Web-приложения и данные: проблемы абстракции и масштабируемости

Андрей Посконин, <aposk@yandex.ru>

Аннотация. Сегодня всё чаще звучит утверждение, что универсальных хранилищ данных не существует. Несмотря на то, что SQL-ориентированные системы управления базами данных (СУБД) широко применялись и применяются сегодня, масштабирование приложений при использовании этих систем сильно затруднено. В связи с этим появилось много различных хранилищ данных, стремящихся соответствовать требованиям современных высоконагруженных Web-приложений. В настоящее время наблюдается тенденция к использованию нескольких технологий и систем в рамках одного приложения для решения различных задач. В данной статье рассматриваются основные подходы к реализации доступа к данным и проблемы абстракции от используемых хранилищ, а также предлагаются некоторые принципы организации слоя работы с данными при использовании нескольких хранилищ данных разного типа.

Ключевые слова. Web-приложения, масштабируемость, SQL, NoSQL, абстракция от деталей хранения данных.

1. Введение

В последнее время Web-приложения приобрели огромную популярность. С развитием Интернета всё острее встаёт проблема разработки приложений, которые могут легко масштабироваться, то есть адаптироваться к постоянно возрастающим нагрузкам. Важнейшую роль в возможностях такой адаптации играют используемые технологии хранения данных, так как на уровне приложения масштабирование является относительно легкой задачей и сводится к добавлению новых серверов, обрабатывающих запросы.

Наиболее часто используемой для Web-приложений архитектурой является Модель – Вид – Контроллер (Model – View – Controller, MVC). MVC – это архитектурный шаблон, который разбивает приложение на три части: Модель (объектная модель, бизнес-логика и доступ к данным), Вид (отображение данных, генерация страниц) и Контроллер (маршрутизация запросов) [1]. Вид и Контроллер обычно поддерживаются различными фреймворками (программными каркасами) и библиотеками (например, Zend Framework [2]), тогда как реализация Модели целиком ложится на разработчика. В данной статье будут рассматриваться вопросы, связанные с Моделью, которая обычно

состоит из объектной модели предметной области (ключевые абстракции и связи между ними), слоя доступа к данным, структур для проверки прав доступа, форм, правил проверки корректности данных и других компонентов, реализующих бизнес-логику приложения. Также Модель часто включает в себя слой сервисов [1]. Этот слой состоит из классов, которые манипулируют объектами предметной области и координируют выполнение сложных задач.

Объектная модель приложения практически всегда должна долговременно сохраняться, например, с использованием SQL-ориентированной СУБД. Так как объектная модель приложения и модель данных хранилища имеют различные структуры и оперируют разными понятиями, то необходим промежуточный слой доступа к данным, осуществляющий отображение между объектами языка программирования и базой данных и предоставляющий интерфейс вышестоящим уровням. В данной статье рассматриваются различные аспекты реализации доступа к данным: выбор подходящего хранилища, модели и уровня согласованности данных, взаимосвязь абстракции и производительности, а также совместное использование различных СУБД для решения задач масштабирования.

2. Web-приложения и SQL

SQL-ориентированные СУБД господствуют на рынке уже более 30 лет. Эти системы до сих пор широко используются для хранения данных в различных Web-приложениях, для них было разработано много методов, помогающих частично преодолеть проблемы масштабирования при увеличивающихся нагрузках. К сожалению, особенности архитектуры этих систем не позволяют им хорошо масштабироваться горизонтально, однако мощный язык запросов, богатые возможности, поддержка ACID-транзакций и большое число разработанных инструментов делают их популярным инструментом для решения многих задач.

2.1. SQL и ООП

Как уже было сказано, объектная модель приложения и модель данных SQL оперируют разными понятиями, поэтому напрямую их связать не получается, и зачастую приходится прибегать к объектно-реляционному отображению (Object-Relational Mapping, ORM). Возможны, например, следующие варианты:

- Отказ от полноценной объектной модели, работа с БД напрямую.
- Использование слоя абстракции от конкретной реализации языка SQL.
- Использование простых оберток для строк и столбцов таблиц (образцы Row Data Gateway и Table Data Gateway, реализованные, например, в [2]).
- Ручная реализация отображения объектов (требует написания большого количества кода, содержащего SQL-запросы для работы с данными).

- Использование ORM-библиотек (например, [3], [4]).

Здесь уровень абстракции каждого следующего решения выше, чем предыдущего. Это значит, что объектная модель приложения может становиться (почти) независимой от деталей хранения, что упрощает разработку и сопровождение кода. Кроме того, уровень абстракции напрямую связан с производительностью: чем выше уровень абстракции, тем больше промежуточных слоев и меньше возможностей для низкоуровневых оптимизаций. Чем большая нагрузка планируется на приложение, тем ниже должен быть уровень абстракции, чтобы обеспечить требуемую производительность. Например, ORM-библиотеки предоставляют богатую функциональность и высокий уровень абстракции, но использовать такую библиотеку в высоконагруженном приложении не представляется возможным из-за слишком медленной работы и невозможности использовать низкоуровневые оптимизации, зависящие от конкретной СУБД [5].

2.2. Проблемы реализации ORM

Рассмотрим ключевые вопросы реализации объектно-реляционного отображения. По распределению обязанностей можно выделить два основных образца реализации ORM:

- Active Record – обязанности по доступу к БД возлагаются на сами классы-сущности: в этом случае они будут иметь, помимо своих методов, такие методы, как Save(), Update() или Delete().
- Data Mapper – обязанности по доступу к БД возлагаются на отдельные объекты, что позволяет отделить объектную модель приложения от логики взаимодействия с хранилищем. Другим полезным образцом является Repository – скрытие всех деталей работы с хранилищем от объектной модели.

При работе с базой данных крайне важно минимизировать объем выбираемых данных, тем более не выбирать лишних данных, поэтому в ORM часто применяется так называемая «ленивая» загрузка (Lazy Loading), суть которой сводится к тому, что объекты и коллекции загружаются из базы данных только тогда, когда к ним производится обращение (при помощи классов-заместителей). Несмотря на то, что такая стратегия кажется разумной, она несет в себе много опасностей, таких как непреднамеренная загрузка всей коллекции, «путешествия» по графу объектов, вызывающие множество отдельных запросов и т.д. В этом плане даже при использовании библиотек для ORM необходимо помнить о специфике доступа к БД.

На слой поддержки ORM также возлагаются обязанности по недопущению дубликатов объектов с одинаковым первичным ключом, для чего обычно применяется образец Identity Map. Оптимизировать выполнение операций изменения данных помогает образец Unit of Work путем накопления операций изменения и выполнения затем всех операций сразу в одной транзакции.

Полезным также может являться механизм событий, срабатывающих при записи или чтении из БД, что помогает выполнять сложные проверки корректности данных и прав доступа на уровне приложения (подобный механизм событий реализован, например, в [4]). Более подробно об образцах, применяемых при реализации объектно-реляционного отображения, можно узнать, например, из [6].

Что касается самого отображения объектов на таблицы, то в ORM-библиотеках оно осуществляется с помощью метаданных (в виде XML, аннотаций и т.п.), которые определяют атрибуты и их типы, а также связи между объектами (таблицами). На основе метаданных ORM-библиотека может генерировать схему целевой базы данных. Существенным недостатком метаданных является усложнение и замедление работы приложения, однако кэширование позволяет бороться с этой проблемой.

Абстракция от конкретного хранилища данных отчасти достигается с помощью «объектной» модификации языка SQL и конструкторов запросов, позволяющих обойтись без смешения кода приложения и запросов. По «объектному» запросу строится запрос к целевой базе данных, который также может кэшироваться, чтобы избежать трансляции при каждом вызове.

Таковы основные механизмы, позволяющие до какой-то степени отделить объектную модель приложения от деталей работы с СУБД. К сожалению, применение «тяжелых» библиотек ORM очень негативно сказывается на производительности приложений и затрудняет оптимизацию.

2.3. Оптимизация работы приложений с SQL СУБД

Наиболее важным вопросом при проектировании Web-приложения является выбор уровня абстракции при доступе к БД. Высоконагруженные приложения обычно используют написанные вручную классы для сохранения и загрузки объектов из БД, позволяющие применять низкоуровневые оптимизации для каждой конкретной СУБД [5]. Далее приводятся некоторые общие способы, позволяющие повысить производительность работы с БД:

- Оптимизация SQL-запросов
- Анализ использования индексов
- Избежание «долгих» транзакций и работы в режиме auto-commit
- Выбор уровня изолированности транзакций
- Партицирование таблиц (разбиение большой таблицы на логические части по строкам или по столбцам)
- Денормализация таблиц
- Кэширование объектов и результатов «долгих» запросов

2.4. Масштабирование на уровне SQL СУБД

Если оптимизация работы с БД и покупка более мощного оборудования (вертикальное масштабирование) не могут решить проблем с возрастающей нагрузкой, то существуют некоторые возможности горизонтального масштабирования, хотя и довольно ограниченные [7], [8]. Рассмотрим сначала репликацию данных (поддержку нескольких серверов с одними и теми же данными). Выделяют два вида репликации: синхронную (данные всегда совпадают) и асинхронную (данные на копиях могут в течение какого-либо промежутка времени различаться). Асинхронная репликация работает быстрее синхронной, однако при её использовании возникает проблема появления устаревших данных. Рассмотрим две схемы репликации, позволяющие масштабировать нагрузку:

- Master – Slaver: один сервер является ведущим, все запросы на запись идут к нему, а он пересылает измененные данные на второстепенный сервер, с которого может осуществляться только чтение. Если используется асинхронная репликация и устаревшие данные недопустимы, то можно осуществлять чтения с ведущего сервера. Схема Master-Slave позволяет масштабировать чтения.
- Master – Master: оба сервера доступны как для чтения, так и для записи. Эта схема сложнее в реализации, но позволяет масштабировать операции записи.

Кроме масштабирования, репликация применяется и для повышения надежности, а также для других целей (например, можно держать один сервер в запасе и выполнять на нем «долгие» аналитические запросы).

Кроме репликации существует еще одна схема масштабирования, называемая шардингом (разделением данных). Горизонтальный шардинг подразумевает перемещение части строк большой таблицы на новый сервер, что позволит распараллелить обращения к ним. Вертикальный шардинг подразумевает перемещение каких-либо таблиц (столбцов таблиц) на отдельный сервер с той же целью. Обычно шардинг и репликация применяются в комплексе, чтобы обеспечить не только распараллеливание обработки запросов, но и надежность.

К сожалению, горизонтальное масштабирование традиционных SQL-ориентированных систем довольно затруднительно, непрозрачно для приложений, затрудняет поддержку целостности данных и выполнение запросов, затрагивающих несколько разделенных частей таблицы.

Чтобы преодолеть трудности масштабирования, были предложены и разработаны распределенные SQL-ориентированные СУБД, которые масштабируются легче и поддерживают основные возможности SQL. Такие системы, однако, не лишены недостатков: проектирование схемы становится

сложнее, при написании запросов надо учитывать расположение данных, а ряд возможностей SQL недоступен.

2.5. Преимущества и проблемы SQL

К несомненным преимуществам модели данных и языка SQL можно отнести богатые возможности, поддержку интуитивно понятной семантики ACID-транзакций, четко определенную схему данных, возможность задавать ограничения целостности на уровне БД, большое количество инструментов, качество и зрелость программных продуктов.

Однако, как было показано ранее, SQL-ориентированные СУБД испытывают серьезные проблемы, связанные с масштабированием. Но это не единственная проблема: также SQL не очень хорошо сочетается с ООП, требуя достаточно сложного объектно-реляционного отображения. Рассмотрим, например, реализацию динамического набора атрибутов. При использовании SQL СУБД есть следующие варианты:

- Таблица с большим числом столбцов.
- Базовая таблица и таблицы с атрибутами для каждого подтипа.
- Entity-Attribute-Value, то есть поддержка трех таблиц – сущностей, атрибутов и значений, а потом их дорогостоящее и сложное соединение.

Ни один из этих вариантов не подойдет при большом количестве разнообразных объектов и атрибутов [9]. Возможно, для решения этой задачи лучше подойдет СУБД с моделью данных, отличной от SQL.

3. Web-приложения и NoSQL

Для преодоления ограничений SQL-ориентированных систем появилось множество новых распределенных систем, объединенных в класс так называемых NoSQL-систем (Not Only SQL). Эти системы отказываются от модели данных SQL, а также от поддержки строгой согласованности данных и ACID-транзакций. Взамен они предоставляют высокую надежность, производительность (благодаря хранению данных в основной памяти) и хорошую горизонтальную масштабируемость (благодаря репликации и шардингу). Модели данных, на которых строятся NoSQL-системы, могут быть как простыми (хранилища типа ключ-значение), так и более сложными (документные хранилища, хранилища расширяемых записей). Кроме того, существуют NoSQL-хранилища на основе облачных решений, что позволяет оптимизировать также и финансовые затраты [10].

3.1. Согласованность данных

В соответствии с утверждением, известным как теорема CAP [11], [12], распределенная система не может гарантировать одновременно:

- Согласованность данных (Consistency) - все копии объектов в любой момент времени находятся в согласованном состоянии. Таким образом, здесь понятие согласованности отличается от согласованности из свойств ACID-транзакций.
- Доступность (Availability) - система всегда отвечает на запросы.
- Устойчивость к разделению сети (Partition Tolerance) - система, даже будучи разделенной на части, продолжает обслуживать пользователей.

Каждая распределенная система реализует некий компромисс между этими свойствами. Другой, возможно, более важной, причиной отказа от строгой согласованности данных и распределенных транзакций является стремление уменьшить задержки и увеличить производительность [13]. Таким образом, в NoSQL-системах обычно поддерживаются более слабые модели согласованности – согласованность в конечном счете (Eventual Consistency) и её вариации [14]. Многие Web-приложения, однако, не требуют строгой согласованности и могут работать со слегка устаревшими данными. На практике значительная часть NoSQL-систем возвращает последние версии данных, допуская устаревшие данные только в случае сбоев или высокой нагрузки [15]. Некоторые из них поддерживают дополнительные гарантии согласованности, атомарные операции и другие возможности, упрощающие реализацию приложения.

3.2. NoSQL и ООП

Модели данных NoSQL-систем являются более простыми и гибкими, чем SQL. Например, динамический набор атрибутов может быть легко реализован при использовании документного хранилища, например, MongoDB [9]. Документ – это произвольный набор атрибутов, который может включать вложенные документы и коллекции документов, при этом поддерживается индексация и поиск по атрибутам. Такая модель данных лучше сочетается с ООП, чем SQL, а объектно-документное отображение (Object-Document Mapping, ODM) осуществляется существенно проще, чем ORM. Для ODM также уже существуют библиотеки, например Doctrine ODM [4], MongoMapper [16].

Более простой моделью данных является модель «ключ-значение». Хранилища этого типа поддерживают только доступ по уникальному ключу, но могут с успехом применяться для хранения сериализованных объектов, если не требуется поиск по дополнительным атрибутам. Такие хранилища просты в использовании, обеспечивают очень высокую производительность и могут быть использованы и как отдельное хранилище, и для кэширования каких-либо данных. Подробный обзор и сравнение различных хранилищ данных можно найти в [17].

3.3. Преимущества и проблемы NoSQL

Основными преимуществами систем класса NoSQL является их высокая производительность, хорошая горизонтальная масштабируемость и надежность. К сожалению, отсутствие строгой согласованности данных и поддержки ACID-транзакций ограничивают область применения этих систем.

В качестве примера рассмотрим реализованный в [18] прототип Web-форума, использующий NoSQL решения:

- MongoDB [19] (в конфигурации Replica-Set) в качестве основного хранилища данных (данные о пользователях, разделы, темы, сообщения). Для осуществления объектно-документного отображения использовалась библиотека Doctrine ODM.
- Membase [20] – распределенное хранилище типа «ключ-значение» для хранения данных пользовательских сессий и кэширования.

В качестве языка программирования использовался PHP5, приложение построено с применением Zend Framework [2]. Благодаря использованию NoSQL-решений может быть достигнута хорошая масштабируемость. Данное приложение не требует строгой согласованности данных и ACID-транзакций, поэтому может быть успешно реализовано и без них. Однако есть приложения, для которых наличие транзакционной семантики и контроль целостности данных являются критически важными (например, приложения в банковской сфере или в сфере электронной коммерции). Для таких приложений и задач лучше подходят SQL-ориентированные СУБД.

Проблема отсутствия универсальной СУБД приводит к новой тенденции – использованию нескольких различных систем в одном приложении для решения подходящих задач (“polyglot persistence”) [21], [22].

4. Совместное использование SQL и NoSQL-систем

Существует несколько вариантов совместного применения SQL и NoSQL-систем:

- Использование NoSQL-хранилища в качестве кэша при использовании SQL-ориентированной СУБД. В этом случае операции чтения, не требующие строгой согласованности данных, могут читать данные из кэша, чтобы снять нагрузку с основной СУБД. В зависимости от используемого NoSQL-решения, такой кэш может поддерживать даже достаточно сложные запросы и при этом обеспечивать высокую производительность.
- Использование NoSQL-хранилища через SQL-интерфейс. Этот подход позволяет переносить приложения, написанные под SQL-ориентированные СУБД, на NoSQL-системы с минимальными изменениями. Такой подход, однако, имеет некоторые ограничения, но зато не требует от разработчиков

изучения новых языков запросов и API. Например, GenieDB поддерживает и NoSQL-, и SQL-интерфейсы для доступа к данным [23].

- Репликация данных между SQL и NoSQL системами. Этот подход можно реализовать, например, с помощью Tungsten Replicator [24]. Таким образом, чтение и запись могут осуществляться из разных хранилищ (на уровне приложения возможно применение образца CQRS для разделения чтения и записи [25]).

- «Polyglot Persistence», то есть использование нескольких СУБД для решения различных задач внутри одного приложения [21], [22]. Этот подход будет подробнее рассматриваться далее.

4.1. «Polyglot Persistence»

Как было показано ранее, универсальной СУБД не существует, каждая конкретная система хорошо решает лишь определенный круг задач. Например, в [9] рассматривается совместное применение MySQL и MongoDB для реализации платформы электронной коммерции. Тем не менее, использование в одном Web-приложении нескольких разных хранилищ данных (даже при использовании библиотек объектного отображения) связано с определенными сложностями:

- Несколько различных языков запросов и API
- Сложно поддерживать согласованность данных между хранилищами
- Смена хранилища требует переписывания части приложения
- Проблема отображения объектов на разные хранилища

Существуют возможности частичного преодоления этих проблем (например, использование LINQ [26]), однако полностью решить их не удается. В данной ситуации может помочь применение слоя абстракции от деталей хранения, чтобы сделать объектную модель приложения максимально независимой от них. Кроме того, большинству проектов в начальной фазе развития трудно определиться с используемыми СУБД, и впоследствии приходится переписывать большую часть приложения. Поддержание определенного уровня абстракции может помочь более легкому переходу на другие системы хранения данных. В настоящее время появляются библиотеки отображения, поддерживающие (в ограниченной степени) отображение на разные источники данных (например, DataMapper [27]).

4.2. Абстракция от деталей хранения

Преодолеть проблемы отображения объектов на различные хранилища данных и сделать объектную модель приложения максимально независимой от деталей хранения можно с помощью применения дополнительного слоя абстракции. Основными его функциями могут быть следующие:

- Отображение объектов приложения на различные модели данных (SQL/NoSQL).
- Обеспечение независимости логики приложения от деталей реализации хранения данных.
- Поддержка абстрактного языка запросов, позволяющего выбирать объекты (примером такого языка может служить LINQ).
- Поддержка репликации и шардинга между различными хранилищами.
- Поддержка кэширования данных, по возможности, прозрачно для приложения.
- Обеспечение гибкости и расширяемости.

Несмотря на все преимущества подобного слоя абстракции, при реализации возникает много проблем.

4.3. Проблемы реализации и возможные решения

Основной проблемой здесь является обеспечение высокого уровня абстракции при сохранении приемлемой производительности и гибкости. Рассмотрим подробнее некоторые проблемы и возможные решения:

- Обеспечение независимости объектной модели приложения от деталей хранения может быть достигнуто с помощью применения образца Data Mapper [1].
- Поддержка абстрактного языка запросов может быть реализована с помощью конструкторов запросов. В этом случае отпадает необходимость в лексическом, синтаксическом разборе кода запроса, упрощается трансляция в язык запросов конкретного хранилища.
- Для описания правил отображения объектов требуются метаданные. С помощью метаданных (например, в виде XML-документов) может быть описана вся объектная модель приложения и правила её хранения (связи, репликация, хранилища для чтения и записи и т.д.) На основе метаданных также может осуществляться генерация кода. Для обеспечения производительности метаданные должны кэшироваться в основной памяти.
- Моделирование связей между объектами, находящимися в разных хранилищах, является еще одной сложной проблемой. Эти связи могут задаваться на уровне метаданных и реализовываться с помощью механизма «ленивой» загрузки. Возможен и вариант отказа от моделирования связей, что, однако, портит объектную модель и требует явных запросов. Компромиссом между этими подходами является автоматическая поддержка связей для объектов, находящихся в одном хранилище.

- Для обеспечения гибкости и расширяемости возможно поддержание нескольких уровней абстракции и интерфейсов. Кроме того, необходима возможность переопределять классы-мэпперы, чтобы обеспечить возможность оптимизации и расширяемости. При этом вышестоящие уровни не потребуют изменений.
- Для оптимизации обращения к хранилищам данных может быть применен образец Unit of Work. Кроме того, этот подход может использоваться для обработки ошибок и позволяет избежать компенсирующих операций при использовании хранилищ, не поддерживающих транзакции.
- Для усиления ограничений целостности на уровне приложения может быть реализован механизм событий, позволяющий проверять целостность данных перед записью.
- Для повышения производительности можно применять кэширование на разных уровнях. Так как NoSQL-системы в большинстве своем не поддерживают строгой согласованности данных, то кэширование не привнесёт в этом случае больших проблем.

5. Заключение

В этой статье был дан обзор основных проблем, возникающих при разработке Web-приложений: работа с данными, вопросы масштабирования и оптимизации, выбор хранилища данных и уровня абстракции для решения определенных задач. В целом можно сделать следующий вывод: выбор конкретной СУБД и методов работы с данными целиком определяется требованиями к приложению по части производительности, масштабируемости, надежности и скорости разработки. Для приложений, требующих транзакционной семантики и строгой согласованности данных по-прежнему лучше всего подходят SQL-ориентированные СУБД, в то время как для высокопроизводительных Web-приложений с высокими требованиями к горизонтальной масштабируемости лучше использовать распределённые SQL-ориентированные или NoSQL-системы.

Универсального решения проблемы работы с данными не существует, однако с помощью применения слоя абстракции можно облегчить разработку приложений при использовании нескольких хранилищ данных, а также сделать приложение более переносимым.

Список литературы

[1] M. Fowler, Patterns of Enterprise Application Architecture, Addison Wesley, 2002.
[2] «Zend Framework», [В Интернете]. URL: http://framework.zend.com/.
[3] «Hibernate - Relational Persistence for Java and .NET», [В Интернете]. URL: http://hibernate.org/.
[4] «Doctrine Project», [В Интернете]. URL: http://www.doctrine-project.org/.

[5] «The case against ORM Frameworks in High Scalability Architectures», 2008. [В Интернете]. URL: http://highscalability.com/blog/2008/2/2/the-case-against-orm-frameworks-in-high-scalability-architec.html.
[6] J. Miller, «Design Patterns for Data Persistence», 2009. [В Интернете]. URL: http://msdn.microsoft.com/en-us/magazine/dd569757.aspx.
[7] A. Wiggins, «SQL Databases Don't Scale», 2009. [В Интернете]. URL: http://adam.heroku.com/past/2009/7/6/sql_databases_dont_scale/.
[8] D. Obasanjo, «Building scalable databases: Denormalization, the NoSQL movement and Digg», 2009. [В Интернете]. URL: http://www.25hoursaday.com/weblog/2009/09/10/BuildingScalableDatabasesDenormalizationTheNoSQLMovementAndDigg.aspx.
[9] S. Francia, J. Hileman, «Augmenting RDBMS with MongoDB for eCommerce», 2011. [В Интернете]. URL: http://www.nosqldatabases.com/main/2011/4/11/augmenting-rdbms-with-mongodb-for-ecommerce.html.
[10] D. Florescu, D. Kossmann, «Rethinking Cost and Performance of Database Systems», *ACM SIGMOD Record,* т. 38, № 1, 2009.
[11] E. Brewer, «Towards Robust Distributed Systems», в *ACM Symposium on the Principles of Distributed Computing*, Portland, Oregon, 2000.
[12] S. Gilbert, N. Linch, «Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services», 2002.
[13] D. Abadi, «Problems with CAP, and Yahoo's little known NoSQL system», 2010. [В Интернете]. URL: http://dbmsmusings.blogspot.com/2010/04/problems-with-cap-and-yahoos-little.html.
[14] W. Wogels, «Eventually Consistent,» *ACM Queue,* т. 6, № 6, 2008.
[15] H. Wada, A. Fekete, L. Zhaoy, K. Lee, A. Liu, «Data Consistency Properties and the Tradeoffs in Commercial Cloud Storages: the Consumers' Perspective» в *Conference on Innovative Data Systems Research*, 2011.
[16] «MongoMapper», [В Интернете]. URL: http://mongomapper.com/.
[17] R. Cattell, «Scalable SQL and NoSQL Data Stores», 2011. [В Интернете]. URL: http://www.cattell.net/datastores/Datastores.pdf.
[18] А. В. Посконин, «Новые направления в развитии СУБД: компромисс как основа архитектуры» в *Тезисы лучших дипломных работ факультета ВМК МГУ 2011 года*, Москва, 2011.
[19] «MongoDB», [В Интернете]. URL: http://www.mongodb.org/.
[20] «Membase Server», [В Интернете]. URL: http://www.couchbase.com/membase.
[21] M. Stonebraker, U. Çetintemel, «"One Size Fits All": An Idea Whose Time Has Come and Gone» в *ICDE '05: Proceedings of the 21st International Conference on Data Engineering*, Washington, 2005.
[22] M. Fowler, «Polyglot Persistence», 2011. [В Интернете]. URL: http://martinfowler.com/bliki/PolyglotPersistence.html.
[23] A. Snell-Pym, «One Database, Many Interfaces: A Look at SQL/NoSQL Integration Using GenieDB», *Cloudbook Journal,* т. 2, № 1, 2011.
[24] «Tungsten Replicator», [В Интернете]. URL: http://continuent.com/solutions/tungsten-replicator.
[25] M. Fowler, «Command Query Responsibility Segregation», 2011. [В Интернете]. URL: http://martinfowler.com/bliki/CQRS.html.
[26] «NoRM - MongoDB Driver for .NET», [В Интернете]. URL: https://github.com/atheken/NoRM.

[27] «DataMapper - Ruby Object Relational Mapper», [В Интернете]. URL: http://datamapper.org.