SQL и NoSQL: пути сотрудничества

Андрей Посконин

аспирант кафедры Системного программирования ВМК МГУ

Московская секция ACM SIGMOD
27 марта 2014

Содержание

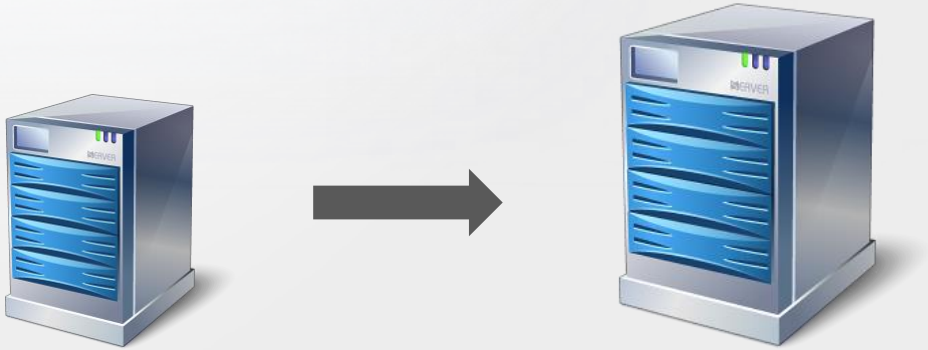
- Введение. Современные приложения и сервисы
- RDBMS или NoSQL – что выбрать?
- SQL + NoSQL = ?
- Объектное отображение в приложениях
- SQL + NoSQL на уровне объектного отображения
- Заключение

Современные приложения и сервисы

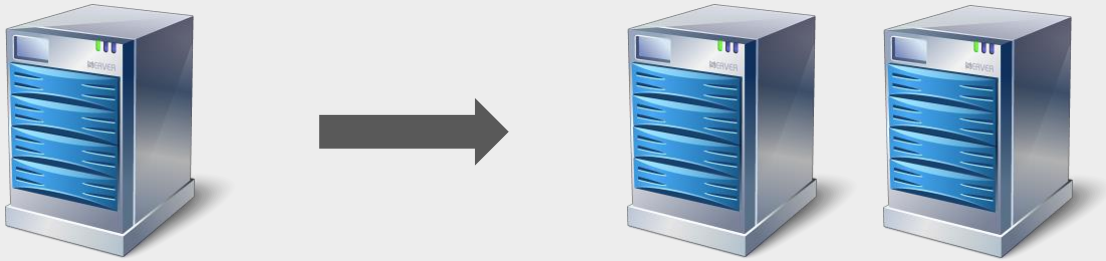
- Быстрый рост – проблема масштабируемости
  - Рост объёма данных
  - Рост нагрузок
- Быстрая разработка прототипа
- Расширяемость и поддержка
- Оптимизация финансовых затрат
- Отказоустойчивость
- Опыт взаимодействия (User Experience)

Современные приложения и сервисы

- Вертикальное масштабирование (Scale-Up)



- Горизонтальное масштабирование (Scale-Out)

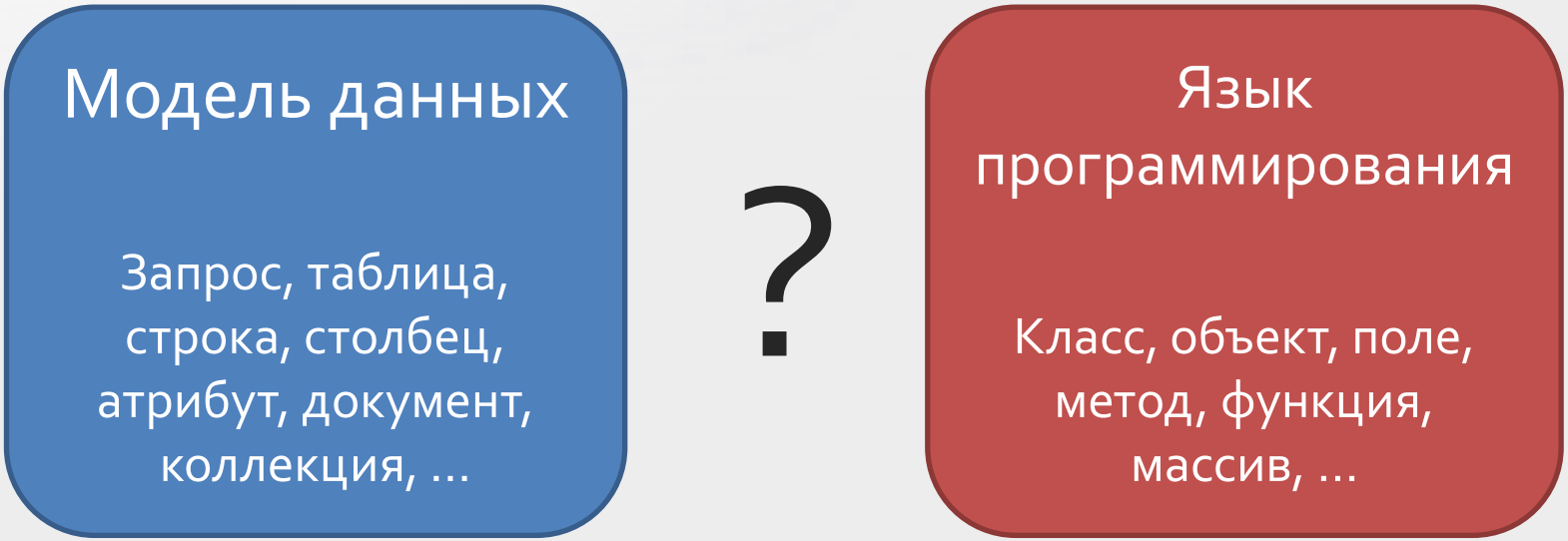


Современные приложения и сервисы

- Работа с неоднородными данными
  - Структурированные и неструктурированные данные
  - Соотношение операций чтения/записи
  - Время отклика и надежность
  - Необходимость транзакционной семантики
  - Объём данных
  - Нагрузка на систему
  - Возможности языка запросов
  - …

Современные приложения и сервисы

- Impedance Mismatch



Современные приложения и сервисы

Данные

Бизнес-логика

Клиенты

Традиционные RDBMS

- Строгая схема базы данных
- ACID-транзакции
- Используются более 40 лет
- Развитый инструментарий
- Богатая функциональность
- Наличие стандартов

Традиционные RDBMS

- Вертикальное масштабирование
- Горизонтальное масштабирование
  - Репликация
    - Master-Slave
  - Шардинг (разделение данных)
    - Непрозрачно для приложения
    - Проблемы с транзакциями и контролем целостности

SQL-ориентированные СУБД

- Оптимизация
  - Оптимизация запросов
  - Создание индексов
  - Оптимизация транзакций
  - Кэширование
  - Ослабление ACID
  - Денормализация

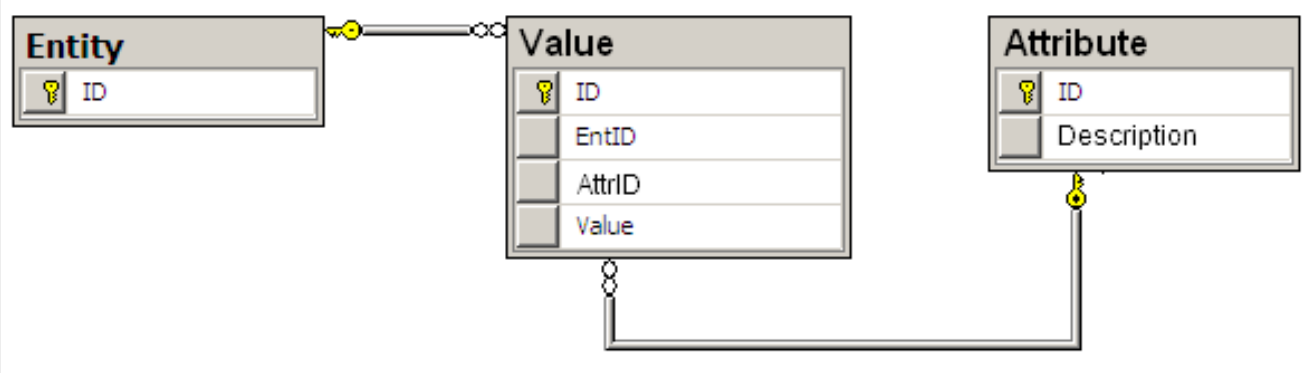
Отказ от основ RDBMS

SQL-ориентированные СУБД

- Модель данных
  - Нормализация
    - Декомпозиция сложной сущности
    - JOIN – дорогая операция
    - Отказ от нормализации ради быстродействия
  - Строгая схема базы данных (таблица с фиксированным набором столбцов)
    - Динамические атрибуты?

SQL-ориентированные СУБД

- Динамические атрибуты средствами SQL
  - «Широкая» разреженная таблица
  - Таблица для каждого подтипа
  - Entity-Attribute-Value



Новые RDBMS

- Промежуточное ПО для масштабирования поверх традиционных RDBMS
- NewSQL– масштабируемый SQL
  - Шардинг и репликация
  - Поддержка ACID-транзакций
  - Предпочтительны «узкие» транзакции
  - Проектирование и развертывание усложняются
  - Поддерживается не вся функциональность SQL

NoSQL

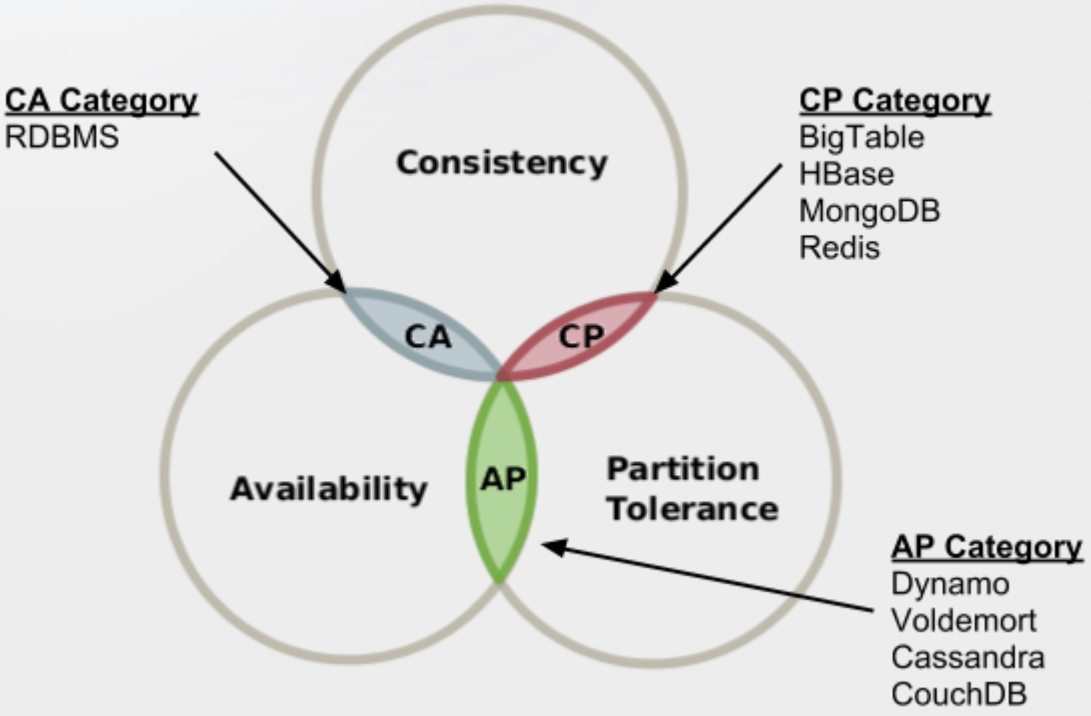
- **N**ot **O**nly **SQL**
- Нереляционные модели данных
- Распределённая архитектура
- Горизонтальная масштабируемость
- Ослабленные гарантии согласованности данных
- Нет ACID-транзакций
- Большое разнообразие решений

NoSQL: модели данных

- Ключ-значение
  - Поиск и модификация по уникальному ключу
- Документная
  - Документ – это объект с произвольным набором атрибутов
  - Поиск по сочетанию атрибутов, индексы
- Google BigTable
  - Горизонтально и вертикально разделённая таблица

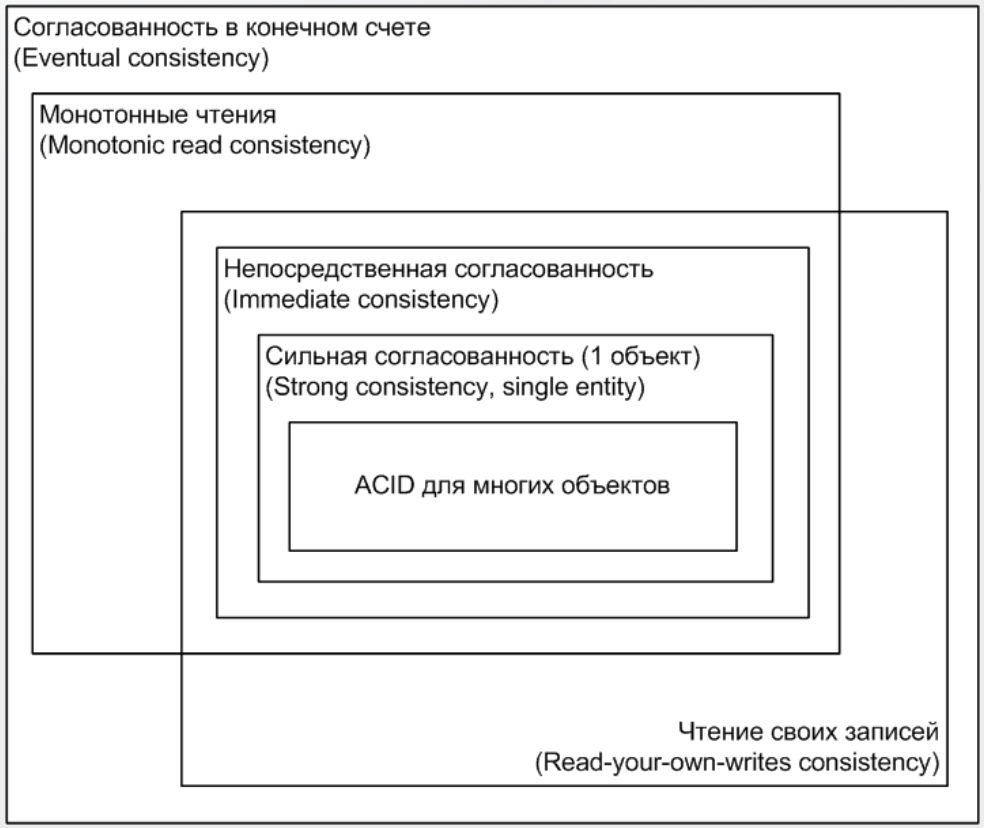
NoSQL: согласованность данных

- Теорема CAP (E. Brewer, 2000)



NoSQL: согласованность данных

- Модели согласованности



NoSQL: MongoDB

- Документная модель данных (JSON)

NoSQL: MongoDB

- Нет строгой схемы
- Запросы по образцу
- MapReduce, Aggregation Framework
- Поддержка индексов
- Репликация и шардинг
- Нет ACID-транзакций
- Атомарность на уровне одного документа
- Настраиваемые параметры надежности
- Богатая функциональность (пространственные индексы, GridFS и т.д.)

NoSQL: MongoDB

- Транзакции
  - На уровне одного документа все операции атомарны
  - Версии, Compare-and-Set для оптимистических блокировок
  - Durability выполняется при включенном журналировании
  - Реализация полноценных транзакций сложна и неэффективна

SQL или NoSQL?

- Выбор системы хранения определяется задачей
  - Структура данных
  - Требования к масштабируемости
  - Необходимость полноценных транзакций
  - Гарантии согласованности
  - Надежность и отказоустойчивость
  - Возможности языка запросов
- Универсальной СУБД не существует

SQL + NoSQL

- Постепенное сближение SQL и NoSQL
  - Усиление согласованности в NoSQL-системах
  - Появление NewSQL-систем
  - Поддержка как SQL, так и NoSQL-интерфейсов для доступа к данным
  - Появление трансляторов SQL в запросы к NoSQL-системам и MapReduce
  - UnQL (Unstructured Query Language) – SQL-подобный язык запросов

SQL + NoSQL

- Использование NoSQL (преимущественно Key-Value-систем) в качестве кэша, чтобы разгрузить RDBMS
- Репликация между SQL и NoSQL-системами (для последующей обработки или хранения)
- Использование нескольких систем хранения данных в одном приложении (Polyglot Persistence)

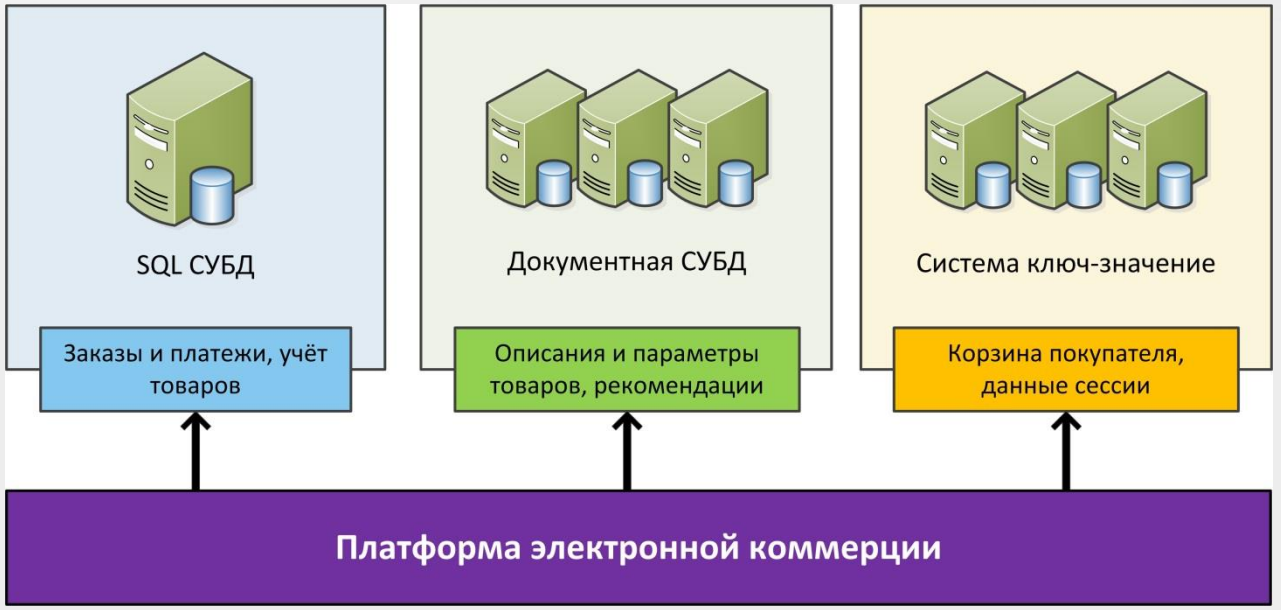
Polyglot Persistence

Polyglot Persistence

- Несколько систем хранения данных в одном приложении
  - Каждая подзадача решается эффективно
  - Минимизируются компромиссы и ручная реализация недостающей функциональности
  - Повышается сложность приложения
  - Различные интерфейсы доступа к данным

Polyglot Persistence

- Service-Oriented Architecture (SOA) как способ борьбы со сложностью



Polyglot Persistence

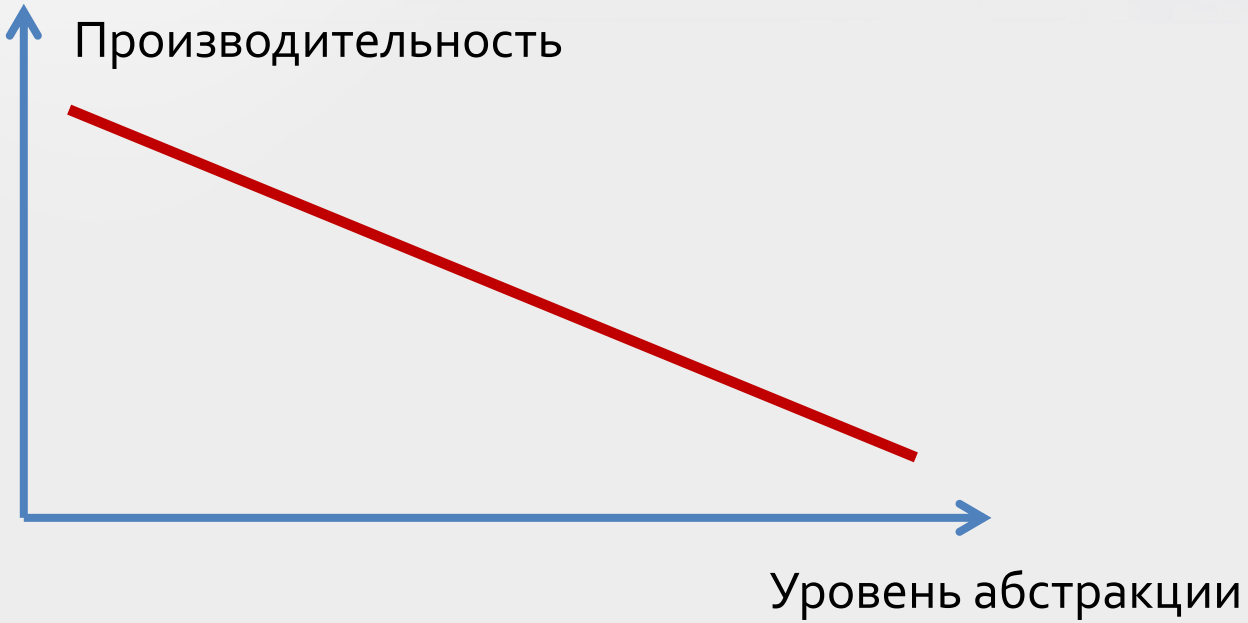
- Service-Oriented Architecture (SOA)
  - Все детали работы с данными скрываются за интерфейсом сервиса
  - Возможность удобно распределять обязанности разработчиков
  - Больше работы и накладных расходов на реализацию абстракции и взаимодействия
  - Нецелесообразно для небольших проектов

Объектное отображение

- Чаще всего для разработки приложений применяются объектно-ориентированные языки
- Требуется отображение объектной модели на модель данных целевой СУБД
- Отображение может быть реализовано вручную, либо с использованием библиотек
- Возможны различные уровни абстракции

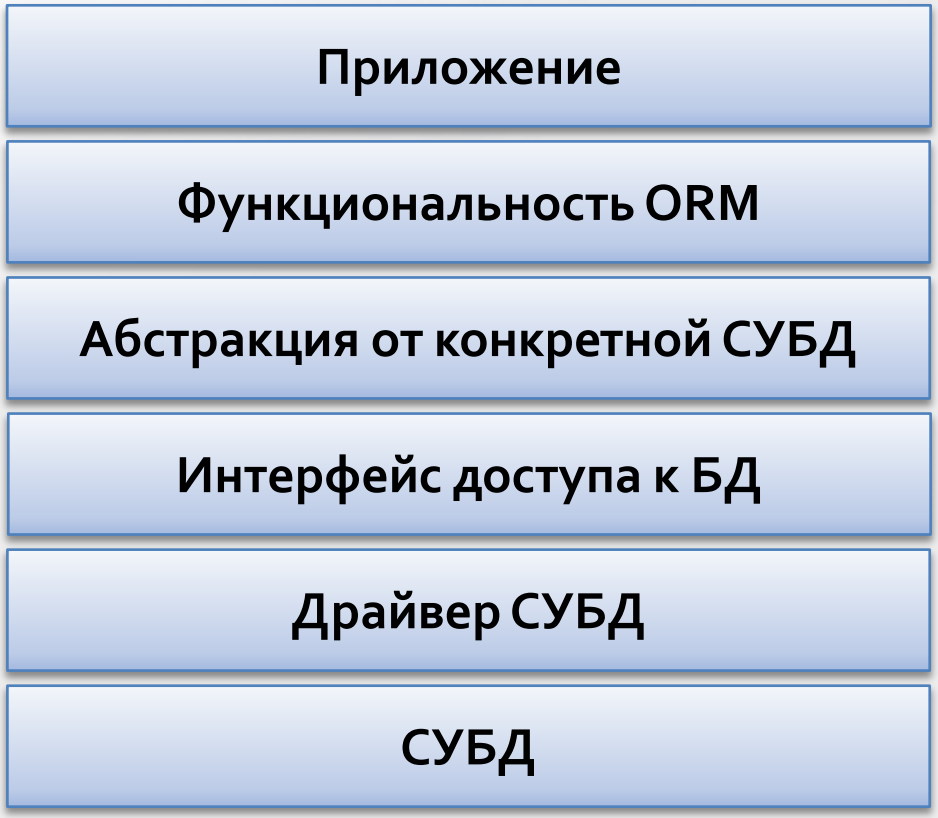
Абстракция или производительность?

- Большое количество промежуточных слоев между приложением и системой хранения ведёт к падению производительности и сложности оптимизации.



Object-Relational Mapping (ORM)

- Высокий уровень абстракции ORM-библиотек



Object-Relational Mapping (ORM)

- Высокий уровень абстракции ORM-библиотек
  - Высокоуровневый интерфейс
  - Не зависят от особенностей целевой СУБД
  - Управляют жизненным циклом объектов
  - Предоставляют «объектный» язык запросов
  - Трудно оптимизировать под конкретную СУБД
  - Легко написать неэффективный код
  - Низкая производительность по сравнению с работой напрямую с драйвером СУБД

Object-Document Mapping (ODM)

- ODM требует гораздо меньше преобразований (Impedance Mismatch меньше)
- Поддерживаются вложенные объекты, списки и другие «денормализованные» структуры
- Каждая библиотека работает только с одной конкретной документной СУБД (например, MongoDB)

Библиотеки объектного отображения

- Нацелены на один тип систем
- Сложно расширять и оптимизировать
- Трудно использовать в высоконагруженных приложениях
- Уменьшают Impedance Mismatch
- Ускоряют и упрощают разработку приложения
- Уровень объектного отображения естественным образом подходит для интеграции систем хранения

Интеграция на уровне отображения

- Отображение на различные системы хранения
- Простота
- Быстрая разработка приложений
- Настраиваемый уровень абстракции
- Возможность оптимизации
- Производительность
- Расширяемость

Интеграция на уровне отображения

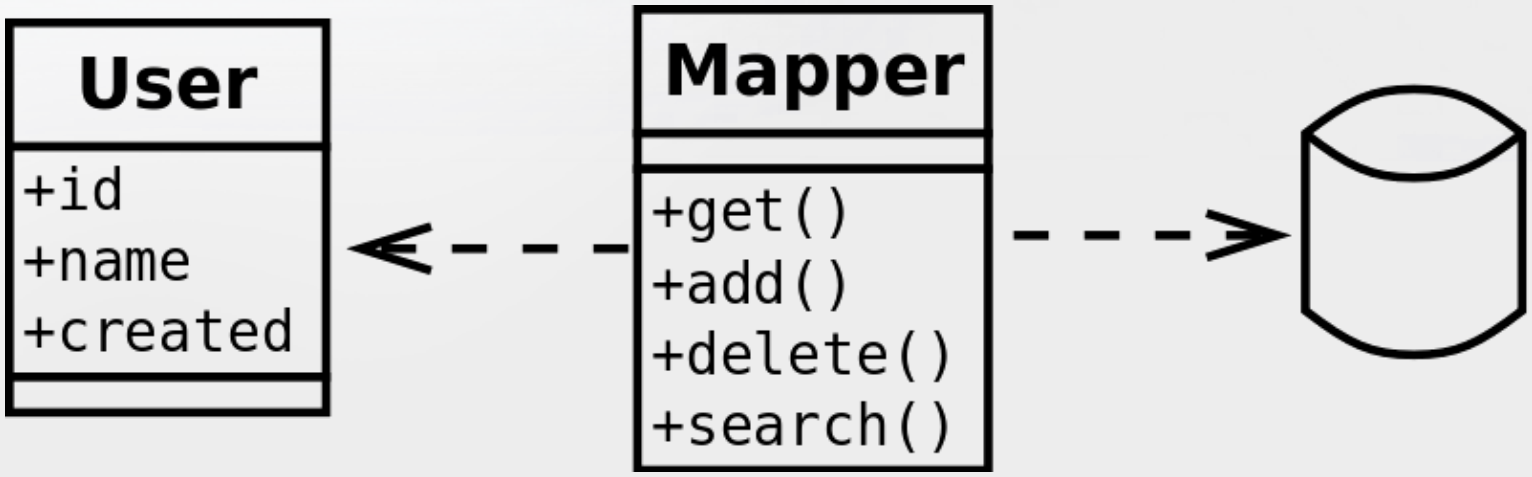
- Существующие решения
  - Hibernate OGM (Object/Grid Mapper) – позволяет использовать несколько NoSQL-систем с JPA, работа вместе с RDBMS пока не поддерживается
  - Doctrine Project: ORM, MongoDB ODM, CouchDB ODM
  - LINQ to SQL, FluentMongo с поддержкой LINQ
  - Адаптеры для некоторых библиотек и программных каркасов

Интеграция на уровне отображения

- Реализация: MapperStack – библиотека/программный каркас для Web-приложений
  - Прототип
  - Язык реализации - PHP 5.3
  - В настоящий момент поддерживается работа с MySQL и MongoDB
  - Высокоуровневый язык запросов
  - Модульная архитектура

Интеграция на уровне отображения

- Data Mapper



Дополнительный слой ответственен за отображение объектов предметной области

Пример. Сущность BlogPost

```php
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');
    }

    // ...
}
```

Пример. Инициализация

```php
$m = new MapperStack();

// Установка кэша по умолчанию
$m->setDefaultCache(new ApcCache());

// Создание соединений
$mongoConnection = new MongoConnection('mongodb://localhost:27017');
$mySqlConnection = new SQLConnection(
    array(
        'driver' => 'mysql',
        'host' => 'localhost',
        'dbname' => 'db',
        'user' => 'user',
        'password' => 'password'
    ));

// Добавление отображений для сущностей
$m->addMapper(new MongoMapper('Blog\BlogPost', $mongoConnection));
$m->addMapper(new SQLMapper('Blog\User', $mySqlConnection));
```

Пример. Запросы

```
// Сущность User отображается с использованием SQLMapper на таблицу MySQL
$users = $m->query('Blog\User')
          ->filter('birthDate > :date && !like(email, "%@gmail.com")')
          ->bind('date', $date)
          ->sort('birthDate', 'asc')
          ->project('username')
          ->limit(10)
          ->getResult();

// Сущность BlogPost отображается с использованием MongoMapper
// на коллекцию JSON-документов MongoDB
$post = $m->query('Blog\BlogPost')
          ->filter('username == "admin" &&
                    tags {
                        @element == "politics"
                    } &&
                    comments {
                        regexp(text, "/Ukraine/i")
                    }')
          ->limit(1)
          ->cache(3600)
          ->getSingleResult();
```

Пример. Создание, изменение, удаление

```php
$mySqlConnection->beginTransaction(); // Начать транзакцию MySQL

// Удалить найденных пользователей
foreach ($users as $user)
{
    $m->delete($user);
}

// Создать нового пользователя
$user = new User("test@test.com");

$m->save($user); // Отметить объект как требующий сохранения

$post->setUsername("administrator"); // Модификация данных

$m->save($post); // Отметить объект как требующий сохранения

$m->flush(); // Выполнить все отложенные операции записи

$mySqlConnection->commit(); // Завершить транзакцию MySQL
```

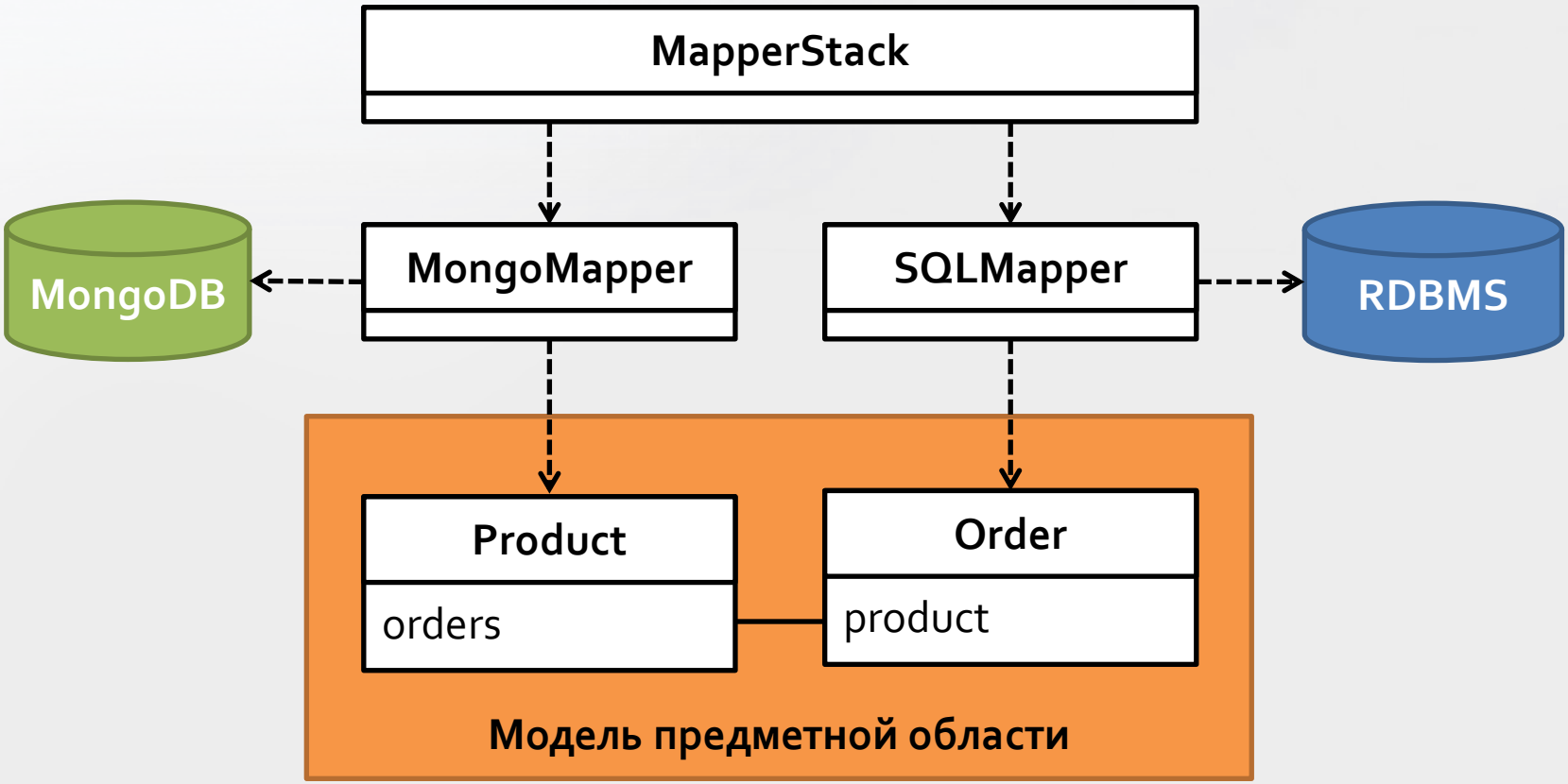
Сущность

- Сущность (Entity)
  - В отличие от ORM может быть графом объектов
  - Помимо полей скалярных типов может содержать:
    - Списки
    - Объекты
    - Списки объектов
    - …
  - Может отображаться на JSON-документ
  - Таблица – частный случай JSON-документа

Сущность

- Сущность (Entity)
  - Обладает уникальным идентификатором (id)
  - Может содержать произвольный набор полей
  - Операции проверки наличия (isset) и удаления поля (unset)
  - NULL и отсутствие поля – разные вещи
  - Может иметь конструктор с параметрами
  - Определяет метаданные для отображения
  - Может содержать код для проверки ограничений целостности данных

Отображение сущностей



Реализация отображения

- Mapper – объект, осуществляющий отображение
  - Должен реализовывать интерфейс Mapper (например, методы find(), findBy(), findOneBy(), getEntityId() и т.д.)
  - Может использоваться независимо
  - Стандартные реализации – SQLMapper, MongoMapper
  - Могут быть расширены пользователем
  - Возможны полностью пользовательские реализации отображения

Метаданные

- Метаданные описывают отображение сущностей. Например:
  - Соответствие полей объекта и колонок/атрибутов
  - Отображение типов
  - Поле уникального идентификатора
  - Таблица/коллекция для отображения
  - Требуют эффективного доступа
  - Стандартные реализации: MetaClass, EntityMetaClass

Реализация отображения

- Тип – определяет отображение типа данных системы хранения на тип данных языка программирования и обратно
- Например, IP-адреса эффективнее хранить как BIGINT, но в приложении удобнее иметь дело со строковым представлением
- Стандартные типы (integer, string, …)
- Возможно определение пользовательских типов или работа без преобразования типов

Реализация отображения

- Преобразование из «сырых» данных в объекты
  - Преобразование типов данных (transform)
  - Построение объектов (hydrate)
- Преобразование объектов в «сырые» данные
  - [Построение списка изменений (changeset)]
  - Извлечение данных из объектов (extract)
  - Обратное преобразование типов данных (transform)
- Могут использоваться как стандартные реализации, так и пользовательские

Реализация отображения

- Отслеживание изменений
  - **Явное** (сущность сама генерирует события изменения полей). Низкая производительность при множественных изменениях полей.
  - **Неявное** (исходные данные сохраняются, а затем сравниваются с новым состоянием сущности). Накладные расходы на сравнение всех полей при небольших изменениях. Память не дублируется (copy-on-write).

Реализация отображения

- Unit of Work
  - Все операции изменения откладываются до вызова метода flush(), который выполняет их в нужном порядке
  - Каждый Mapper реализует UoW для управляемых им сущностей
  - Объект класса MapperStack координирует работу Mapper-объектов, позволяя учитывать ассоциации между сущностями и реализует глобальный UoW

Ассоциации

- Могут связывать сущности, отображаемые на разные системы хранения
- Реализуются поверх слоя отображения, используются методы find(), findBy(), findOneBy()
- Соответствующие «внешние ключи» получают актуальные значения на при вызове метода MapperStack::flush()
- «Ленивая» загрузка сущностей
- Не требуется генерация кода
- Owning/Inverse-стороны связи

Ассоциации

```php
class Message extends MapperStack\Object
{
    protected $userId;

    public static function associate(AssociationBuilder $ab)
    {
        $ab->manyToOne('user', 'Domain\User', 'userId');
    }

    public function setUser(User $user)
    {
        $this->user = $user;
    }

    public function getUser()
    {
        return $this->user;
    }

    // ...
```

Наследование

- Динамические атрибуты позволяют в некоторых случаях избежать наследования

- Полноценное наследование может быть реализовано с использованием атрибутов-дискриминаторов и поддержки со стороны компонентов библиотеки

Унифицированный язык запросов

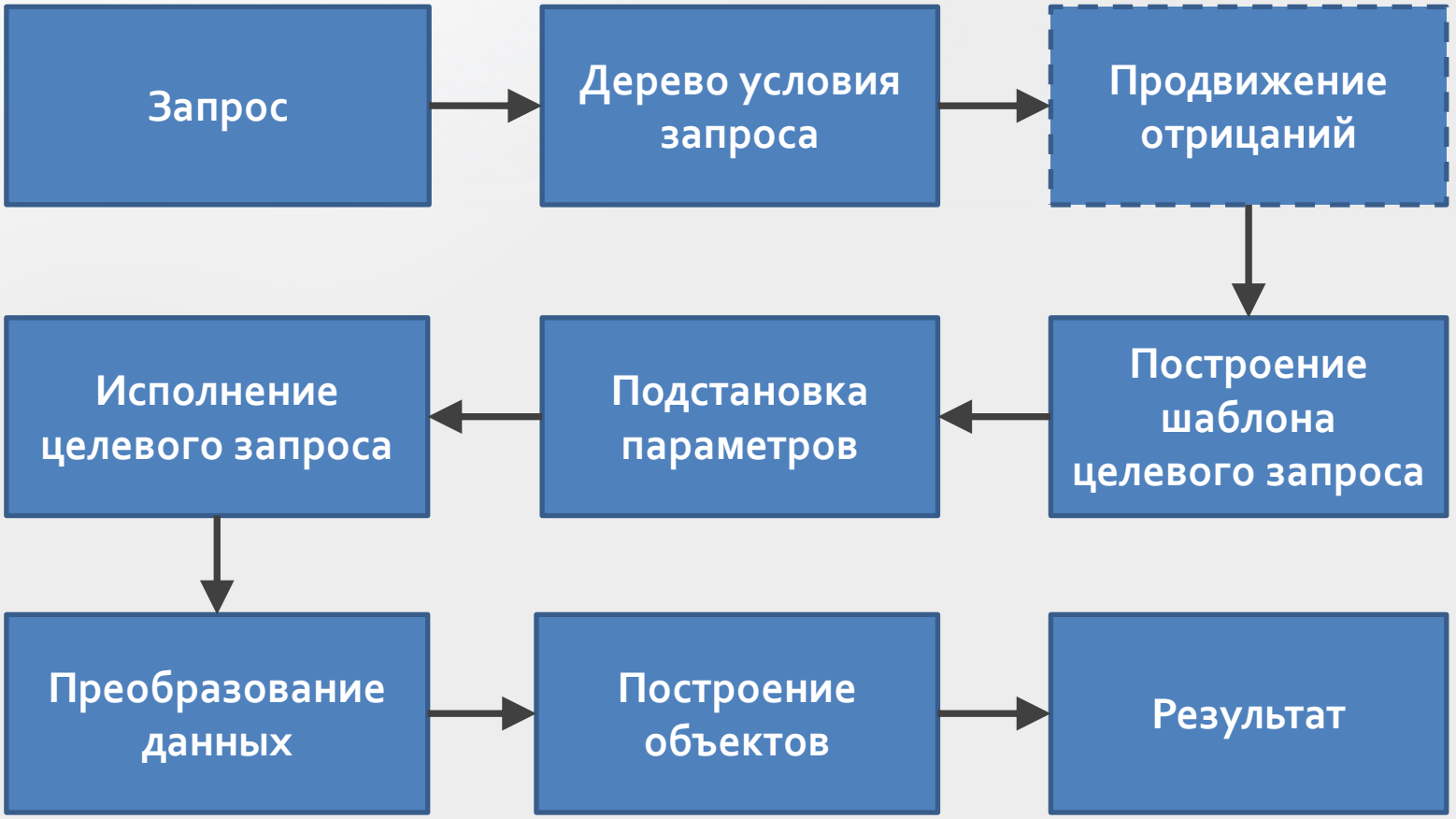
- Синтаксис не связан с SQL или другим языком запросов, например, JSONiq
- Запросы в терминах сущностей
- Покрывает базовые потребности по выборке объектов
- Основные операции:
  - filter, sort, skip, limit, project

Унифицированный язык запросов

- Filter – позволяет задать логическое выражение для выбора подмножества сущностей
  - Сравнение значения поля и параметра
  - Операции И (&&), ИЛИ (||), НЕ (!)
  - Предикаты, в том числе пользовательские (regexp, …)
  - Пути в графе объектов (comment.user.name)
  - Сопоставление элементов списка (tags{ … })
  - Поддержка встроенных и связываемых параметров

Унифицированный язык запросов

- Трансляция и исполнение запросов



Унифицированный язык запросов

- Связываемые параметры позволяют кэшировать шаблон целевого запроса и не производить лексический и синтаксический анализ в следующий раз
- Язык запросов может быть расширен добавлением новых предикатов
- Для сложных запросов эффективнее использовать «родной» язык запросов целевой системы хранения

Унифицированный язык запросов

```
$p = $q->filter('price >= 1000 && type == "shoes"')
       ->sort('price', 'asc')->limit(3)
       ->getResult();
```

MySQL:

```
SELECT * FROM `products`
    WHERE `price` > 1000 AND `type` = 'shoes'
    ORDER BY `price` LIMIT 3
```

MongoDB:

```
db.products.find( {
    $and : [
        { 'price' : { $gte : 1000} },
        { 'type' : 'shoes' }
    ] } ).sort( { 'price' : 1 } ).limit(3);
```

Унифицированный язык запросов

- Пример (MongoDB):

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

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

Кэширование

- Адаптеры для различных систем кэширования (Memcache, APC и т. д.)
- Кэш первого уровня (Identity Map)
- Кэш результатов запросов
- Кэш второго уровня может быть реализован на уровне отображения
- Кэширование служебной информации

Целостность данных

- Отложенное исполнение операций модификации
- Возможность проверки целостности данных перед записью
- Возможность использовать транзакции на уровне соединения с сервером СУБД
- Возможна реализация оптимистических блокировок

Дальнейшее развитие

- Поддержка большего числа систем и источников данных (например, REST-сервисов)
- Оптимизация
- Поддержка наследования
- Поддержка агрегатных функций
- Поддержка оптимистических блокировок
- Поддержка MapReduce
- Модификация без чтения
- …

Заключение

- Соединение в приложениях SQL и NoSQL-решений открывает новые возможности
- Интеграция SQL и NoSQL может быть осуществлена на уровне объектного отображения
- Отсутствие стандартизации существенно усложняет интеграцию

Спасибо за внимание!