Object-relational impedance mismatch
Объектно-реляционная несогласованность

Из Википедии, свободной энциклопедии

Объектно-реляционная несогласованность представляет собой набор концептуальных и технических трудностей, с которыми часто сталкиваются, когда система управления реляционными базами данных (RDBMS) обслуживается прикладной программой (или несколькими прикладными программами), написанной на объектно-ориентированном языке или в объектно-ориентированном стиле программирования, в частности, потому что объекты или определения классов должны быть отображены на таблицы базы данных, определенные реляционной схемой.

Термин «объектно-реляционная несогласованность» введен по аналогии с согласованием импедансов в электротехнике.

Оглавление

1 Несогласование

Объекты (экземпляры) ссылаются друг на друга и поэтому образуют граф в математическом смысле (сеть, включающая петли и циклы). Реляционные схемы, напротив, являются табличными и основаны на реляционной алгебре, которая определяет связанные неоднородные кортежи (группировки полей данных в «строку» с разными типами для каждого поля).

1.1 Объектно-ориентированные концепции

1.1.1 Инкапсуляция

Объектно-ориентированные программы хорошо разрабатываются с использованием техник, которые приводят к инкапсулированным объектам, внутреннее представление которых может быть скрыто. Ожидается, что в объектно-ориентированной среде базовые свойства данного объекта не будут открыты какому-либо интерфейсу, кроме того, который реали-

зован вместе с объектом. Тем не менее, объектно-реляционное отображение обязательно раскрывает основное содержимое объекта для взаимодействия с интерфейсом, который не может точно определить реализация объекта. Следовательно, объектно-реляционное отображение нарушает инкапсуляцию объекта, поскольку многие объектно-реляционные преобразователи автоматически генерируют открытые поля, соответствующие столбцам базы данных.

1.1.2 Доступность

При реляционном ходе мыслей «частный (private)» или «публичный (public)» доступ связывается с потребностями. В объектно-ориентированной (ОО) модели это абсолютная характеристика состояния данных. В реляционных и ОО-моделях часто возникают противоречия между относительностью и абсолютизмом классификаций и характеристик.

1.1.3 Интерфейс, класс, наследование и полиморфизм

В рамках объектно-ориентированной парадигмы объекты имеют интерфейсы, только посредством которых обеспечивается доступ к внутренним свойствам этого объекта. Реляционная модель, с другой стороны, использует производные переменные значения отношений (представления), с тем чтобы обеспечить различные перспективы и ограничения с целью обеспечить целостность. Также, основные концепции ООП для классов объектов, наследования и полиморфизма не поддерживаются системами реляционных баз данных.

1.1.4 Отображение на реляционные концепции

Правильное соответствие между реляционными и объектно-ориентированными концепциями может быть достигнуто, если таблицы реляционной базы данных связываются [требуется дополнительное объяснение] со связями, обнаруженными в объектно-ориентированном анализе.

1.2 Различия типов данных

Главным несоответствием между существующими реляционными и ОО-языками являются различия в системе типов. Реляционная модель строго запрещает ссылочные атрибуты (или указатели), тогда как ОО-языки включают в себя и предполагают использование ссылок. Скалярные типы и их операторная семантика могут значительно различаться между моделями, вызывая проблемы при отображении.

Например, большинство SQL систем поддерживают строковые типы с различающимися параметрами сортировки (collations) и ограничениями на максимальную длину (неограниченные текстовые типы, как правило, снижают производительность), в то время как большинство языков ОО рассматривают параметры сортировки только как аргумент для подпрограмм сортировки, а строки содержат собственный размер в отведенной для них памяти. Более коварный, но имеющий отношение к делу пример состоит в том, что системы SQL часто игнорируют конечные пробелы в строках при сравнении, тогда как библиотеки строк ОО этого не делают. Как правило, невозможно создать новые типы данных из-за ограничения на возможные значения других примитивных типов в языке ОО.

1.3 Структурные и целостные различия

Другое несоответствие связано с различиями в структурных аспектах и аспектах целостности сопоставляемых моделей. В ОО-языках объекты могут составляться из других объектов — часто высокой степени вложенности — или конкретизироваться из более общего определения. Это может сделать отображение на реляционные схемы менее простым. Это связано с тем, что реляционные данные, как правило, представляются в виде именованного набора глобальных, невложенных переменных отношения. Сами отношения, представляющие собой наборы кортежей, все соответствующие одному и тому же заголовку (см. Реляционное исчисление кортежей), не имеют идеального аналога в ОО-языках. Ограничения в ОО-

языках, как правило, не объявляются как таковые, но проявляются в виде защитной логики выброса исключений для окружающего кода, который работает с инкапсулированными внутренними данными. С другой стороны, реляционная модель требует декларативных ограничений на скалярные типы, атрибуты, переменные отношения и базу данных в целом.

1.4 Манипулятивные различия

Однако, особенно заметны семантические различия в манипулятивных аспектах противопоставляемых моделей.

Реляционная модель имеет встроенный, относительно небольшой и четко определенный набор примитивных операторов для использования в запросах и манипулировании данными, в то время как ОО-языки обычно обрабатывают запросы и выполняют манипуляции с помощью специально созданных или низкоуровневых, зависящих от ситуации и особенностей физического доступа императивных операций.

Некоторые ОО-языки действительно поддерживают декларативные подъязыки запросов, но поскольку ОО-языки обычно работают со списками и, возможно, хеш-таблицами, манипулятивные примитивы обязательно отличаются от операций на основе множеств реляционной модели. [нужна цитата]

1.5 Транзакционные различия

Параллелизм и аспекты транзакций также существенно различаются. В частности, транзакции, наименьшие элементы работы, выполняемые базами данных, намного больше представлены в реляционных базах данных, чем любые операции, выполняемые классами на ОО-языках. Транзакции в реляционных базах данных представляют собой динамически ограниченные наборы произвольных манипуляций с данными, тогда как степень детализации транзакций на языке ОО обычно находится на уровне отдельных назначений полей примитивного типа. В целом, у ОО-языков нет аналогов изоляции или долговечности, поэтому атомарность и согласованность обеспечиваются только при записи в поля этих примитивных типов.

2 Устранение несогласованности

Обход проблемы несогласованности для объектно-ориентированных программ начинается с распознавания различий в конкретных используемых логических системах. Тогда несогласованност либо минимизируется, либо компенсируется.

2.1 Альтернативные архитектуры

Проблема объектно-реляционной несогласованности не является универсальной проблемой между ОО и базами данных. Как следует из названия, эта проблема несогласованности возникает только с реляционными базами данных. Наиболее распространенным решением этой проблемы является использование альтернативной базы данных, такой как база данных NoSQL или XML.

2.2 Минимизация

Были предприняты некоторые попытки создания объектно-ориентированных систем управления базами данных (OODBMS), которые позволили бы избежать проблемы несогласованности. Однако на практике они оказались менее успешными, чем реляционные базы данных, отчасти из-за ограничений принципов ОО как основы для модели данных. [1] Были проведены исследования по расширению для ОО-языков возможностей, аналогичных тем, которыми обладают базы данных, с помощью таких понятий, как транзакционная память[1].

1 В компьютерных технологиях, программная транзакционная память (англ. software transactional memory, STM) представляет собой механизм управления параллелизмом, аналогичный механизму транзакций баз

Одним из распространенных решений проблемы несогласованности является введение слоя доменной[2] и рамочной логики (domain and framework logic). В данной схеме язык ОО используется для моделирования определенных реляционных аспектов во время выполнения, а не для попытки более статического отображения. Фреймворки, использующие этот метод, обычно имеют аналог для кортежа, обычно в виде «строки» в компоненте «набора данных» или в качестве общего класса «экземпляра сущности», а также аналог для отношения. Преимущества этого подхода могут включать в себя:

– простые пути для создания шаблонной структуры и автоматизации того, что касается транспорта, представления и проверки данных домена;
– меньший размер кода — более быстрая компиляция и загрузка;
– возможность динамического изменения схемы[3];
– избежание проблем с пространством имен и семантическими несоответствиями;
– выразительная проверка ограничений;
– не требуется сложное отображение.

Недостатки могут включать в себя:

– отсутствие статических проверок типа «безопасность». Иногда используются типизированные средства доступа как один из способов смягчить этот момент;
– возможные издержки эффективности, связанные с созданием и доступом во время выполнения;
– невозможность естественным образом использовать уникальные для ОО аспекты, такие как полиморфизм.

2.3 Компенсация

Смешивание уровней дискурса в коде приложения ОО представляет проблемы, но есть некоторые общие механизмы, используемые для компенсации. Самая большая проблема заключается в обеспечении поддержки шаблонной структуры, автоматизации манипулирования данными и шаблонов представления на уровне дискурса, на котором моделируются данные предметной области. Для решения этой проблемы используется рефлексия[4] и/или генерация кода. Рефлексия позволяет обращаться к коду (классам) как к данным и таким образом обеспечивать автоматизацию передачи, представления, целостности и т. д. данных. Генерация решает проблему путем обращения к структурам сущностей в качестве входных данных для инструментов генерации кода или языков метапрограммирования, которые создают классы и поддерживают инфраструктуру в целом. Обе эти схемы могут все еще быть предметами аномального поведения, если эти уровни дискурса сливаются. Например, сгенерированные классы сущностей обычно будут иметь свойства, которые сопоставляются с доменом (например, имя, адрес), а также свойства, которые обеспечивают управление состоянием и другую инфраструктуру инфраструктуры (например, IsModified).

данных для управления доступом к совместно используемой памяти в параллельных вычислениях. Это альтернатива для синхронизации на основе блокировки. Транзакция в этом контексте является частью кода, который выполняет считывание и запись в разделяемую (совместно используемую) память. Считывание и запись логически происходит в единичный момент времени, а промежуточные состояния невидимы для других (результативных) транзакций.

2 Домен в реляционной модели данных — тип данных, то есть множество допустимых значений. Понятие типа данных является фундаментальным; каждое значение, каждая переменная, каждый параметр, каждый оператор чтения, и особенно каждый реляционный атрибут относится к тому или иному типу.

3 Схема базы данных включает в себя описания содержания, структуры и ограничений целостности, используемые для создания и поддержки базы данных.

4 В информатике отражение или рефлексия (холоним интроспекции, англ. reflection) означает процесс, во время которого программа может отслеживать и модифицировать собственную структуру и поведение во время выполнения. Парадигма программирования, положенная в основу отражения, называется рефлексивным программированием. Это один из видов метапрограммирования.

3 Точки зрения

Кристофер Дж. Дейт и другие утверждали, что действительно реляционная СУБД не мможет создавать такие проблемы [2-4], поскольку домены и классы — это одно и то же. Естественное отображение между классами и реляционными схемами является фундаментальной ошибкой проектирования [5]; и что отдельные кортежи в таблице (отношении) базы данных должны рассматриваться как устанавливающие отношения между сущностями; не как представления для самих сложных объектов. Однако эта точка зрения имеет тенденцию уменьшать влияние и роль объектно-ориентированного программирования, используя его как нечто большее, чем система управления типами полей.

Несогласованность состоит в программировании между объектами домена и пользовательским интерфейсом. Сложные пользовательские интерфейсы, позволяющие операторам, менеджерам и другим непрограммистам получать доступ к записям в базе данных и манипулировать ими, часто требуют глубоких знаний о природе различных атрибутов базы данных (помимо имени и типа). В частности, считается хорошей практикой (с точки зрения производительности для конечного пользователя) разрабатывать пользовательские интерфейсы таким образом, чтобы пользовательский интерфейс предотвращал ввод незаконных транзакций (тех, которые приводят к нарушению ограничения базы данных); для этого требуется, чтобы большая часть логики, присутствующей в реляционных схемах, была продублирована в коде.

В некоторых средах для разработки кода могут использоваться определенные формы логики, представленные в схеме базы данных (например, ограничения ссылочной целостности), так что такие проблемы решаются общим и стандартным способом с помощью библиотечных процедур, а не специального кода, написанного для случая на индивидуальной основе.

Утверждалось, что SQL из-за очень ограниченного набора доменных типов (и других предполагаемых недостатков) затрудняет надлежащее моделирование объектов и доменов; и что SQL представляет собой очень ущербный и неэффективный интерфейс между СУБД и прикладной программой (независимо от того, написан ли он в объектно-ориентированном стиле или нет). Тем не менее, SQL в настоящее время является единственным общепринятым распространенным языком баз данных на рынке. Использование специфичных для поставщика языков запросов считается плохой практикой, если ее можно избежать. Были предложены другие языки базы данных, такие как Business System 12 и Tutorial D; но ни один из них не был широко принят поставщиками СУБД.

В текущих версиях основных «объектно-реляционных» СУБД, таких как Oracle и Microsoft SQL Server, вышеупомянутый пункт может не быть проблемой. При помощи этих инструментов функциональность некоторой базы данных может быть произвольно расширена с помощью хранимого кода (функций и процедур), написанного на современном языке ОО (Java для Oracle и язык Microsoft .NET для SQL Server), и эти функции могут быть вызваны в свою очередь, в выражениях SQL прозрачным образом: пользователь не знает и не заботится о том, чтобы эти функции/процедуры изначально были частью ядра базы данных. Современные парадигмы разработки программного обеспечения полностью поддерживаются: таким образом, можно создать набор библиотечных подпрограмм, которые можно повторно использовать в нескольких схемах базы данных.

Эти поставщики решили поддержать интеграцию ОО-языка на стороне СУБД, поскольку поняли, что, несмотря на попытки комитета ISO SQL-99 добавить процедурные конструкции в SQL, SQL никогда не будет иметь богатого набора библиотек и структур данных, которые программисты современных приложений посчитают само собой разумеющимся, и разумно использовать их как можно более непосредственно, вместо того, чтобы пытаться расширить основной язык SQL. Следовательно, разница между «прикладным программированием» и «администрированием базы данных» теперь размыта: для надежной реализации таких функций, как ограничения и триггеры, часто может потребоваться человек с двойными

навыками программирования DBA/OO или партнерство между людьми, которые объединяют эти навыки. Этот факт также имеет отношение к проблеме «разделения ответственности», чтот представлено ниже.

Некоторые, однако, отметили бы, что это утверждение является спорным из-за того факта, что: (1) СУБД никогда не предназначались для облегчения моделирования объектов, и (2) SQL обычно следует рассматривать только как «ущербный» интерфейс или «неэффективный» язык, когда кто-то пытается найти решение, для которого RDBMS не были разработаны. SQL очень эффективен для выполнения того, для чего он предназначен, а именно для запроса, сортировки, фильтрации и хранения больших наборов данных. Некоторые дополнительно отметили бы, что включение функциональности языка ОО в программно-аппаратную часть сервиса (back-end) просто содействует плохой архитектурной практике, так как она допускает высокоуровневую прикладную логику на уровне данных, прямо противоположном RDBMS.

Здесь находится «каноническая» копия состояния. Модель базы данных обычно предполагает, что система управления базами данных является единственно допустимым хранилищем состояния, что касается предприятия; любые копии такого состояния, хранящиеся в прикладной программе, являются лишь временными копиями (которые могут быть устаревшими, если базовая запись в базе данных была впоследствии изменена транзакцией). Многие объектно-ориентированные программисты предпочитают рассматривать представления объектов в памяти как канонические данные, а базу данных — как механизм хранения и механизм сохранения.

Другим предметом спора является правильное разделение ответственности между программистами приложений и администраторами баз данных (DBA). Часто бывает так, что необходимые изменения в коде приложения (для реализации запрошенной новой функции или функциональности) требуют соответствующих изменений в определении базы данных; в большинстве организаций за определение базы данных отвечает администратор базы данных. В связи с необходимостью круглосуточного обслуживания системы производственной базы данных многие администраторы баз данных не хотят вносить изменения в схемы базы данных, которые они считают необоснованными или излишними, а в некоторых случаях и вовсе отказываются это делать. Использование эволюционных баз данных, допускающих непрерывное изменение структуры данных (за исключением работающих систем) может несколько помочь; но когда вновь разработанное приложение будет запущено, администратор БД должен будет одобрить любые изменения. Некоторые программисты рассматривают это как непримиримость; однако администратор базы данных часто несет ответственность, если какие-либо изменения в определении базы данных приводят к проблемам с обслуживанием в работающей системе — в результате многие администраторы баз данных предпочитают держать изменения проекта в коде приложений, где дефекты проекта с гораздо меньшей вероятностью могут иметь катастрофические последствия.

Однако в организациях с нефункциональными отношениями между администраторами баз данных и разработчиками вышеуказанная проблема не должна возникать, поскольку решение изменять схему базы данных или нет будет зависеть только от потребностей бизнеса: новое требование сохранять дополнительные данные или, например, повысить производительности критически важного приложения может вызвать изменение схемы.

4 Философские различия

Основные философские различия между ОО и реляционными моделями можно обобщить следующим образом:

Декларативные и императивные интерфейсы. Реляционное мышление использует данные в качестве интерфейсов, а не поведение в качестве интерфейсов. Таким образом, он имеет декларативный уклон в философии разработки в отличие от поведенческого уклона

ОО. (Некоторые сторонники реляции предлагают использовать триггеры, хранимые процедуры и т. д. для обеспечения сложного поведения, но это не является общей точкой зрения.)

Пределы схемы — Объекты не должны следовать «родительской схеме», согласно которой объект имеет атрибуты или средства доступа, в то время как строки таблицы должны следовать схеме объекта. Данная строка должна принадлежать одному и только одному объекту. Самая близкая вещь в ОО — это наследование, но обычно оно имеет форму дерева и является необязательным. Динамические системы баз данных, которые допускают специальные столбцы, могут ослабить привязку схемы, но такие системы в настоящее время либо редки, либо их классификация как «реляционная» под вопросом.

Правила доступа. В реляционных базах данных к атрибутам обращаются и изменяют их через предопределенные реляционные операторы, в то время как ОО позволяет каждому классу создавать свой собственный интерфейс и методы изменения состояния. Точка зрения ОО под названием «самообслуживаемое существительное» дает независимость каждому объекту, что не позволяет реляционная модель. Это дебаты «стандарты против местной свободы». ОО имеет тенденцию утверждать, что реляционные стандарты ограничивают выразительность, в то время как сторонники реляций предполагают, что соблюдение правила допускает более абстрактные математические рассуждения, целостность и согласованность дизайна.

Отношения между существительными и глаголами — ОО поощряет тесную связь между глаголами (действиями) и существительными (сущностями), с которыми работают операции. Возникающая в результате тесно связанная сущность, содержащая как существительные, так и глаголы, обычно называется классом или, в ОО-анализе, концепцией. Реляционные конструкции обычно не предполагают, что в таких тесных ассоциациях есть что-то естественное или логичное (за исключением реляционных операторов).

Идентичность объекта. Объекты (кроме неизменяемых) обычно считаются уникальными; два объекта, которые оказались в одном и том же состоянии в одно и то же время, не считаются идентичными. Отношения, с другой стороны, не имеют внутренней концепции такого рода идентичности. Тем не менее, обычной практикой является изготовление «идентификаторов» для записей в базе данных посредством формирования глобально уникальных потенциальных ключей[5]; хотя многие считают это плохой практикой для любой записи базы данных, которая не имеет соответствия один-к-одному с сущностью реального мира. (Отношения, как и объекты, могут использовать доменные ключи[6], если они существуют во внешнем мире для целей идентификации). Реляционные системы на практике стремятся и поддерживают «постоянные» и проверяемые методы идентификации, тогда как методы идентификации объектов имеют тенденцию быть временными или ситуативными.

Нормализация. Реляционные методы нормализации часто игнорируются проектами ОО. Тем не менее, это может быть просто плохая привычка вместо естественной функции ОО. Альтернативное представление состоит в том, что совокупность объектов, связанных определенными указателями, эквивалентна сетевой базе данных; которая в свою очередь может рассматриваться как чрезвычайно денормализованная реляционная база данных.

Наследование схемы. Большинство реляционных баз данных не поддерживают наследование схемы. Хотя такую функцию теоретически можно добавить, чтобы уменьшить конфликт с ООП, сторонники реляционных отношений с меньшей вероятностью будут верить в полезность иерархических таксономий и подтипов, поскольку они склонны рассматривать таксономии на основе множеств или системы классификации как более мощные и гибкие,

5 Потенциальный ключ (англ. candidate key) — в реляционной модели данных — подмножество атрибутов отношения, удовлетворяющее требованиям уникальности и минимальности (несократимости).

6 A natural key (also known as business key) is a type of unique key, found in relational model database design, that is formed of attributes that already exist in the real world. It is used in business-related columns. In other words, a natural key is a candidate key that has a logical relationship to the attributes within that row. A natural key is sometimes called domain key.

чем деревья. Сторонники ОО указывают, что модели наследования/подтипирования не обязательно должны ограничиваться деревьями (хотя это является ограничением во многих популярных языках ОО, таких как Java), но не-древовидные ОО-решения рассматриваются как более трудные для формулировки, чем вариации на основе множеств. методы управления по теме предпочтительнее реляционных. По крайней мере, они отличаются от методов, обычно используемых в реляционной алгебре.

Структура против поведения. ОО в первую очередь сосредотачивается на том, чтобы структура программы была разумной (поддерживаемой, понятной, расширяемой, многократно используемой, безопасной), тогда как реляционные системы фокусируются на том, какое поведение имеет конечная система времени выполнения (эффективность, адаптивность). отказоустойчивость, живучесть, логическая целостность и т. д.). Объектно-ориентированные методы обычно предполагают, что основным пользователем объектно-ориентированного кода и его интерфейсов являются разработчики приложений. В реляционных системах мнение конечных пользователей о поведении системы иногда считается более важным. Однако реляционные запросы и «представления» являются общими методами представления информации в конфигурациях, специфичных для приложения или задачи. Кроме того, реляционная структура не запрещает создание локальных или специфичных для приложения структур или таблиц, хотя многие распространенные инструменты разработки напрямую не предоставляют такую возможность, предполагая, что вместо них будут использоваться объекты. Это затрудняет определение того, является ли заявленная реляционная перспектива, не относящаяся к разработчикам, неотъемлемой частью реляционных или просто продуктом текущей практики и предположений реализации инструмента.

Отношения между множеством и графом. Отношения между различными элементами (объектами или записями), как правило, обрабатываются по-разному в разных парадигмах. Реляционные отношения обычно основаны на идиомах, взятых из теории множеств, в то время как объектные отношения склоняются к идиомам, взятым из теории графов (включая деревья). Хотя каждый из них может представлять ту же информацию, что и другой, подходы, которые они предоставляют для доступа к информации и управления ею, различаются.

В результате объектно-реляционной несогласованности сторонники обеих сторон дебатов часто утверждают, что от другой технологии следует отказаться или уменьшить ее объем [6]. Некоторые сторонники баз данных считают традиционные «процедурные» языки более совместимыми с СУРБД, чем многие ОО-языки; или предполагают, что следовало бы менее использовать стиль ОО. (В частности, утверждается, что долгоживущие доменные объекты в коде приложения не должны существовать; любые такие объекты, которые существуют, должны создаваться при выполнении запроса и уничтожаться при завершении транзакции или задачи). И наоборот, некоторые сторонники ОО утверждают, что необходимо разработать и использовать более устойчивые к ОО механизмы персистентности, такие как СУБД, и что стадию реляционных технологий следовало бы завершить. Многие (если не большинство) программисты и администраторы баз данных не придерживаются ни одной из этих точек зрения; и рассматривать объектно-реляционную несогласованность как простой факт жизни, с которым приходится иметь дело информационной технологии.

Также утверждается, что O/R отображение в некоторых ситуациях окупается, но, возможно, это переоценивается: оно имеет преимущества, помимо недостатков. Скептики отмечают, что перед его использованием стоит тщательно обдумать, так как в некоторых случаях это мало что даст [7].

5 Смотрите также

Объектно-реляционное отображение
Объектно-реляционная СУБД

6 References

1. C. J. Date, Relational Database Writings

2. Date, Christopher ‘Chris’ J; Pascal, Fabian (2012-08-12) [2005], "Type vs. Domain and Class", Database debunkings (World Wide Web log), Google, retrieved 12 September 2012.

3. --- (2006), "4. On the notion of logical difference", Date on Database: writings 2000–2006, The expert’s voice in database; Relational database select writings, USA: Apress, p. 39, ISBN 978-1-59059-746-0, "Class seems to be indistinguishable from type, as that term is classically understood".

4. --- (2004), "26. Object/Relational databases", An introduction to database systems (8th ed.), Pearson Addison Wesley, p. 859, ISBN 978-0-321-19784-9, "...any such rapprochement should be firmly based on the relational model".

5. Date, Christopher ‘Chris’ J; Darwen, Hugh, "2. Objects and Relations", The Third Manifesto, "The first great blunder"

6. Neward, Ted (2006-06-26). "The Vietnam of Computer Science". Interoperability Happens. Retrieved 2010-06-02.

7. Johnson, Rod (2002). J2EE Design and Development. Wrox Press. p. 256.

7 Внешние ссылки

The Object-Relational Impedance Mismatch[7] – Agile Data Essay

The Vietnam of Computer Science[8] – Examples of mismatch problems

7 http://www.agiledata.org/essays/impedanceMismatch.html

8 http://blogs.tedneward.com/post/the-vietnam-of-computer-science