких-либо ограничений на набор атрибутов входящих в них документов (отсутствие строгой схемы данных). Одной из наиболее популярных систем такого типа является MongoDB. Эта система имеет

Рис. 1. Реляционная и документная модели данных

богатую функциональность и поддерживает атомарные операции на уровне одного документа, однако если приложение нуждается в полноценных ACID-транзакциях, то SQL-ориентированные системы по-прежнему вне конкуренции. На рис. 1 приведено соответствие между реляционной (SQL-ориентированной) и документной моделями данных.

Polyglot Persistence

Термин "Polyglot Persistence" [7] применяется в англоязычных источниках для обозначения практики использования нескольких различных систем хранения данных в рамках одного приложения. Потребность в этом обусловлена тем фактом, что современные приложения (особенно Web-приложения) вынуждены работать одновременно с несколькими видами данных. Например, это может быть информация о клиентах, каталог товаров, журналы событий и переходов по ссылкам, данные пользовательских сессий и т. д. Все эти данные не только обладают различной структурой и соотношением операций чтения/записи, но и предъявляют разные требования к надежности хранения и возможностям языка запросов. Таким образом, естественно применить для работы с этими данными несколько систем хранения (см., например, [8, 9]). Пример такого подхода приведен на рис. 2, показывающем применение различных систем управления данными для компонентов платформы электронной коммерции.

Однако применение такого подхода усложняет приложение, требуя использования различных языков запросов и интерфейсов для доступа к данным. Одним из решений этой проблемы является применение сервис-ориентированной архитектуры (Service-Oriented Architecture, SOA). При этом все детали доступа к данным эффективно скрываются за интерфейсом соответствующего сервиса, позволяя, к тому же, разбить приложение на более или менее независимые части, что упрощает управление разработкой. Этот подход, однако, требует значительных дополнительных усилий и накладных расходов на организацию взаимодействия между сервисами, что делает его применение нецелесообразным для небольших приложений.

Рис. 2. Polyglot Persistence

При разработке приложений, использующих системы хранения данных, практически всегда возникает проблема, известная как "impedance mismatch", а именно — необходимость каким-либо образом отображать объектную модель приложения на модель данных целевой системы хранения. В случае реляционных СУБД существует множество готовых решений и подходов для осуществления объектно-реляционного отображения (Object-Relational Mapping, ORM). Однако и для NoSQL-решений бывает необходимо осуществлять подобное отображение, например, объектно-документное отображение (Object-Document Mapping, ODM).

Применение готовых решений для объектного отображения, таких как ORM-библиотеки, упрощает и ускоряет разработку приложений, предоставляя высокий уровень абстракции и богатую функциональность. К сожалению, высокий уровень абстракции заметно снижает производительность и делает практически невозможными низкоуровневые оптимизации под конкретную используемую СУБД. По этой причине подобные решения редко применяются в высоконагруженных приложениях, где предпочтительнее использование более оптимизированных под конкретную задачу реализаций (см. [10]).

Интеграция на уровне объектного отображения

Слой объектного отображения представляется подходящим для интеграции разнородных систем хранения данных, однако нужно постараться избежать недостатков, присущих традиционным библиотекам объектного отображения. Для этого требуется пересмотр классической архитектуры ORM-библиотек, а именно:

- использование модульного подхода вместо монолитной системы: слой объектного отображения может быть собран из отдельных "строительных блоков", чтобы лучше отвечать потребностям разработчика. Кроме того, если стандартные реализации компонентов по каким-то причинам не подходят, они должны быть легко заменяемыми;
- максимально "чистая" объектная модель, свободная от логики взаимодействия с подсистемами хранения. Сущность, в отличие от случая ORM, может содержать не только поля простых типов,

но также списки, вложенные объекты и динамические свойства;
- настраиваемый уровень абстракции: поддержка интерфейсов разных уровней позволяет найти нужный компромисс между уровнем абстракции и производительностью;
- независимое отображение сущностей и поддержка ассоциаций поверх слоя отображения позволяет иметь целостную объектную модель, несмотря на то, что сущности могут отображаться на разные системы хранения;
- наличие унифицированного языка запросов с поддержкой базовой функциональности, не привязанного к SQL или другому языку запросов;
- возможность усилить гарантии целостности данных с помощью валидации на стороне приложения;
- поддержка кэширования, "ленивой" загрузки, Unit of Work и других успешно себя зарекомендовавших практик (см., например, [11]).

Приведенные принципы были применены при разработке прототипа программного каркаса MapperStack для интеграции разнородных систем хранения данных на уровне объектного отображения в Web-приложениях на PHP. В архитектуре MapperStack можно выделить два основных слоя: слой отображения, состоящий из объектов, реализующих интерфейс Mapper (например, методы getEntityId(), find(), fmdBy() и т. д.), и верхний слой MapperStack, координирующий работу слоя отображения и предоставляющий возможности интеграции, в том числе поддержку ассоциаций (рис. 3).

Каждый объект в слое отображения ответственен за отображения сущностей определенного класса. Каждая сущность должна иметь уникальный идентификатор (аналог первичного ключа в реляционной модели данных) и может содержать динамические свойства, списки и вложенные объекты, а также метаданные, определяющие правила отображения, например:

```
class BlogPost extends MapperStack\Object
{
    protected $id;
    protected $username;
    protected $text;
    protected $tags;
    protected $comments;
    public static function meta(EntityMetaClass $metaClass)
    {
    $metaClass->db('blog')
                ->collection('posts')
                ->field('id', 'integer', '_id')
                ->field('username', 'string')
                ->field('text', 'string')
                ->field('tags', 'string[]')
                ->object('comments', 'Blog\Comment[]')
                ->id('id');
    }
}
```

В настоящее время реализовано отображение сущностей на MySQL и MongoDB, а также унифицированный язык запросов с трансляцией в SQL и запросы к MongoDB. Поддерживаются операции filter, sort, skip, limit, project и другие, условия выборки задаются в виде логического выражения, поддерживаются предикаты (как механизм расширения языка запросов), связываемые и встроенные параметры. Кроме того, если целевая система хранения данных поддерживает документную модель данных, то в запросах можно использовать операции сопоставления элементов коллекции и переход по графу объектов, чтобы добраться до атрибутов вложенных объектов.

Рассмотрим пример запроса в терминах сущностей и результирующие запросы к целевым системам хранения данных:

```
$p := $m->query ( ' Domain\Product' )
        ->filter('price >= 1000 & & type == "shoes"')
        ->sort('price', 'asc')
        ->limit(3)
        ->getResult() ;
```

Этот запрос допускает трансляцию как в SQL, так и в язык запросов MongoDB (сопоставление по образцу):

```
SELECT*FROM "products'     db.products.find( {
    WHERE 'price' > 1000          $and : [
        AND 'type' = 'shoes'          { 'price' : { $gte : 1000} },
    ORDER BY 'price'              j 'type' : 'shoes' }
    LIMIT 3                   ] } ).sort( { 'price' : 1 } ).limit(3)
```

Рассмотрим теперь запрос, содержащий операцию сопоставления элементов массива (в данном случае в массиве comments ищется элемент, для которого выполняется условие в фигурных скобках, кроме того, присутствует предикат регулярного выражения). Такой запрос не может быть выражен средствами SQL, но поддерживается в MongoDB:

```
$posts = $m->query('Blog\BlogPost')
        ->filter('regexp(text, "/programming/i" & &
                comments {username == "ivan"} || rating > 100')
        ->getResult ();
```

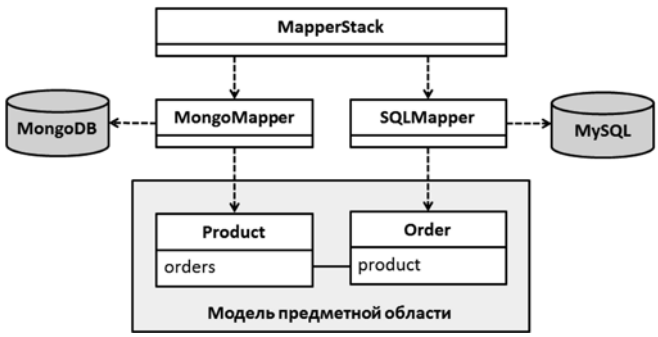

Рис. 3. Отображение сущностей с помощью MapperStack

Результирующий запрос к MongoDB будет выглядеть следующим образом (можно отметить, что запрос в MapperStack является более наглядным и привычным):

```
db.posts.find( { $or : [
                { $and : [
                    { 'text' : /programming/i },
                    { 'comments' :
                        { $elemMatch: { 'username' : 'ivan' } }
                    }
        ] },
        { 'rating' : { $gt : 100 } }
    ] } ) ;
```

Для непосредственной организации отображения в MapperStack реализован ряд компонентов, таких как классы для хранения метаданных, адаптеры для систем кэширования, абстракция отображения типа, механизмы для отслеживания изменений и преобразования данных, Unit of Work (откладывание всех операций модификации данных до вызова метода flush(), синхронизующего состояние с системой хранения и выполняющего операции с учетом ассоциаций [12]) и ряд других компонентов. Каждый компонент может быть легко заменен другой реализацией, кроме того, возможны полностью пользовательские реализации классов отображения. Дальнейшее развитие данного подхода может включать поддержку других типов систем, реализацию оптимистических блокировок, абстракцию MapReduce, поддержку модификации без чтения, агрегатных функций, наследования и т. д.

Производительность библиотеки

В заключение рассмотрим некоторые аспекты производительности библиотеки MapperStack на примере модуля объектно-реляционного отображения (SQLMapper). Для этого был разработан тестовый сценарий, включающий выборку сущностей, модификацию полей, удаление сущностей и фиксирование сделанных изменений в базе данных. В качестве эталона приведены характеристики для того же сценария, реализованного с помощью библиотеки Doctrine ORM — мощной и хорошо оптимизированной библиотеки объектно-реляционного отображения для PHP (см. таблицу).

Результаты исполнения тестового сценария

Библиотека	Время работы, мс	Пиковое потребление памяти, Кбайт
MapperStack SQLMapper (первый запуск)	1625	10190
MapperStack SQLMapper (дальнейшие запуски)	595	6940
Doctrine ORM (первый запуск)	1540	9472
Doctrine ORM (дальнейшие запуски)	560	6144

При первом запуске сценария происходит анализ и трансляция запросов с последующим кэшированием, чем и обусловлено несколько большее время работы. Кэш результатов запросов при тестировании не использовался. В плане производительности SQLMapper незначительно уступает специализированным ORM-библиотекам (главным образом вследствие большей модульности и слабой связанности компонентов программного каркаса). Стоит заметить, что с использованием компонентов MapperStack можно реализовать и более производительное отображение (например, работая напрямую с драйвером СУБД, пропуская этап преобразования типов и т. д.).

Заключение

Реализация модульного программного каркаса для интеграции различных систем хранения на уровне объектного отображения позволяет быстро разработать прототип приложения, используя стандартную функциональность, а затем при необходимости постепенно заменять их на более эффективные реализации исходя из решаемых задач. Интеграция различных систем хранения открывает новые возможности, позволяя использовать наиболее подходящие решения для работы с данными, сохраняя при этом целостную объектную модель приложения. Это актуально, прежде всего, для Web-приложений и мобильных сервисов, хранящих и обрабатывающих большие объемы разнородных данных.

Список литературы

1. **Merriman D.** On Distributed Consistency, 2010. URL: http://blog.mongodb.org/post/475279604/on-distributed-consistencv-part-1
2. **Cattell R.** Scalable SOL and NoSQL Data Stores, 2011. URL: http://www.cattell.net/datastores/Datastores.pdf
3. **Aslett M.** NoSQL, NewSQL and Beyond: The drivers and use cases for database alternatives // Отчет компании 451 Group, 2011. URL: https://451research.com/report-long?icid=1651
4. **Кузнецов С. Д., Посконин А. В.** Распределенные горизонтально масштабируемые решения для управления данными // Труды Института системного программирования РАН. 2013. Т. 24. С. 327—358.
5. **Stonebraker M., Cetintemel U.** "One Size Fits All": An Idea Whose Time Has Come and Gone // ICDE '05: Proceedings of the 21st International Conference on Data Engineering, Вашингтон, 2005.
6. **Entity-Attribute-Value** model. URL: http://en.wikipedia.org/wiki/Entity %E2 %80 %93attribute %E2 %80 %93value model
7. **Fowler M.** Polyglot Persistence, 2011. URL: http://www.martinfowler.com/bliki/PolyglotPersistence.html
8. **Кузнецов С. Д., Посконин А. В.** Возможно ли сотрудничество SQL и NoSQL? // Открытые системы. СУБД. 2013. N. 9. С. 38—41.
9. **Francia S., Hileman J.** Augmenting RDBMS with MongoDB for eCommerce. URL: http://www.nosqldatabases.com/main/2011/4/11/augmenting-rdbms-with-mongodb-for-ecommerce.html
10. **Посконин А. В.** Web-приложения и данные проблемы абстракции и масштабируемости // Труды Института системного программирования РАН. 2012. Т. 23. С. 159—171.
11. **Miller J.** Design Patterns for Data Persistence, 2009. URL: http://msdn.microsoft.com/en-us/magazine/dd569757.aspxy
12. **Fowler M.** Patterns of Enterprise Application Architecture. Бостон: Addison Wesley, 2002.

A. V. Poskonin, Postgraduate, e-mail: aposk@yandex.ru, Lomonosov Moscow State University

Integrating SQL-Based DBMSs with NoSQL Datastores at the Object-Mapping Layer

This paper will discuss and evaluate an approach to building an object-mapping layer that provides integration of multiple different data management systems, including SQL-based DBMSs and popular NoSQL systems. A prototype object-mapping framework was developed to illustrate basic concepts of such integration including unified entity-based query language, transparent object mapping and a high-level interface, that still allows low-level optimizations and delivers sufficient performance for most use cases. Unified query language makes it possible to translate a single query to different underlying datastores without a need to modify the query while object-mapping layer manages object construction and lifecycle, providing simple interface for persisting, modifying and deletion of objects. In conclusion, performance of the implemented framework is measured and compared with popular ORM solution.

Keywords: *SQL, NoSQL, polyglot persistence, service-oriented architecture, object-relational mapping, object-document mapping, integration*

References

1. **Merriman D.** *On Distributed Consistency.* 2010. URL: http://blog.mongodb.org/post/475279604/on-distributed-consistencv-part-1
2. **Cattell R.** *Scalable SQL and NoSQL Data Stores.* 2011. URL: http://www.cattell.net/datastores/Datastores.pdf
3. **Aslett M.** *NoSQL, NewSQL and Beyond:* The drivers and use cases for database alternatives, 451 Group, 2011. URL: https://451research.com/report-long?icid=1651
4. **Kuznetsov S. D., Poskonin A. V.** Raspredelennye gorizontal'no masshtabiruemye reshenija dlja upravlenija dannymi. *Trudy Instituta sistemnogo programmirovanija* RAN. 2013. V. 24 P. 327—358.
5. **Stonebraker M., Cetintemel U.** "One Size Fits All": An Idea Whose Time Has Come and Gone, ICDE '05. *Proceedings of the 21st International Conference on Data Enegineering, Washington*, 2005.
6. **Entity-Attribute-Value** model. URL: http://en,wikipedia.org/wiki/Entitv %E2 %80 %93attribute %E2 %80 %93value_model
7. **Fowler M.** *Polyglot Persistence*, 2011. URL: http://www.martin-fpwler.com/bliki/PolyglotPersistence.html
8. **Kuznetsov S. D., Poskonin A. V.** Vozmozhno li sotrudnichestvo SQL i NoSOL? *Otkrytye sistemy. SUBD.* 2013. N. 9. P. 38—41.
9. **Francia S., Hileman J.** *Augmenting RDBMS with MongoDB for eCommerce.* URL: http://www.nosqldatabases.com/main/2011/4/11/augmenting-rdbms-with-mongodb-for-ecommerce.html
10. **Poskonin A. V.** Web-prilozhenija i dannye: problemy abstrakcii i sshtabiruemosti. *Trudy Instituta sistemnogo programmirovanija RAN.* 2012. V. 23. P. 159—171.
11. **Miller J.** *Design Patterns for Data Persistence.* 2009. URL: http://msdn.microsoft.com/en-us/magazine/dd569757.aspx
12. **Fowler M.** *Patterns of Enterprise Application Architecture.* Boston: Addison Wesley, 2002.

УДК 681.518: 339.13

Э. М. Димов, д-р техн. наук, проф.,
О. Н. Маслов, д-р техн. наук, проф., зав. каф., e-mail: maslov@psati.ru,
Ю. В. Трушин, канд. техн. наук, доц.,
Поволжский государственный университет телекоммуникаций и информатики, г. Самара

Выбор средств программного обеспечения процесса статистического имитационного моделирования

Рассматривается проблема выбора средств программного обеспечения процесса статистического имитационного моделирования (СИМ) сложных систем организационно-технического типа. Представлены примеры реализации СИМ-моделей с применением универсального программного языка Delphi и среды имитационного моделирования AnyLogic.

Ключевые слова: *метод статистического имитационного моделирования, средства программного обеспечения, универсальный язык Delphi, среда имитационного моделирования AnyLogic*

Введение

Разработка и реализация систем управления (СУ) иерархическими и многокритериальными сложными системами (СС) организационно-технического типа (социально-экономическими, экологическими, военными и т. п.), элементами которых являются лица, принимающие решения (ЛПР), до настоящего времени является актуальной проблемой, которая имеет важное практическое значение. Решению данной проблемы способствует создание