Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

ВЫПУСКНАЯ РАБОТА БАКАЛАВРА

Тема: **Автогенерирование описания модели данных для интеграции системы бизнес-аналитики**

Студент гр. 43501/3 А.Р. Раскин

Санкт-Петербург – 2017

Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Работа допущена к защите
зав. кафедрой

_____ В.М. Ицыксон

«____» _____ 2017 г.

ВЫПУСКНАЯ РАБОТА БАКАЛАВРА

Тема: **Автогенерирование описания модели данных для интеграции системы бизнес-аналитики**

Направление: 09.03.01 – Информатика и вычислительная техника

Выполнил студент гр. 43501/3 _____ А.Р. Раскин

Научный руководитель,
ст.преподаватель _____ А.В. Зозуля

Санкт-Петербург – 2017

РЕФЕРАТ

Дипломная работа, 67 стр., 20 рис., 5 табл., 11 ист., 1 прил.

ИНФОРМАЦИОННЫЕ СИСТЕМЫ, БИЗНЕС-АНАЛИТИКА, PENTAHO, VTIGER, МЕТАДАННЫЕ, MONDRIAN, ИНТЕГРАЦИЯ

В выпускной работе проводится разработка плагина для системы бизнес-аналитики Pentaho BI с целью интеграции CRM-системы Vtiger для расширения возможностей аналитики данных.

В работе проведён анализ рынка, обзор существующих систем и их сравнение. Был проведён анализ инструментов аналитики различных систем, в результате чего был выбран OLAP-движок.

Средством разработки является объектно-ориентированный язык программирования Java. Взаимодействие между CRM-системой и платформой Pentaho реализовано с помощью REST-интерфейса.

В результате работы получен полноценный модуль Pentaho с графическим интерфейсом, анализирующий метаданные системы и составляющий по ним структуру хранилища для анализа инструментами Pentaho BI Suit.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 5

1. Анализ существующих решений 7
1.1. Информационные системы 7
1.1.1. Виды информационных систем 7
1.1.2. Способы аналитики ИС 8
1.1.3. Инструменты аналитики в Vtiger 9
1.2. Платформы бизнес-аналитики 11
1.2.1. Виды бизнес-аналитики 11
1.2.2. Анализ данных в Pentaho BI 13
1.3. Интеграция BI и информационной системы 16
1.4. Итоги 17

2. Постановка задачи 18

3. Архитектура системы и выбор средств разработки . 19
3.1. OLAP схема 19
3.2. Соответствие схемы и хранилища CRM 20
3.3. Разработка плагина Pentaho 21
3.3.1. Разработка на языке Java 22
3.3.2. Разработка с помощью Sparkl 23
3.4. REST API сервис 24
3.5. Разбор XML конфигурации 25
3.6. Итоги 26

4. Проектирование и разработка системы 27
4.1. Функциональность 27
4.1.1. Получение конфигурации 27
4.1.2. Обработка XML-файла 28
4.2. Реализация 28
4.2.1. Структура компонентов и описание 28
4.2.2. REST-сервис 29
4.2.3. Создание Mondrian схемы 30
4.2.4. Графический интерфейс 35
4.3. Итоги 36

5. Тестирование системы 37
5.1. Тестирование внутренней реализации 37
5.2. Тестирование REST-сервиса 38
5.3. Тестирование веб-интерфейса 38
5.4. Итоги 39

6. Анализ разработанной системы 40

ЗАКЛЮЧЕНИЕ 44

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ . . . 45

ПРИЛОЖЕНИЕ А. 46
А.1. Общие сведения 47
А.2. Используемые технические средства 47
А.3. Входные данные 47

ПРИЛОЖЕНИЕ А. ЛИСТИНГИ 48

ВВЕДЕНИЕ

Почти все крупные организации сталкиваются с быстро меняющейся деловой средой и постоянными сложностями в связи с огромным количеством поступающих данных, требующих анализа и хранящихся в компаниях.

Проблема сложности и качества аналитики таких данных для успешного решения проблем в настоящее время приводит всё больше компаний к решению использовать специальные информационные системы и средства аналитики в них.

Одним из таких средств является CRM(Customer relationship management) - термин, характеризующий практики, стратегии и технологии, используемые компаниями для управления и анализа данных и взаимодействий с пользователем. Такая система позволяет сосредоточиться на релевантных данных о какой-либо части бизнес-процесса.

К сожалению, аналитические инструменты в большинстве информационных систем предоставляют скудный функционал и возможности аналитики данных. Во многих системах присутствует только табличный тип отчётов и столбиковый вид диаграм, чего во многих случаях не хватает для полноты анализа данных.

Проблемы, касающиеся мощности аналитики и прогнозирования являются очень актуальными. Для решения данной проблемы многие организации используют более мощные средства и платформы, например “бизнес-аналитику”. Business Intelligence (BI) - это подход, которые помогает улучшить наблюдаемость бизнес-процессов, упрощает анализ данных и создание отчётов.

Чаще всего, организациям приходится нанимать специалистов, занимающихся интеграцией информационных систем для использования систем бизнес-аналитики. В данной работе рассматривается возможность упрощённой интеграции CRM-системы Vtiger в платформу бизнес-аналитики Pentaho BI Suit, с помощью внедрения данных из системы Vtiger в платформу.

В процессе работы будет разработан модуль, предназначенный для внедрения данных из CRM в платформу Pentaho BI. Данный модуль позволит импортировать и использовать данные из хранилища CRM-системы для глубокого анализа и построения отчётов.

1. Анализ существующих решений

1.1. Информационные системы

Информационные системы - системы, которые включают в себя организацию и анализ информации о бизнесе с помощью внедрения технологий. Это смесь базовых идей менеджмента, операций и теории информационных систем с компьютерными технологиями и инженерным подходом для управления данными в организациях.

1.1.1. Виды информационных систем

Существует несколько видов информационных систем, которые наиболее часто используются в повседневном мире. Каждая из них построена вокруг определённого инструмента организации данных(хранилища), имеет набор инструментов для анализа, формирования отчётов, систему управления данными(Dashboard) и возможность интеграции со сторонними ресурсами.

1. **CRM-системы**. Customer Relationship Management (CRM) - системы для сбора и хранения всей информации, связанной со взаимодействиями с клиентами. Цель CRM - предоставить полный набор данных о клиентах, которые могут использоваться для увеличения продаж, удержания клиентов и повышения эффективности взаимоотношений с ними. Программное обеспечение CRM помогает организовать, автоматизировать и синхронизировать продажи, маркетинг и обслуживание клиентов. Примерами таких систем являются: Microsoft Dynamics CRM, Salesforce, Vtiger.

2. **ERP системы**. В отличии от CRM, ERP (Enterprise Resource Planning) системы ориентированы больше на сами бизнес-процессы. Как и CRM, ERP-система позволяет быстро распределять стандартизованную информацию по всем подразделениям . Все сотрудники компании вводят информацию в систему ERP, создавая моментальный снимок в масштабе всего предприятия в режиме реального времени. О любой проблеме автоматически оповещаются все связанные отделы. Это позволяет более широко видеть процессы, происходящие в организации и принимать

более точные решения. Таким образом, ERP система позволяет компаниям сосредоточиться на данных, а не на операциях. Примеры ERP-систем: Microsoft Dynamics ERP, Epicor, Adaxa Suite.

3. **BPM системы.** Business Process Management (BPM) - позволяют создавать и совершенствовать широкий спектр процессов, которые прямо или косвенно влияют на клиента. BPM системы автоматизируют процессы документооборота и работы с клиентами. Основой является круги непрерывного улучшения, описывающие ключевые шаги системы, включающие в себя наблюдение, проектирование, моделирование, выполнение и мониторинг. Иногда круг улучшается за счет оптимизации и реинжиниринга. BPM обычно не используется изолированно. Примеры ERP-систем: Apian, PromApp, Camunda.

В данной работе используется CRM-система Vtiger. [1]Это бесплатная система с открытым кодом и возможностью создания пользовательских модулей.

1.1.2. Способы аналитики ИС

Для всех трёх видов систем характерен большой минус - малые возможности в анализе собранных данных. В информационных системах возможны несколько способов анализа и визуального представления результатов в виде отчётов.

1. Работа мастеров - masters in management (MiM). Во всех рассмотренных выше системах есть набор инструментов для анализа данных. Это специальные программы, автоматизирующие создание отчётов, диаграмм, срезов данных за счёт графического интерфейса и простоты использования. Внутренне такие отчёты представляют из себя простые выборки из базы данных с помощью операторов Join и фильтрации по фиксированному набору элементов. Такие отчёты могут выполнять непосредственно менеджеры, использующие систему.

2. Специализированные отчёты. Мастера отчётов могут предоставить лишь возможность составления простейших отчётов. В случаях, когда необходимо углубиться в анализ данных и предоставить данные, требующие более широкого знания системы и

навыков написания на языке SQL, компании привлекают программистов-профессионалов. Профессионалы обычно используют SQL-язык для выборок из базы данных, фильтрации и создания частей, требующих сложных арифметических операций. Обычно такой способ - это трудоёмкий ручной процесс, требующий специальных навыков.

1.1.3. Инструменты аналитики в Vtiger

Vtiger CRM для анализа и интеллектуальной обработки данных предоставляет специальные модули: Reports, Dashboards и различные виджеты. [2]"Виджет - это небольшой графический элемент, выводимый на главную страницу системы Vtiger CRM, либо на страницу "Виджеты" для любых Модулей системы. "

Виджеты домашней страницы, как можно увидеть на рисунке 1.1, помогают визуализировать представление данных в системе. На одном экране можно поместить и метрики, и результаты анализа. Это помогает следить за состоянием процессов и быстро принимать решения.

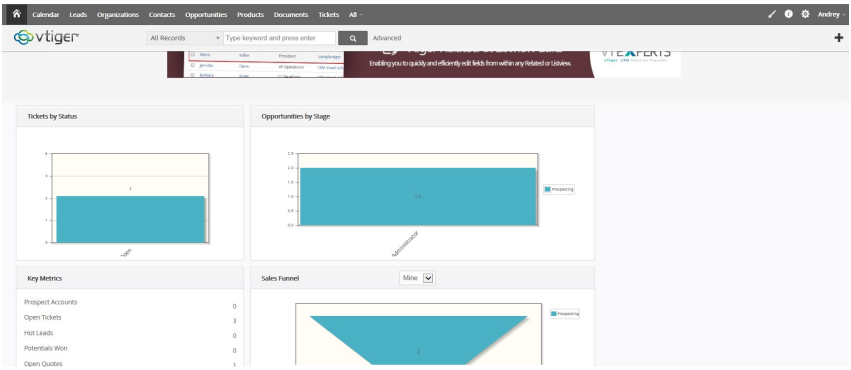


Рисунок 1.1. Пример виджета домашней страницы

Другим вариантом аналитики в представленной системе является модуль Отчёты. Модуль Отчёты суммирует, обрабатывает и подытоживает данные, хранящиеся в системе, с помощью отчётов. Vtiger предоставляет генератор и дизайнер отчётов:

1. Генератор отчётов предназначен для создания отчётов, которые будут отображаться на экране;

2. Дизайнер отчётов помогает выбрать данные, которые будут отображаться в отчёте.

Как видно на рисунке 1.2, стандартно Vtiger CRM предоставляет широкий выбор вариантов составления отчётов.

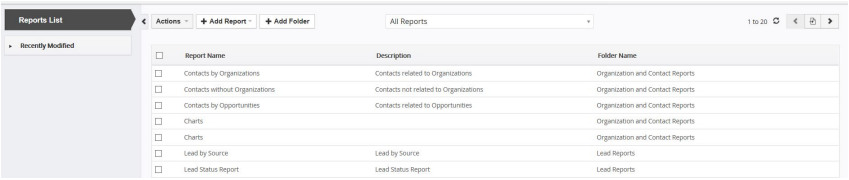


Рисунок 1.2. Список стандартных отчётов

Существует два вида отчётов: **диаграммы** и **детальные отчёты**. Детальные отчёты позволяют выводить информацию в табличной форме и выделять из потока данных только релевантную информацию. Пример такого отчёта на рисунке 1.3.

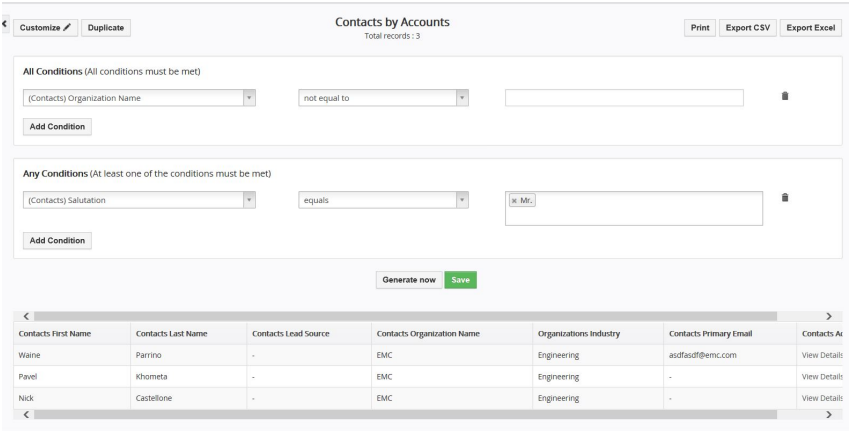


Рисунок 1.3. Пример детального отчёта

Диаграмма в свою очередь - это визуальное представление дан-

ных в виде круговой или столбиковой диаграммы, или линейного графика. Пример диаграммы на рисунке 1.4.

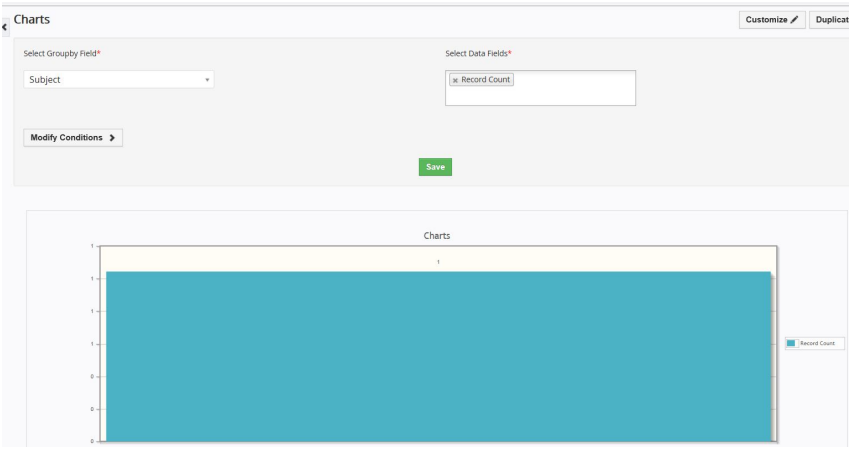


Рисунок 1.4. Пример отчёта-диаграммы

Каждый из двух видов отчётов требует дополнительной настройки и применения фильтров. Однако оба вида отчётов не всегда являются полными для аналитики.

1.2. Платформы бизнес-аналитики

Business Intelligence (BI) - это платформа или система, построенная из комбинаций различных компонентов. Эти компоненты могут формировать различные технологии и инструменты.

1.2.1. Виды бизнес-аналитики

[3] Платформа BI(рисунок 1.5) - это всё, что связано с данными. Во-первых, идентификация нужных данных и их источников; во-вторых, хранение данных и их анализ; в-третьих, визуализация данных как информации, используя создание отчетов, запросы к хранилищу и т.д.

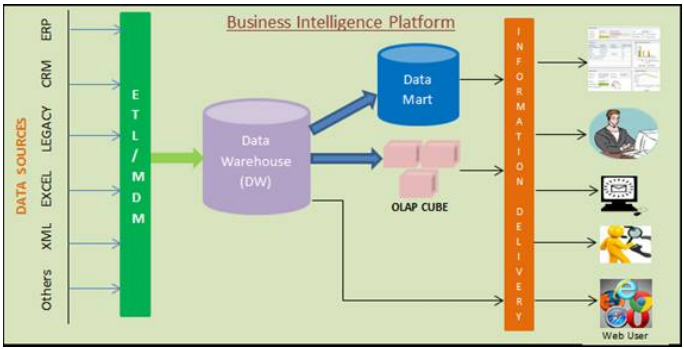


Рисунок 1.5. Структура BI платформы

Несмотря на то, что при внедрении CRM-систем в бизнес-процессы компании обычно наблюдается прирост в продажах 20%+([4]), такие результаты относятся к небольшим фирмам. В таких организациях обычно основными и единственными метриками являются метрики продаж, и использование CRM-систем даёт большой прирост к эффективности. Используя BI-решения, компании получают большие возможности для анализа данных и принятия своевременных решений, что влияет на доход и эффективность управления ресурсами компании. Инструменты аналитики BI можно разделить на три группы(рисунок 1.6), каждая из которых реализует разные BI подходы и возможности:

1. Управляемый анализ и создание отчётов. В эту группу можно отнести отчёты, панели управления, таблицы. В таких инструментах метрики и данные установлены заранее, но фильтрация, визуализация, сравнение и анализ варьируются в зависимости от потребностей компании.

2. Самостоятельный анализ BI системы. Эта категория включает в себя специализированный анализ данных, например рекуррентный. В отличии от управляемого анализа, позволяет пользователям добавлять и редактировать данные без помощи программистов. Такие инструменты включают в себя: OLAP технологии, Ad Hoc анализ, Data mining.

3. Продвинутые способы аналитики. Включают в себя инструмен-

ты для прогнозирования, статистического моделирования и интеллектуального анализа данных(Data mining).

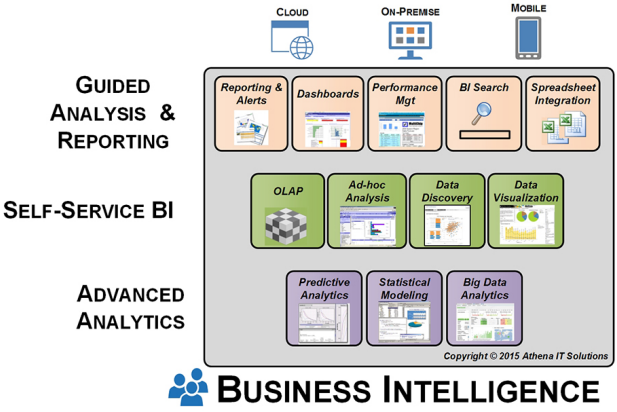


Рисунок 1.6. Инструменты BI аналитики

Примеров BI-платформ множество: IBM Cognos, Tableu, SAP BI, Oracle tools и др. Одним из лидеров также является Pentaho BI. Это полноценный набор инструментов, покрывающий все нужды большинства компаний. Pentaho BI является open-source проектом, но также имеет проприетарную платную версию. Сервер платформы написан на языке Java и предоставляет инфраструктуру для отображения отчётов через веб-сервисы. Главным преимуществом является расширяемая комплексная Data Platform, не имеющая пределов в бизнес аналитике и интеграции данных.

1.2.2. Анализ данных в Pentaho BI

В платформе Pentaho есть два варианта умного анализа данных:

1. Pentaho Reporting - встраиваемый движок создания отчётов. [5] Это набор инструментов с открытым кодом, предоставляющий возможность создания реляционных и аналитических отчётов.

   Благодаря Pentaho Reporting можно преобразовывать данные в значимую информацию, адаптированную для клиентов. Pentaho

Reporting может создавать HTML, Excel, PDF, Text или печатные отчеты. Для разработчиков также имеется экспорт отчётов в форматах CSV и XML для передачи другим системам.

Одним из преимуществ является отсутствие необходимости знаний о системе. Графический интерфейс отчётов прост в использовании, как видно из рисунка 1.7, и не требует специальных умений.

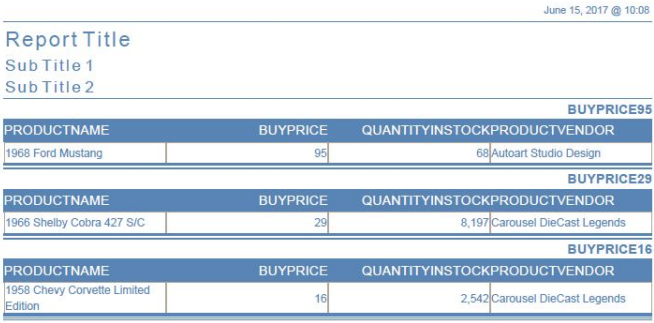


Рисунок 1.7. Отчёт в Pentaho Reporting

2. Mondrian OLAP Server. [6] OLAP - Online Analytical Processing - позволяет пользователям делать срезы нескольких измерений, чтобы изолировать и найти нужные части данных. Например, гиперкуб (многомерная структура данных) может содержать данные о продажах, структурированных по региону, продукту и другие данные. Для работы необходима специальная Mondrian схема - структура хранилища данных. OLAP-аналитика позволяет анализировать сразу огромные массивы данных, проводить аналитику с разных точек зрения, а также поддерживает принятие решений. В отличие от статических отчётов, OLAP-инструменты являются динамическими. Также пользователю не обязательно знать языки выборки, достаточно лишь минимальных знаний о системе. Пример отчёта сделанного с помощью OLAP-аналитики на рисунке 1.8.

Рисунок 1.8. Анализ с помощью инструмента Saiku

Сравнение двух вариантов отчётов представлено в таблице 1.2.2.

Таблица 1.1. Сравнение Reporting инструментов и Analysis инструментов

Reporting	Analysis
Назначение	
Преобразование данных в информацию	Элемент OLAP-куба
Показывает, что происходит в компании	Показывает, что происходит в компании и что с этим можно сделать
Наводить на вопросы о бизнес-процессе со стороны пользователя системы	Отвечать на вопросы с помощью интерпретации результатов аналитики
Подход	

Пассивный подход	Активный подход
Вопросы посылаются пользователям, которые затем извлекают содержательную уникальную информацию, касающуюся бизнес-процессов	Определённые данные извлекает пользователь для ответа на вопросы, связанные с бизнес-процессами компании

При реализации варианта с OLAP-аналитикой, получаемая в процессе схема может также использоваться в отчётах Pentaho Reporting. Несмотря на то, что OLAP не позволяет проводить анализ в режиме реального времени, преимущество использования языка SQL для запросов данных неоспоримо. Большие объёмы данных также легче обрабатывать с помощью Mondrian. Система отвечает на запросы к базе данных достаточно быстро даже при миллионах строк и автоматизирована в отличие от ручных отчётов.

В данной работе будет проведена интеграция CRM-системы Vtiger с платформой Pentaho для получения возможности использования OLAP-технологий для анализа данных.

1.3. Интеграция BI и информационной системы

И BI и CRM решения относятся к сбору и анализу данных, которые крайне важны для эффективного управления в компании. Отличия однако заключаются в том, что происходит с собранными данными. В случае BI, принятие решений полностью лежит на пользователе. CRM, напротив, позволяет пользователям участвовать в любом процессе анализа данных и автоматизирует их.

CRM-система, несмотря на свои минусы и основные этапы, может предложить действенные идеи при подключении BI. В большинстве организаций, связь приложения CRM с системой бизнес-аналитики, позволяет получить более эффективный анализ и сохранить лояльность клиентов. Многие пакеты CRM-систем сегодня поставляются с небольшим количеством инструментов для анализа. Необходимость приобретения дополнительных программных средств ведёт к дополнительным финансовым расходам, что является проблемой, особенно для небольших кампаний.

Специалисты-аналитики, использующие системы бизнес-аналитики, в свою очередь, не всегда имеют возможность изучить строение данных для анализа CRM-системы. Знание языка программирования,

на котором написано CRM-приложение, необходимо для добавления пользовательских модулей и построения нестандартных отчётов. Используя бизнес-аналитику можно сократить необходимость дополнительных знаний в CRM и сосредоточиться непосредственно на анализе данных.

BI является стратегическим решением, с помощью которого компании производят глубокую аналитику. BI обрабатывает большие объёмы информации, а CRM-система в результате интеграции выступает провайдером данных. Бизнес-аналитика позволяет обходить ограничения CRM-систем в способах и глубине анализа.

1.4. Итоги

В современном мире объемы анализируемых данных и требования организаций к вариантам их анализа продолжают расти. Многие компании начинают понимать эффективность использование CRM-систем в управлении и внедряют их. К сожалению, большинство информационных систем, в том числе и CRM-системы имеют довольно небольшой набор для анализа данных. Кроме того, создание отчётов в CRM требует привлечения специалиста и чаще всего трудоёмкой ручной работы. Расширить возможности, автоматизировать и упростить процесс анализа позволяет интеграция информационной системы с BI платформой. Различные BI платформы предоставляют инструменты для гибкого, многоцелевого анализа и прогнозирования. В данной работе произведена интеграция системы Vtiger CRM в платформу Pentaho BI Suite.

2. Постановка задачи

Необходимо разработать модуль для системы Pentaho BI Suite, для автоматического составления физической модели предоставленной системе базы данных в виде XML-файла с набором таблиц, полей в них и связей между таблицами.
Плагин должен:

1. Получать конфигурацию базы данных на основе вводимой пользователем ссылки на домен сервера информационной системы, логина и пароля к этой системе;

2. Преобразовывать конфигурацию в формат, принимаемый Mondrian OLAP сервером;

3. Публиковать сгенерированный файл с Mondrian схемой на сервер Pentaho BI, подключая его к указанному источнику данных.

Необходима простота использольвания для пользователя - возможность установки плагина без дополнительных приложений, а также интиутивно понятный интерфейс.

Конечная версия модуля должна представлять из себя "универсальный"конвертор реляционной структуры базы данных в Mondrian схему.

3. Архитектура системы и выбор средств разработки

Чтобы расширить возможности анализа CRM-системы с помощью интеграции платформы Pentaho необходимо задать платформе структуру базы данных, используемую в системе Vtiger. Для доступа к базе данных в Mondrian используются MDX-запросы. Для их исполнения на сервере Mondrian находится Mondrian-схема, представляющая логическую структуру базы данных.

3.1. OLAP схема

Почему Mondrian не читает данные напрямую из базы данных, а использует схему для хранения структуры? Ответ - Гибкость. В современном мире базовые концепции хранения данных разнообразны. В компаниях использует и различные базы данных, и различные форматы данных. Mondrian-схема - это логическая модель многомерной структуры данных. Она определяет один или несколько кубов в базе данных, каждый из которых содержит одно или несколько измерений и метрик. В Pentaho Mondrian схема соотносит концепт OLAP-технологии со структурой таблиц и строк в базе данных, а также определяет поля, которые будут рассчитываться в процессе выборок, но не хранятся в физической модели базы.

Формат схемы описывается с помощью языка XML. Каждая схема имеет набор кубов. Куб на рисунке 3.1 - это многомерная структу-

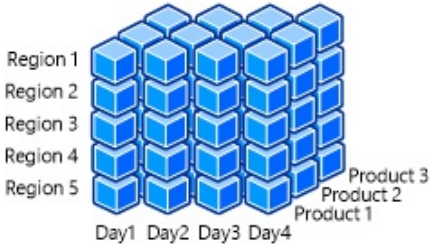


Рисунок 3.1. Пример OLAP-куба

ра, которую можно делить на части с целью отображения релевантной

информации по определённой теме. Каждый куб содержит несколько измерений(Dimension), обычно описывающих определённую бизнес-проблему. Измерения позволяют фильтровать, группировать и маркировать данные. Далее для более глубокого анализа существуют иерархии с возможностью более детальных уровней анализа. Структура куба представлена в листинге 3.4.

```
 Dimension (shared dimension)
  Hierarchy
    Level
      Property
Cube
  Dimension Usage
  Dimension
  Measure
  Calculated Measures
```

Также в каждом кубе могут быть метрики - численные значения, которые пользователи делят, агрегируют и анализируют.

3.2. Соответствие схемы и хранилища CRM

Для того, чтобы автоматически составить описанную выше схему, необходимо проанализировать и соотнести структуру CRM-системы со схемой базы данных. Рассмотрим, что представляет из себя структура системы Vtiger. Данная CRM-система состоит из модулей, пример которого указан в листинге 3.2.

```
<module>
  <blocks>
    <block>
      <label>LBL_ASSET_INFORMATION</label>
      <sequence>1</sequence>
      <fields>
        <field>
          <fieldname>asset_no</fieldname>
          <uitype>4</uitype>
          <columnname>asset_no</columnname>
          <tablename>vtiger_assets</tablename>
          <columntype>varchar(30)</columntype>
          ...
        </field>
  ...
  </blocks>
</module>
```

Каждый модуль - это бизнес-процесс. Он состоит из набора полей, каждое из которых представляет определённый атрибут. Среди атрибутов находятся тип поля и его имя, метка, например, для локализа-

ции, указание и наличие соединённых с ней таблиц. В соотношении полей с физической моделью, поля являются колонками базы данных. Таким образом, каждый модуль - это куб, имеющий связи с несколькими другими кубами. Для того, чтобы анализировать несколько кубов, в Mondrian имеется структура виртуального куба. Это несколько совмещённых кубов, имеющих перекрёстные связи.

3.3. Разработка плагина Pentaho

После разработки Mondrian схемы, следующим шагом является загрузка её на сервер. Это возможно несколькими способами. Самый простой способ - загрузка вручную. Платформа Pentaho оборудована инструментом загрузки файлов. В графическом интерфейсе сервера пользователь может управлять любыми источниками данных. Пример загрузки схемы через консоль можно увидеть на рисунке 3.2.

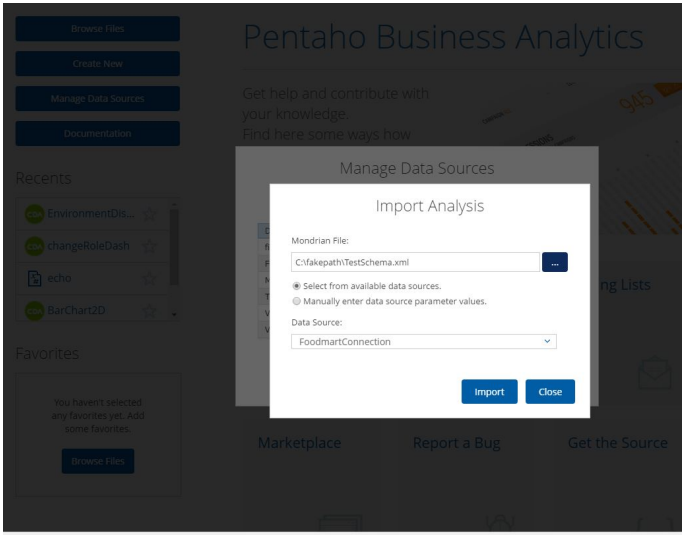


Рисунок 3.2. Загрузка схемы с помощью консоли

Другой способ это Kettle трасформации. Kettle или Pentaho Data Integration - это компонент платформы, отвечающий за основные процессы управления данными ETL - Extract, Transform, Load — «извле-

чение, преобразование, загрузка». С помощью таких операций можно импортировать и экспортировать данные на сервер Pentaho.

Однако, Kettle требует изучения. Альтернативным способом является использование Pentaho API[7]. В нём имеется набор точек входа, позволяющих манипулировать источниками данных и файлами, находищимися на сервере. В работе помещение источника описания на сервер реализовано с помощью таких URL. Пример можно увидеть в листинге 3.1.

Листинг 3.1. "Команда добавления нового источника данных Pentaho API"

```
1 PUT pentaho/plugin/data-access/api/datasource/analysis/catalog/
      SampleSchema
```

Pentaho BI - платформа с открытым кодом, написанная на языке Java. Плагины обладают следующими функциями:

1. Обслуживание другого приложения;
2. Визуализация данных в формате, описанном пользователем;
3. Регистрация новых файлов: схем, отчётов, параметров для сервера;
4. Интеграция сервера Pentaho со сторонними источниками;
5. Улучшение консоли пользователя Pentaho User Console.

Для интеграции Pentaho BI с Vtiger CRM необходим REST-сервис, обозначенный в пункте 1. Также следует разработать способ взаимодействия с пользователем, а именно, веб интерфейс.

Для разработки интерфейса предоставлено две воможности: разработка на языке платформы - язык Java и разработка с помощью Pentaho Plugin Builder(SPARKL).

3.3.1. Разработка на языке Java

С помощью языка Java платформа Pentaho предлагает разработку плагинов на основе платформы Spring и JAX-RS. Эти инструменты дают возможность создать REST-приложение, основанное на аннотациях и XML-конфигурации зависимостей классов. Требуются навыки программирования на языке Java, понимание технологии Модель-Представление-Контроллер. Такой способ разработки освобождает программиста от необходимости получения навыков в CTools, DI

и предоставляет более широкий выбор возможностей для разработки, например, использования более современных програмнных платформ и технологий. Java плагин обычно представляет из себя REST-сервис, запускающийся при старте сервера Pentaho и работающий на том же домене. Такой сервис может быть доступен извне с помощью заданного URL-ссылки при наличии данных аутентификации. В момент обращения к точке входа плагина сервер принимает запрос, идентифицирует пользователя и сам обращается к сервису.

3.3.2. Разработка с помощью Sparkl

Sparkl сам является плагином для BI сервера и позволяет создавать плагины и приложения для Pentaho. Приложение, созданное с

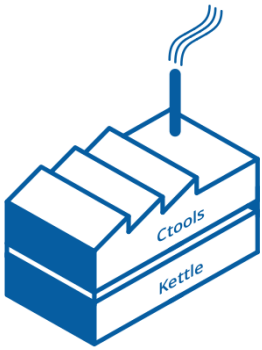


Рисунок 3.3. Структура Sparkl

помощью Sparkl, - это набор досок(dashboard), разработанных с помощью инструментов CDE и Pentaho Data Integration(или Kettle) работ и трансформаций с определёнными характеристиками. Как видно на рисунке 3.3, два слоя взаимодействуют друг с другом, предлагая широкий выбор возможностей для создания плагинов. Достоинством такого способа является прямое взаимодействие с данными и файлами сервера, что обеспечивает высокую скорость работы при обработке больших объёмов данных. Недостатки Sparkl:

1. Требование знания созданий трансформацтй Kettle;
2. Проблемы с выполнением Java кода из Kettle;

3. Сложность создания и редактирования панелей интерфейсов.

3.4. REST API сервис

[8] REST (сокр. от англ. Representational State Transfer — «передача состояния представления») — архитектурный стиль взаимодействия компонентов распределённого приложения в сети.

REST приложения используют HTTP запросы:

1. GET - возвращает указанный ресурс или коллекцию;
2. POST - создаёт ресурс на сервере. Также используется во всех остальных случаях;
3. PUT - обновляет указанный ресурс или коллекцию. Также используется в качестве аналога POST для создания;
4. DELETE - удаляет указанный ресурс.

[9]Pentaho поддерживает Rest Api и предлагает его использовать для создания плагинов для сервера(рисунок 3.4).

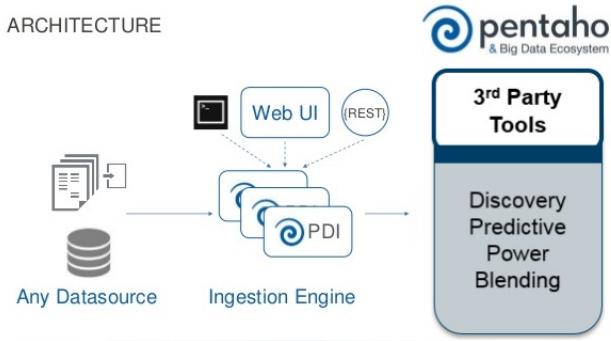


Рисунок 3.4. Схема платформы Pentaho Open BI

Pentaho API использует модель REST и технологию точек входа для доступа к данным. Точки входа позволяют легко соотнести URI и нужный HTTP метод. Иными словами, - это URL-ссылка, которую другая программа(приложение-клиент) будет использовать для общения с REST-сервисом.

Все точки входа - обычный набор данных. В качестве модели могут выступать различные медиа("MIME") типы в зависимости от назначения от точек входа. Данные описываются с помощью XML схемы, которая однозначно определяет XML отображения данных и помогает описать другие форматы, такие как JSON.

В работе в качестве способа анализа данных используется OLAP-сервер. Структура базы данных в таком сервере описывается с помощью формата XML, обладающего рядом достоинств:

1. Пространство имён позвляет обмениваться структурами;

2. Наличие стандартизированных способов выражения структуры документа: XML схема, DTD;

3. Использование стандартов синтаксического анализа: DOM, SAX и другие.

3.5. Разбор XML конфигурации

Перед тем, как данные из XML становятся доступными, их необходимо загрузить в объекты. Для этих целей есть несколько видов инструментов анализаторов:

1. DOM - определяет древовидную структуру, содержащую элементы документа. Другими словами, создаёт объектно-ориентированный граф - представление документа для навигации;

2. SAX - спецификация, определяющая основанный на событиях анализ, когда по мере сканирования документа вызываются функции на основании определённых найденных частей данных;

3. JAXB - API, реализующее как привязку XML к Java объектам, так и наоборот. Является наиболее простым в реализации.

Так как для составления конфигурации базы данных необходимо иметь полную структуру, чтобы ссылаться на существующие таблицы, была выбрана технология DOM. Для составления же конечной Mondrian схемы будет использоваться JAXB, как наиболее гибкий и простой в использовании.

3.6. Итоги

Разработка на языке Java проще и не требует дополнительных инструментов и знаний. В целях минимизации требуемых к установке средств платформы Pentaho для работы разрабатываемого плагина также не подходит использование SPARKL, для которого необходимо наличие CDE плагинов.

На рисунке 3.5 представлена схема архитектуры разрабатываемого плагина. Через веб-интерфейс пользователь вызывает функции получения конфигурации из CRM-системы, с помощью разработанного REST-сервиса на сервере Pentaho выполняется преобразование в Mondrian схему, и через API Pentaho загружается на сервер, подключая схему к указанному источнику данных(базе данных).

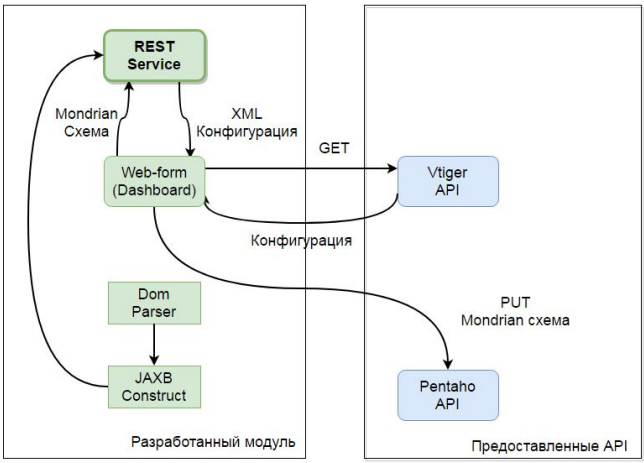


Рисунок 3.5. Архитектура разрабатываемого приложения

4. Проектирование и разработка системы

Разработка плагина производилась на языке Java, с использованием REST API функций в качестве точек входа. Для реализации веб-интерфейса использовался стандартный HTML 5 с Javascript.

4.1. Функциональность

4.1.1. Получение конфигурации

Для получения конфигурации Vtiger используется REST команда GET, обращающаяся к серверу Vtiger. Конфигурация предоставляется в формате XML и содержит описание модулей, их полей, связь между модулями и связи между таблицами в модулях. Окно ввода содержит поля:

1. VTiger URL - ссылка на расположение CRM;
2. Username - имя пользователя для доступа к CRM;
3. Password - пароль для доступа к CRM;
4. Datasource - источник данных(база данных), к которому будет относится созданная схема.

В случае удачного подключения и обработки конфигурации, пользователь видит оповещение в браузере с кодом ответа сервиса. В случае неудачного подключения, пользователю выводится соответствующее сообщение об ошибке. Такими ошибками могут быть:

1. Неверный логин или пароль;
2. Недоступность сервера;
3. Неудачная попытка загрузки новой схемы;
4. Ошибка при синтаксического анализа конфигурации.

4.1.2. Обработка XML-файла

Полученная от сервера Vtiger конфигурация приходит в формате XML. Для создания схемы необходимо сначала разобрать документ на классы, соответствующие элементам XML. Затем, имея полную коллекцию модулей, необходимо установить между ними связи, как непосредственно между таблицами в одном модуле, так и между самими модулями. Затем, с помощью маршалинга Java-объектов в XML-схему получить необходимую Mondrian-схему. Для того чтобы обработать файл необходимо:

1. Загрузить дерево документа в память;
2. Взять каждый элемент дерева;
3. Создать на основе элемента соответствующий класс.

4.2. Реализация

Главная часть приложения состоит из анализа и генерации Mondrian-схемы на основе конфигурации модулей информационной системы.

4.2.1. Структура компонентов и описание

Основной набор компонентов относится к разбору XML конфигурации, так как каждый элемент DOM дерева должен быть преобразован в класс. В таблице 4.1 представлены основные элементы и соответствующие им элементы куба.

Таблица 4.1. Таблица соответствия элементов конфигурации и OLAP куба

Название класса	Описание
PentahoMondrianService	Описание методов REST сервиса
DatabaseConfiguration	Описание конфигурации
Schema	Отображение Mondrian схемы
XMLParser	Класс-синтаксический анализатор, обрабатывающий XML
Классы-элементы XML	

FieldXML	Класс-описание поля Vtiger
ModuleXML	Класс-описание модуля Vtiger
VTigerXML	Класс-описание конфигурации VTiger
Классы-элементы Mondrian	
Cube	Класс-описание куба Mondrian, содержит коллекцию элементов-измерений
Dimension	Класс-описание измерения, содержит коллекцию иерархий
Hierarchy	Класс-описание иерархии, содержит коллекцию уровней
Level	Класс-описание куба Mondrian, содержит коллекцию элементов-измерений
VirtualCube	Класс-описание набора кубов Mondrian, содержит коллекцию измерений
TableAdapter	Класс-адаптер для размещения специального тега таблицы

4.2.2. REST-сервис

В языке Java определена поддержка REST через Java Specification Request (JSR) 311. Эта спецификация([10]) называется JAX-RS. Для реализации веб-сервиса в данной работе использовалась библиотека Jersey. Jersey предоставляет классы для имплементации веб-сервера в виде сервлета.

Конфигурация сервиса находится в отдельном файле - plugin.spring.xml. Базовый адрес сервиса:

Листинг 4.1. "Базовый адрес сервиса"

```
1 http://your_domain:port/display-name/url-pattern/path_from_rest_class
```

Сервер анализирует входящий HTTP запрос и выбирает соответствующий ему класс и метод для ответа. Выбор осуществляется на

основе аннотаций в классах и методах. Рассмотрим используемые в работе аннотации, представленные в листинге 4.2:

Таблица 4.2. Основные аннотации, используемые в работе

Аннотация	Описание
@PATH(your_path)	Устанавливает путь на конкатенацию адреса домена и адреса метода
@GET	Метод является HTTP GET запросом
@Produces(MediaType)	Какой тип данных будет являться ответом на запрос
@PathParam	Какие параметры будут внесены в запрос

В классе, представленном в листинге 4.2, метод будет доступен по адресу base/mondrian. Он будет являться GET запросом и возвращать XML строку.

Листинг 4.2. "GET запрос сервиса"

```
1 @GET
2 @Path("/mondrian")
3 @Produces(MediaType.APPLICATION_XML)
4 public String mondrian() throws ParserConfigurationException,
      SAXException, IOException, JDOMException {
5   ...
6 }
```

Данный класс-ресурс будет доступен по адресу localhost:8080/pentaho/plugin/api/mondrian при запросе типа GET, и пользователю будет возвращаться сформированная Mondrian схема в виде XML строки. Далее происходит вызов функции преобразования Java-объектов в XML файл - генерация Mondrian схемы через вызов методов программной платформы JAXB. Подробно о работе синтаксического анализатора и преобразования конфигурации рассказано далее.

4.2.3. Создание Mondrian схемы

Создание схемы делится на два этапа: синтаксический анализ конфигурации системы и создание Mondrian схемы на основе обработки полученных объектов.

Разбор XML конфигурации

Реализация анализатора находится в классе XMLParser. Используется библиотека JDOM2, предоставляющая стандартизированные методы для разбора документа в формате XML в древовидную структуру и обращения к элементам этой структуры. Для того, чтобы воспользоваться методами JDOM2, необходимо создать экземпляр SAXBuilder - строитель структуры документа. SAXBuilder использует стороннюю реализацию SAX(выбранный JAXP) или же можно указать, какой анализатор использовать вручную.

Рассмотрим часть работы синтаксического анализатора на примере получения элементов модулей. Каждый элемент содержит в себе также коллекцию элементов.

1. Получение конфигурации. Для этого создаётся подключение по HTTP URL ссылке, поданной в качестве аргумента к сервису.

   Листинг 4.3. "Пример открытия соединения"

```
1 //Создание ссылки из параметра
2 URL u = new URL(url);
3 //Открытие соединения
4 URLConnection uc = u.openConnection();
5 HttpURLConnection connection = (HttpURLConnection) uc;
```

2. Преобразование файла в документ JDOM. Необходимо создать объект анализатора, представленного JDOM2 и с помощью него преобразовать строку XML в древовидную структуру.

   Листинг 4.4. "Инициализация анализатора"

```
1 SAXBuilder builder = new SAXBuilder();
2 InputStream in = connection.getInputStream();
3 Document document = builder.build(in);
```

3. Анализ структуры. Для этого был реализован метод, представленный в листинге 4.5, анализирующий поля документа и связывающий между собой компоненты полученной конфигурации в схему. Обрабатывая в цикле все наборы элементов в структуре создаётся Mondrian схема.

   Листинг 4.5. "Разбор документа"

```
1 Element modulesElement = configuration.getChild(
      ROOT_MODULE_ELEMENT);
```

```
2  List<Element> modules = modulesElement.getChildren();
3
4  for (Element moduleElement : modules) {
5      System.out.println();
6      ModuleXML moduleXML = parseModuleXML(moduleElement);
7      vTigerxML.addModule(moduleXML);
8
9  }
```

Каждому тегу конфигурации необходимо привести в соответствие элемент Mondrian схемы. Карта соответствия можно увидеть в таблице 4.3

Таблица 4.3. Таблица соответствия элементов конфигурации и OLAP куба

Тег конфигурации	Описание тега	Элемент OLAP-куба
<module>	Тег, определяющий границы модуля CRM-системы	Cube - Куб, основной элемент схемы
<field>	Поле модуля	Dimension - Измерение, по которому может производится срез.
<tablename>	Таблица, в которой хранится поле	Table - таблица, в которой находится измерение
<columnname>	Имя колонки в таблице	Level column - колонка в иерархии измерения
<uitype>	Тип поля	Указание, является ли поле внешним ключом к другой таблице
<relatedlists>	Модули, связанные с выбранным	Virtual Cube - элемент, связывающий несколько кубов
<entityidcolumn>	Тег, указывающий на главный ключ сущности модуля	Hierarchy primary key - главный ключ таблицы измерения
<translated>	Перевод наименования сущности на русский	Caption - наименование, показываемое в графическом интерфейсе

Каждому элементу Mondrian схемы создана функция генерации дочерних элементов. Так, например, элемент “Куб” генерирует несколько элементов “Измерение” и так далее. Рассмотрим это на примере “Куба”.

Генерация схемы

Листинг 4.6. "Компонент генерации куба"

```
1  @XmlAccessorType(XmlAccessType.FIELD)
2  @Getter
3  @Setter
4  public class Cube {
5      @XmlAttribute
6      private String name;
7      @XmlAttribute
8      private String defaultMeasure;
9      @XmlAttribute
10     private String caption;
11     @XmlJavaTypeAdapter(TableAdapter.class)
12     private String table;
13     @XmlElement(name = "Dimension", type = Dimension.class)
14     private List<Dimension> dimension;
15
16     public Cube generate(ModuleXML module) {
17         for (FieldXML fieldXML : module.getFieldXMLList()) {
18             Dimension dimensionGen = new Dimension(fieldXML.
                   getFieldName());
19             dimensionGen = dimensionGen.generate(fieldXML);
20             dimensionGen.setForeignKey(fieldXML.getColumnName());
21             dimension.add(dimensionGen);
22         }
23         return this;
24     }
25 }
```

В строках 17-21 листинга 4.6 в цикле рассматриваются полученные модули. Для каждого поля модуля необходимо создать измерение. Каждое измерение - это срез многомерного куба, на основе которого будет происходить фильтрация данных. Каждый дочерний класс имеет аннотации элементов и атрибутов. Элементы - поля, которые станут тегами в конечной Mondrian схеме, а аттрибуты - это параметры этих тегов. В листинге 4.7 представлен пример сгенерированного куба.

Листинг 4.7. "Пример сгенерированного куба"

```
1  <Cube name="Assets">
2      <Table name="vtiger_assets"/>
3      <Dimension name="asset_no" foreignKey="asset_no" type="
           StandardDimension">
4          <table name="vtiger_assets"/>
5          <Hierarchy name="asset_no" visible="true" hasAll="true">
6              <Level name="asset_no" column="asset_no" visible="true"
                   uniqueMembers="false"/>
7          </Hierarchy>
8      <Dimension/>
9  <Cube/>
```

В строке 3 примера “Dimension” является тегом, а “name”, “foreignKey”, “type” - атрибутами.

Основной задачей также является возможность объединения нескольких кубов для перекрёстного анализа модулей. Для этой функции в Mondrian существуют виртуальные кубы [6].

4.2.4. Графический интерфейс

Проектирование графического дизайна не было главной задачей разработки. Главное в интерфейсе для плагина - интуитивность и скорость работы.

Для реализации было решено использовать технологии HTML и Javascript. На рисунке 4.1 показано главное окно, которое открывается пользователь при выборе владки плагина.

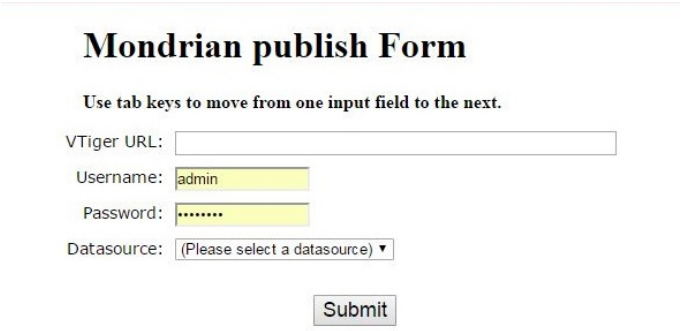


Рисунок 4.1. Окно интерфейса плагина

Страница выглядит как форма для заполнения с кнопкой подтверждения. За дизайн отвечает css код, находящийся в отдельном файле.

Важной частью приложения также является Javascript код, скрытый за HTML формой. Реализацию функций можно найти в приложении 4.1. В таблице указаны основные функции.

Таблица 4.4. Основные функции Javascript-части программы

Функция	Комментарий
getSources	Функция получения XML-конфигурации от Vtiger API
getMondrian	Функция обращения к REST-сервису, обрабатывающему конфигурацию
publish	Функция загрузки сгенерированной схемы на сервер Pentaho
getAnalysis	Функция получения конфигурации с сервера Pentaho по идентификатору

4.3. Итоги

Было разработано приложение-плагин для Pentaho BI server, в частности, для Pentaho User Console. Данное приложение реализует интеграцию CRM-системы в платформу Pentaho посредством анализа и преобразования XML-конфигурации базы данных системы в средство анализа бизнес-аналитики - Mondrian схему.

5. Тестирование системы

Тестирование было разделено на backend(внутренняя реализация веб-сервиса и анализатора) и frontend(в данном случае Javascript часть) части.

5.1. Тестирование внутренней реализации

Тесты инструментов разбора XML могут быть использованы для проверки работоспособности, надёжности и корректности работы анализатора. Такие тесты обычно содержат шаблон, который может быть отредактирован для использования с любым контекстом. В тестировании анализатора сравниваются две строки - идеальная и сгенерированная. Реализован простейший алгоритм:

1. Создана модель данных;
2. Сгенерирована XML строка в соответствии с моделью;
3. Производено сравнение строки с эталонной, указанной вручную.

В данной работе используется библиотека XmlUnit. Она позволяет сравнивать два XML файла, игнорировать лишние пробелы и перестановку элементов и другие, не влияющие на процесс анализа, неточности.

Для сравнения существует класс Diff, представленный в листинге 5.1:

Листинг 5.1. "Класс сравнения двух документов"

```
1  Diff diff = DiffBuilder
2      .compare(expectXml)
3      .withTest(marshaledXml)
4      //Ignore element order
5      .withNodeMatcher(new DefaultNodeMatcher(ElementSelectors.byName)
           )
6      .ignoreWhitespace()
7      .ignoreComments()
8      .checkForSimilar()
9      .build()
10 assert !diff.hasDifferences()
```

5.2. Тестирование REST-сервиса

Jersey, используемый в проекте для тестирования предоставляет минимальный веб-сервер для развёртывания пользователем. Для тестирования разработанного сервиса необходимо проверить работу точки входа для получения сгенерированный схемы. Так как в методе генерации схемы используется сторонний сервис, а именно, установление соединения с сервисом Vtiger, необходимо использовать заглушку (“Mock”).

Листинг 5.2. "Пример описания заглушки"

```
1 @Mock
2 private BackendService backendServiceMock;
```

Чтобы не оставлять сервер открытым после выполнения теста, можно использовать аннотацию @AfterClass, выполняющую часть кода после исполнения всех запланированных в классе тестов.

Листинг 5.3. "Использования аннотации AfterClass"

```
1 @AfterClass
2 public static void afterClass() throws Exception {
3     server.close();
4 }
```

Для верификации результата теста нужно сравнить код ответа от сервера с ожидаемым(200 - OK), указанным в спецификации [11]

Листинг 5.4. "Сравнение ответного кода с ожидаемым положительным"

```
1 assertEquals(Response.Status.OK.getStatusCode(), response.getStatus
      ());
```

Были протестированы варианты кодов ответа: позитивного(200), негативного(505) и случай недоступности сервиса(404).

5.3. Тестирование веб-интерфейса

Тестирование интерфейса плагина, в данном случае, представляет из себя тестирование Javascript кода, отвечающего за ввод данных в форму на странице плагина. Для тестирования использовалась библиотека Jasmine, предоставляющая возможность создавать “заглушки” для элементов:

Листинг 5.5. "Пример использования библиотеки Jasmine"

```
var submit;
beforeEach(){

  submit = jasmine.createSpy();
  spyOn(document 'getElementById').andReturn({submit:submit})
}
```

Тесты для интерфейса были включены в процесс сборки плагина и выполняются автоматически.

5.4. Итоги

При написании любого приложения тестирование является важной частью разработки. Разделённое на две части: FrontEnd и BackEnd тестирование помогает обнаружить ошибки как в серверной части, в данном случае, парсере документа, так и в части интерфейса, в данном случае являющимся Javascript кодом.

Тесты для серверной части помогают:

1. Проверить корректность сгенерированного XML файла по схеме или другому грамматическому источнику;

2. Обнаружить пустые или отсутствующие элементы, которые могут нарушить порядок работы синтаксического анализатора.

В процесс тестирования было создано 4 Unit-теста, а также несколько Javascript-тестов. Как показано на рисунке 5.1, все Unit-тесты закончились удачно.

Рисунок 5.1. Результаты тестирования

Данные полученные в тестировании свидетельствуют о корректной работе сервиса. При некоректной поданной схеме конвертация не происходит и пользователю возвращается корректная ошибка. При отсутствии доступа к серверу Vtiger CRM для получения конфигурации выводится соответствующая ошибка.

6. Анализ разработанной системы

В результате разработки была реализована возможность интеграции хранилища CRM-системы в платформу бизнес-аналитики.

Интеграция происходит с помощью веб-сервиса, находящегося на сервере Pentaho. С помощью приложения возможно преобразовать имеющуюся структуру базы данных CRM в многомерную структуру, необходимую для OLAP-аналитики.

В результате исследования было выяснено, что аналитические инструменты CRM Vtiger недостаточно мощны для комплексного анализа, а также не позволяют использовать несколько источников данных. Реализованное приложение даёт возможность построения диаграмм, аналитики и прогнозирования в Pentaho BI с помощью инструментов платформы.

Сравним отчёты, сформированные в Pentaho и созданные с помощью встроенных инструментов Vtiger. Создадим отчёт, отображающий прибыль по контрагентам. На графике необходимо отобразить сумму сделок у каждого контрагента.

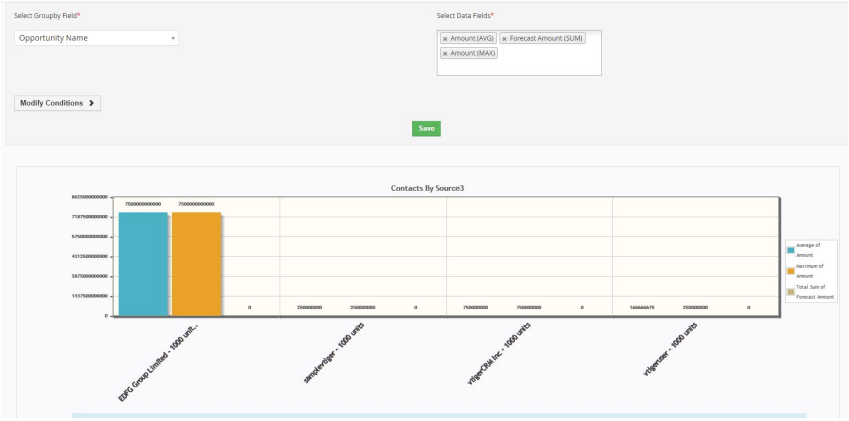


Рисунок 6.1. Пример отчёта в Vtiger модуле отчётов

Как видно на рисунке 6.1, в модуле отчётов Vtiger доступно только три параметра для одного графика. Для сравнения построим похожий отчёт в Pentaho, а конкретно, в инструменте Saiku. На рисунке

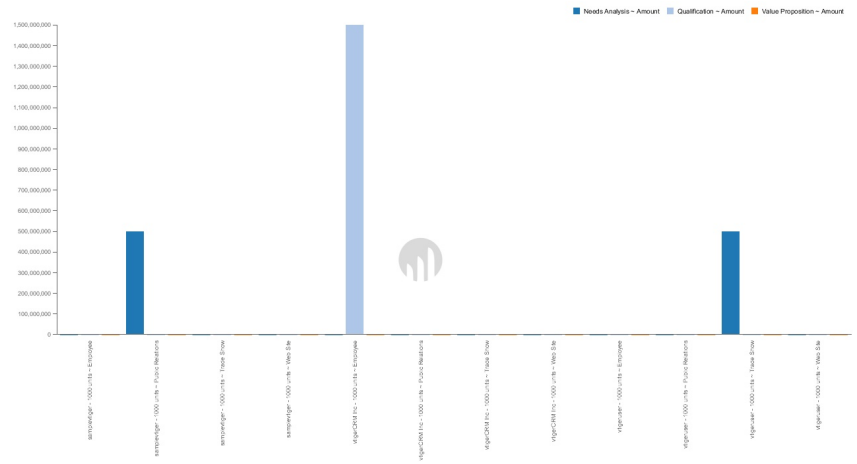
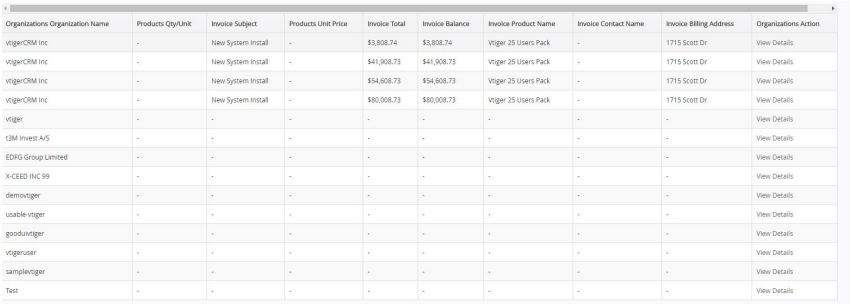


Рисунок 6.2. Пример отчёта в Saiku в системе Pentaho

6.2 представлен график в Pentaho BI. OLAP-аналитика позволяет добавлять любое количество параметров к отчёту, углубиться в какой-либо из пунктов в отчёте, предоставив более подробную статистику. Также, как видно на рисунке 6.3, в Pentaho можно легко переключаться между видами отчётов, получая при этом возможность просматривать нужную информацию с разных сторон.

Рисунок 6.3. Различные виды отчётов Pentaho

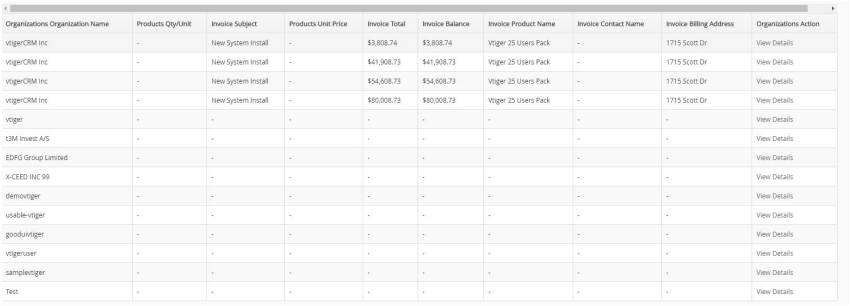
Попробуем создать более сложный отчёт. В Vtiger отобразим статистику продаж по адресам штатов и проданным товарам. На рисунке 6.4 представлен отчёт в виде таблицы.



Organizations Organization Name	Products Qty/Unit	Invoice Subject	Products Unit Price	Invoice Total	Invoice Balance	Invoice Product Name	Invoice Contact Name	Invoice Billing Address	Organizations Action
vtigerCRM Inc	-	New System Install	-	$3,808.74	$3,808.74	Vtiger 25 Users Pack	-	1715 Scott Dr	View Details
vtigerCRM Inc	-	New System Install	-	$41,908.73	$41,908.73	Vtiger 25 Users Pack	-	1715 Scott Dr	View Details
vtigerCRM Inc	-	New System Install	-	$54,608.73	$54,608.73	Vtiger 25 Users Pack	-	1715 Scott Dr	View Details
vtigerCRM Inc	-	New System Install	-	$80,008.73	$80,008.73	Vtiger 25 Users Pack	-	1715 Scott Dr	View Details
vtiger	-	-	-	-	-	-	-	-	View Details
t3M Invest A/S	-	-	-	-	-	-	-	-	View Details
EDFG Group Limited	-	-	-	-	-	-	-	-	View Details
X-CEED INC 99	-	-	-	-	-	-	-	-	View Details
demovtiger	-	-	-	-	-	-	-	-	View Details
usable-vtiger	-	-	-	-	-	-	-	-	View Details
gooduivtiger	-	-	-	-	-	-	-	-	View Details
vtigeruser	-	-	-	-	-	-	-	-	View Details
samplevtiger	-	-	-	-	-	-	-	-	View Details
Test	-	-	-	-	-	-	-	-	View Details

Рисунок 6.4. Статистика продаж в Vtiger

Для полноценного анализа специалистам часто приходится составлять несколько отчётов и использовать их одновременно, так как ограничение в два прикреплённых модуля к одному основному не позволяет строить подробные отчёты. С помощью объединения модулей в виртуальные кубы это ограничение легко обойти. На рисунке 6.5 представлен отчёт в Saiku инструменте, отображающий те же данные, что и в Vtiger.



Organizations Organization Name	Products Qty/Unit	Invoice Subject	Products Unit Price	Invoice Total	Invoice Balance	Invoice Product Name	Invoice Contact Name	Invoice Billing Address	Organizations Action
vtigerCRM Inc	-	New System Install	-	$3,808.74	$3,808.74	Vtiger 25 Users Pack	-	1715 Scott Dr	View Details
vtigerCRM Inc	-	New System Install	-	$41,908.73	$41,908.73	Vtiger 25 Users Pack	-	1715 Scott Dr	View Details
vtigerCRM Inc	-	New System Install	-	$54,608.73	$54,608.73	Vtiger 25 Users Pack	-	1715 Scott Dr	View Details
vtigerCRM Inc	-	New System Install	-	$80,008.73	$80,008.73	Vtiger 25 Users Pack	-	1715 Scott Dr	View Details
vtiger	-	-	-	-	-	-	-	-	View Details
t3M Invest A/S	-	-	-	-	-	-	-	-	View Details
EDFG Group Limited	-	-	-	-	-	-	-	-	View Details
X-CEED INC 99	-	-	-	-	-	-	-	-	View Details
demovtiger	-	-	-	-	-	-	-	-	View Details
usable-vtiger	-	-	-	-	-	-	-	-	View Details
gooduivtiger	-	-	-	-	-	-	-	-	View Details
vtigeruser	-	-	-	-	-	-	-	-	View Details
samplevtiger	-	-	-	-	-	-	-	-	View Details
Test	-	-	-	-	-	-	-	-	View Details

Рисунок 6.5. Подробный отчёт в Pentaho BI

Таким образом, можно сказать, что задача, поставленная в начале работы была выполнена. С помощью разработанного плагина ана-

литики могут составлять подробные отчёты без знания CRM-системы и языка запросов.

ЗАКЛЮЧЕНИЕ

В процессе написания работы, были исследованы информационные системы, хранящие и позволяющие анализировать информацию связанную с бизнес-процессами и менеджментом.

Исследование показало, что компаниям выгодно использовать связку информационная система - платформа бизнес-аналитики. Такая связка расширяет возможности анализа, предлагая мощность инструментов BI платформы в обработке данных из хранилища CRM.

В данной работе исследовалась возможность интеграции Vtiger CRM - системы с открытым кодом, являющейся одним из лидеров на рынке и BI платформы Pentaho BI Suite. Данная платформа имеет возможность анализа с помощью OLAP-кубов - многомерных таблиц.

Для интеграции был написан плагин для Pentaho, преобразующий конфигурацию CRM-системы в Mondrian схему. Разработанный плагин анализирует входящие в систему модули и на их основе составляет кубы, принимая за измерения данные, фигурирующие в модулях. В качестве факт-таблицы берётся главная таблица в модуле.

В процессе разработки возникли определённые трудности. Так, для правильного составления структуры хранилища данных CRM не хватало информации о связи некоторых таблиц. Для решения проблемы был разработан дополнительный тэг в конфигурации, прямо указывающий на колонки таблиц, через которые они имеют связь.

Разработанный плагин действительно расширяет возможности бизнес-аналитики по сравнению с встроенными в CRM-систему инструментами. Также исчезает необходимость в найме специалиста, разрабатывающего и поддерживающего создание ручных отчётов, например, с помощью SQL. При присоединении платформы появляются возможности использовать гибкие инструменты аналитики и прогнозирования, такие как Saiku, JPivot, FusionCharts и др.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Vtiger Documentation [Электронный ресурс], Vtiger. — URL: http://community.vtiger.com/help/ (дата обращения: 11.06.2017).
2. SalesPlatform Wiki [Электронный ресурс], SalesPlatform. — URL: https://goo.gl/EFSpny (дата обращения: 11.06.2017).
3. Understanding BI analytics tools and their benefits [Электронный ресурс], TechTarget. — URL: http://searchbusinessanalytics.techtarget.com/feature/Understanding-BI-analytics-tools-and-their-benefits (дата обращения: 11.06.2017).
4. CRM Statistics [Электронный ресурс], SmallBizCRM.com. — URL: http://www.smallbizcrm.com/crm-reading-lounge/crm-market-statistics/ (дата обращения: 11.06.2017).
5. Pentaho Reporting [Электронный ресурс], Rittan Holdings Ltd. — URL: http://community.pentaho.com/projects/reporting/ (дата обращения: 11.06.2017).
6. OLAP-engine Documentation [Электронный ресурс], Pentaho Corporation. — URL: http://mondrian.pentaho.com/documentation/schema.php (дата обращения: 11.06.2017).
7. API Documentation for BA Platform [Электронный ресурс], Pentaho Corporation. — URL: https://help.pentaho.com/Documentation/6.0/0R0/070/010 (дата обращения: 11.06.2017).
8. REST API Documentation [Электронный ресурс], WP REST API. — URL: http://v2.wp-api.org/ (дата обращения: 11.06.2017).
9. Pentaho Presentation [Электронный ресурс], By Ricardo Pires - Partner and BI Lead. — URL: https://www.slideshare.net/XpandIT/real-use-cases-pentaho-big-data-ecosystem (дата обращения: 11.06.2017).
10. Интернет-ресурс [Электронный ресурс], Oracle Corporation. — URL: https://jcp.org/en/jsr/overview (дата обращения: 11.06.2017).
11. Интернет-ресурс [Электронный ресурс], The Internet Society (1999). — URL: https://www.w3.org/Protocols/rfc2616/rfc2616.txt (дата обращения: 11.06.2017).

ПРИЛОЖЕНИЕ А

Автогенерирование описания модели данных для интеграции системы бизнес-аналитики

Описание программы.

1 лист

А.1. Общие сведения

Разработанное приложение-плагин реализует интеграцию CRM-системы и платформы бизнес-аналитики Pentaho BI. Это позволяет анализировать данные CRM-системы с помощью инструментов аналитики, выполнять прогнозирование и строить подробные отчёты.

А.2. Используемые технические средства

Объектно-ориентированный язык программирования Java, Платформа BI с открытым исходным кодом Pentaho BI Suit, CRM-система Vtiger.

А.3. Входные данные

Входными данными являются URL CRM-системы:

1. Адрес сервера;
2. Порт сервера.

А также данные пользователя, необходимые для аутентификации:

1. Логин пользователя;
2. Пароль пользователя.

ПРИЛОЖЕНИЕ А
ЛИСТИНГИ

Листинг А.1. XMLParser.java

```
1  package org.xerocry.mondrian.xml_elements;
2
3  import org.xerocry.mondrian.DatabaseConfiguration;
4  import org.jdom2.Document;
5  import org.jdom2.Element;
6  import org.jdom2.JDOMException;
7  import org.jdom2.input.SAXBuilder;
8  import org.xml.sax.SAXException;
9
10 import javax.xml.parsers.ParserConfigurationException;
11 import java.io.IOException;
12 import java.io.InputStream;
13 import java.net.HttpURLConnection;
14 import java.net.URL;
15 import java.net.URLConnection;
16 import java.util.HashMap;
17 import java.util.List;
18 import java.util.logging.Logger;
19
20 public class XMLParser {
21
22     private static Logger log = Logger.getLogger(
          DatabaseConfiguration.class.getName());
23
24     private static final String ROOT_CONFIGURATION_ELEMENT = "
          configurations";
25     private static final String ROOT_MODULE_ELEMENT = "modules";
26     private static final String ROOT_BLOCK_ELEMENT = "blocks";
27     private static final String ROOT_FIELD_ELEMENT = "fields";
28     private static final String ROOT_RELATED_MODULE_ELEMENT = "
          relatedmodules";
29     private static final String ROOT_RELATED_LIST_ELEMENT = "
          relatedlists";
30     private static final String ROOT_MODSTRINGS_ELEMENT = "
          modstrings";
31     private static final String DESCRIPTION_ELEMENT = "description";
32
33     private static final String CONFIGURATION_ELEMENT = "
          configuration";
34     private static final String TABLENAME_ELEMENT = "tablename";
35     private static final String FIELDNAME_ELEMENT = "fieldname";
36     private static final String TRANSLATED_ELEMENT = "original";
37     private static final String ORIGINAL_ELEMENT = "original";
38     private static final String COLUMNNAME_ELEMENT = "columnname";
39     private static final String COLUMTYPE_ELEMENT = "columntype";
40     private static final String UI_TYPE_ELEMENT = "uitype";
41     private static final String RELATED_MODULE_ELEMENT = "
          relatedmodule";
42     private static final String MODULE_NAME_ELEMENT = "label";
```

```
43      private static final String FIELD_LABEL_ELEMENT = "fieldlabel";
44      private static final String ROOT_ENTITY_ID_ELEMENT = "
            entityidentifier";
45      private static final String ENTITY_ID_ELEMENT = "entityidcolumn"
            ;
46
47      private static final String CRM_TABLE_FIELD = "vtiger_crmentity"
            ;
48      private static final String TRANSLATION_ELEMENT = "translation";
49
50      private VTigerXML vTigerxML = new VTigerXML();
51
52      private String url;
53
54      public XMLParser(String url) {
55          this.url = url;
56      }
57
58      /**
59       * Метод для получения конфигурации через HTTP запрос и формиров
             ания схемы
60       *
61       * @return Считанная в Java-объекты конфигурация
62       * @throws JDOMException
63       * @throws IOException
64       * @throws ParserConfigurationException
65       * @throws SAXException
66       */
67      public VTigerXML readXML() throws JDOMException, IOException,
            ParserConfigurationException, SAXException {
68          URL u = new URL(url);
69          log.info("Trying to connect to url " + url);
70          URLConnection uc = u.openConnection();
71          HttpURLConnection connection = (HttpURLConnection) uc;
72          SAXBuilder builder = new SAXBuilder();
73          InputStream in = connection.getInputStream();
74          log.info("Input stream get try : " + in.available());
75          Document document = builder.build(in);
76          in.close();
77          connection.disconnect();
78
79          Element rootConfiguration = document.getRootElement();
80          assert rootConfiguration.getName().equals(
                ROOT_CONFIGURATION_ELEMENT);
81          Element configuration = rootConfiguration.getChild(
                CONFIGURATION_ELEMENT);
82          assert configuration != null;
83
84          Element description = configuration.getChild(
                DESCRIPTION_ELEMENT);
85          String[] tmp = description.getValue().split(";");
86          HashMap<String, String> sourceInfo = new HashMap<>();
87          for (String var : tmp) {
88              String[] keyVal = var.split("=");
89              sourceInfo.put(keyVal[0].trim(), keyVal[1].trim());
90          }
91          vTigerxML.getSourceInfo().putAll(sourceInfo);
92          parseXML(configuration);
```

```
93
94          return vTigerxML;
95      }
96
97      private void parseXML(Element configuration) {
98          Element modulesElement = configuration.getChild(
                ROOT_MODULE_ELEMENT);
99          List<Element> modules = modulesElement.getChildren();
100
101         for (Element moduleElement : modules) {
102             System.out.println();
103             ModuleXML moduleXML = parseModuleXML(moduleElement);
104             vTigerxML.addModule(moduleXML);
105         }
106
107     }
108
109     /**
110      * Метод разбора каждого модуля
111      *
112      * @param module
113      * @return
114      */
115     private ModuleXML parseModuleXML(Element module) {
116         ModuleXML moduleXML = new ModuleXML();
117         List<Element> childNodes = module.getChildren();
118         Element p1 = module.getParentElement();
119         Element p2 = p1.getParentElement();
120         Element p3 = p2.getChild(TRANSLATION_ELEMENT);
121         Element mainTranslations = p3.getChild(
                ROOT_MODSTRINGS_ELEMENT);
122
123         /*
124         Получение названий для элементов
125          */
126         Element transNode = module.getChild(TRANSLATION_ELEMENT);
127         Element modstringsNode = transNode.getChild(
                ROOT_MODSTRINGS_ELEMENT);
128
129         /*
130         Итерация по элементам конфигурации
131          */
132         for (Element childNode : childNodes) {
133             if (childNode.getName().equals(MODULE_NAME_ELEMENT)) {
134                 String label;
135                 if ((label = findTranslation(childNode.getValue(),
                        mainTranslations)) != null) {
136                     moduleXML.setModuleCaption(label);
137                 } else moduleXML.setModuleCaption(childNode.getValue
                        ());
138                 moduleXML.setModuleName(childNode.getValue());
139                 continue;
140             }
141             if (childNode.getName().equals(ROOT_BLOCK_ELEMENT)) {
142                 List<Element> blocks = childNode.getChildren();
143                 for (Element block : blocks) {
144                     if (!isContainFields(block)) {
145                         continue;
```

```
146                     }
147                     Element fieldsElement = block.getChild(
                            ROOT_FIELD_ELEMENT);
148                     List<Element> fieldElements = fieldsElement.
                            getChildren();
149
150                     /*
151                     Каждое поле - отдельное измерение.
152                      */
153                     for (Element field : fieldElements) {
154                         FieldXML fieldXML = new FieldXML();
155
156                         if (!field.getChild(TABLENAME_ELEMENT).
                                getValue().equals(CRM_TABLE_FIELD)) {
157                             fieldXML.setTableName(field.getChild(
                                    TABLENAME_ELEMENT).getValue());
158                             moduleXML.setFactTable(field.getChild(
                                    TABLENAME_ELEMENT).getValue());
159                         } else {
160                             fieldXML.setTableName(CRM_TABLE_FIELD);
161                             fieldXML.setCrmEntity(true);
162                         }
163
164                         if (field.getChild(ROOT_ENTITY_ID_ELEMENT)
                                != null) {
165                             fieldXML.setPrimaryKey(field.getChild(
                                    ROOT_ENTITY_ID_ELEMENT)
166                                         .getChild(ENTITY_ID_ELEMENT).
                                                getValue());
167                             if (fieldXML.getTableName().equals(
                                    moduleXML.getFactTable())) {
168                                 moduleXML.setKey(field.getChild(
                                        ROOT_ENTITY_ID_ELEMENT)
169                                             .getChild(ENTITY_ID_ELEMENT)
                                                    .getValue());
170                             }
171                         }
172
173                         fieldXML.setColumnName(field.getChild(
                                COLUMNNAME_ELEMENT).getValue());
174                         fieldXML.setColumnType(field.getChild(
                                COLUMTYPE_ELEMENT).getValue());
175                         String fieldLabel;
176                         if ((fieldLabel = findTranslation(field.
                                getChild(FIELD_LABEL_ELEMENT).getValue
                                (), modstringsNode)) != null
177                                 || (fieldLabel = findTranslation(
                                        field.getChild(
                                        FIELD_LABEL_ELEMENT).getValue()
                                        , mainTranslations)) != null) {
178                             fieldXML.setFieldCaption(fieldLabel);
179                         } else fieldXML.setFieldCaption(field.
                                getChild(FIELDNAME_ELEMENT).getValue())
                                ;
180                         fieldXML.setFieldName(field.getChild(
                                FIELDNAME_ELEMENT).getValue());
181                         fieldXML.setUiType(field.getChild(
                                UI_TYPE_ELEMENT).getValue());
```

```
182                         /*
183                         Если поле является ссылкой на таблицу - обра
                                ботать отдельно.
184                          */
185                         if (fieldXML.isRelated()) {
186                             Element related = field.getChild(
                                    ROOT_RELATED_MODULE_ELEMENT);
187                             try {
188                                 List<Element> modules = related.
                                        getChildren(
                                        RELATED_MODULE_ELEMENT);
189                                 for (Element node : modules) {
190                                     fieldXML.addRelatedModule(node.
                                            getValue());
191                                 }
192                             } catch (NullPointerException e) {
193                                 fieldXML.addRelatedModule(moduleXML.
                                        getModuleName());
194                             }
195                         }
196                         moduleXML.addNewField(fieldXML);
197                     }
198                 }
199             }
200 
201             /*
202 
203              */
204             if (childNode.getName().equals(ROOT_RELATED_LIST_ELEMENT
                    )) {
205                 List<Element> relList = childNode.getChildren();
206                 for (Element relElement : relList) {
207                     if (!relElement.getChild(RELATED_MODULE_ELEMENT)
                            .getValue().equals("")) {
208                         moduleXML.addRelatedModule(relElement.
                                getChild(RELATED_MODULE_ELEMENT).
                                getValue());
209                     }
210                 }
211             }
212         }
213         return moduleXML;
214     }
215 
216     /**
217      * Метод проверки на наличие элементов в блоке.
218      *
219      * @param element - элемент для проверки
220      * @return True - есть поля
221      *         False - нет полей
222      */
223     private boolean isContainFields(Element element) {
224         for (Element child : element.getChildren()) {
225             if (child.getName().equals(ROOT_FIELD_ELEMENT)) {
226                 return true;
227             }
228         }
229         return false;
```

```
230     }
231
232     /**
233      * Метод поиска перевода имени для определённого элемента
234      *
235      * @param source — элемент
236      * @param collection — набор переводов
237      * @return русский перевод элемента
238      */
239     private String findTranslation(String source, Element collection
            ) {
240         List<Element> modstrings = collection.getChildren();
241         for (Element modstring : modstrings) {
242             Element original = modstring.getChild(ORIGINAL_ELEMENT);
243             if (original.getValue().equals(source)) {
244                 return modstring.getChild(TRANSLATED_ELEMENT).
                        getValue();
245             }
246         }
247         return null;
248     }
249 }
```

Листинг А.2. PentahoMondrianService.java

```
1  package org.xerocry.mondrian.endpoints;
2
3  import org.jdom2.JDOMException;
4  import org.xerocry.mondrian.DatabaseConfiguration;
5  import org.xerocry.mondrian.Schema;
6  import org.xerocry.mondrian.endpoints.dtos.responses.
       StringOperationResultDTO;
7  import org.xml.sax.SAXException;
8
9  import javax.ws.rs.GET;
10 import javax.ws.rs.Path;
11 import javax.ws.rs.Produces;
12 import javax.ws.rs.QueryParam;
13 import javax.ws.rs.core.MediaType;
14 import javax.xml.bind.JAXBContext;
15 import javax.xml.bind.Marshaller;
16 import javax.xml.parsers.ParserConfigurationException;
17 import java.io.IOException;
18 import java.io.StringWriter;
19 import java.util.logging.Logger;
20
21 @Path("@plugin.java.rest.path.root@")
22 public class PentahoMondrianService {
23
24     private static Logger log = Logger.getLogger(
           PentahoMondrianService.class.getName());
25
26     public PentahoMondrianService() { }
27
28
29     @GET
30     @Path("/hello")
```

```
31      @Produces(MediaType.APPLICATION_JSON)
32      public StringOperationResultDTO hello() {
33
34          //create result DTO
35          StringOperationResultDTO result = new
                StringOperationResultDTO();
36
37          //fill string
38          result.string = "Hello World from Pentaho Service!";
39
40          //fill status message
41          result.statusMessage.code = "OK_CODE";
42          result.statusMessage.message = "OK_MESSAGE";
43
44          //return result DTO
45          return result;
46      }
47
48      @GET
49      @Path("/addSource")
50      @Produces(MediaType.APPLICATION_JSON)
51      public StringOperationResultDTO addSource(@QueryParam("host")
            String host)
52              throws JDOMException, ParserConfigurationException,
                    SAXException, IOException {
53
54          String url = "http://" + host.trim();
55          DatabaseConfiguration.init(url);
56
57          StringOperationResultDTO result = new
                StringOperationResultDTO();
58
59          result.string = "Hello World from Pentaho Service!";
60
61          result.statusMessage.code = "OK_CODE";
62          result.statusMessage.message = "OK_MESSAGE";
63
64          return result;
65      }
66
67      @GET
68      @Path("/mondrian")
69      @Produces(MediaType.APPLICATION_XML)
70      public String mondrian(@QueryParam("host") String host,
71                             @QueryParam("user") String user,
72                             @QueryParam("pass") String pass)
73              throws ParserConfigurationException, SAXException,
                    IOException, JDOMException {
74
75          String url = "http://" + host.trim();
76  // String url = "http://"+ host.trim() + "?"+ user.trim() + "?"+ pass.trim();
77          DatabaseConfiguration.init(url);
78
79          Schema schema = DatabaseConfiguration.getSchema();
80
81          StringWriter stringWriter = new StringWriter();
82          try {
83              JAXBContext jaxbContext = JAXBContext.newInstance(Schema
```

```
                .class);
84              Marshaller jaxbMarshaller = jaxbContext.createMarshaller
                    ();
85
86              // output pretty printed
87              jaxbMarshaller.setProperty(Marshaller.
                    JAXB_FORMATTED_OUTPUT, true);
88
89              jaxbMarshaller.marshal(schema, System.out);
90              jaxbMarshaller.marshal(schema, stringWriter);
91              if (schema.getCubes().size() == 0) {
92                  throw new Exception("Dump has 0 modules");
93              }
94          } catch (Exception e) {
95              e.printStackTrace();
96          }
97
98          return stringWriter.toString();
99      }
100 }
```

Листинг А.3. DatabaseConfiguration.java

```
1   package org.xerocry.mondrian;
2
3   import org.xerocry.mondrian.xml_elements.VTigerXML;
4   import org.xerocry.mondrian.xml_elements.XMLParser;
5   import org.jdom2.JDOMException;
6   import org.xml.sax.SAXException;
7
8   import javax.xml.parsers.ParserConfigurationException;
9   import java.io.File;
10  import java.io.IOException;
11  import java.io.OutputStreamWriter;
12  import java.net.HttpURLConnection;
13  import java.net.URI;
14  import java.net.URL;
15  import java.net.URLConnection;
16  import java.util.HashMap;
17  import java.util.Iterator;
18  import java.util.Map;
19  import java.util.Scanner;
20  import java.util.logging.Level;
21  import java.util.logging.Logger;
22
23  public class DatabaseConfiguration {
24      private static Logger log = Logger.getLogger(
            DatabaseConfiguration.class.getName());
25      private static String vtigerSource;
26      private static DatabaseConfiguration instance;
27      private static Schema schema;
28
29      /**
30       * Метод, инициализирующий текущую конфигурацию базы данных.
31       *
32       * @param url — адрес домена с API Vtiger
33       * @throws JDOMException
```

```
34          * @throws IOException
35          * @throws ParserConfigurationException
36          * @throws SAXException
37          */
38         public static void init(String url) throws JDOMException,
                IOException, ParserConfigurationException, SAXException {
39             if (instance == null) {
40                 instance = new DatabaseConfiguration(url);
41                 log.info("url = " + url + " in Config");
42                 VTigerXML vTigerXML = new XMLParser(vtigerSource).
                       readXML();
43                 String sourceName = addJDBCResource(vTigerXML.
                       getSourceInfo());
44                 schema = vTigerXML.generateSchema(sourceName);
45             }
46         }
47
48         /**
49          * Геттер для получения экземпляра класса для патерна Синглтон.
50          *
51          * @return экземпляр синглтона
52          */
53         public static DatabaseConfiguration getInstance() {
54             return instance;
55         }
56
57         private DatabaseConfiguration(String vtigerSource) {
58             DatabaseConfiguration.vtigerSource = vtigerSource;
59         }
60
61         public static Schema getSchema() {
62             return schema;
63         }
64
65         /**
66          * Функция создания нового источника данных — БД.
67          *
68          * @return Имя подключения
69          */
70         public static String addJDBCResource(HashMap<String, String>
                sourceInfo) {
71             String body = "{\"changed\": true, \"usingConnectionPool\":
                    true, \"connectSql\": \"\", \"databaseName\": " +
72                     "\""+ sourceInfo.get("db_name") +"\", \"databasePort
                           \": \""+ sourceInfo.get("db_port").substring(1)
                            +
73                     "\", \"hostname\": \""+ sourceInfo.get("db_name") +"
                           \", \"name\": \""+ sourceInfo.get("db_name") +
74                     "\", \"password\": \""+ sourceInfo.get("db_password"
                           ) +"\", \"username\": \""+
75                     sourceInfo.get("db_username") +"\", \"attributes\":
                           {}, \"connectionPoolingProperties\": {}, " +
76                     "\"extraOptions\": {}, \"accessType\": \"NATIVE\",
                           \"databaseType\": {\"defaultDatabasePort\":
                           9001, " +
77                     "\"extraOptionsHelpUrl\": \"http://hsqldb.
                           sourceforge.net/doc/guide/ch04.html#N109DA\", "
                            +
```

```
78                  "\"name\": \"MySQL\", \"shortName\": \"MYSQL\", \"
                        supportedAccessTypes\": " +
79                  "[\"NATIVE\", \"ODBC\", \"JNDI\"] } }";
80
81          try {
82              String domainname = System.getenv("USERDOMAIN");
83              URL url = new URL("http://" + domainname +
84                      "/pentaho/plugin/data-access/api/datasource/jdbc
                            /connection/TestDatasource");
85              HttpURLConnection httpCon = (HttpURLConnection) url.
                    openConnection();
86              httpCon.setDoOutput(true);
87              httpCon.setRequestMethod("PUT");
88              OutputStreamWriter out = new OutputStreamWriter(
89                      httpCon.getOutputStream());
90              out.write(body);
91              out.close();
92              httpCon.getInputStream();
93          } catch (Exception e) {
94              log.log(Level.SEVERE, null, e);
95          }
96          return sourceInfo.get("db_name");
97      }
98  }
```

Листинг А.4. Schema.java

```
1   package org.xerocry.mondrian;
2
3   import org.xerocry.mondrian.mondrian_classes.Cube;
4   import lombok.Getter;
5   import lombok.Setter;
6   import org.xerocry.mondrian.mondrian_classes.Dimension;
7   import org.xerocry.mondrian.mondrian_classes.VirtualCube;
8
9   import javax.xml.bind.annotation.*;
10  import java.util.ArrayList;
11  import java.util.List;
12
13  @Getter
14  @Setter
15  @XmlRootElement(name="Schema")
16  @XmlAccessorType(XmlAccessType.FIELD)
17  public class Schema{
18      @XmlAttribute
19      private String name = "firstVer";
20
21      @XmlElement(name = "Dimension", type = Dimension.class)
22      private List<Dimension> dimensions = new ArrayList<>();
23
24      @XmlElement(name = "Cube", type = Cube.class)
25      private List<Cube> cubes = new ArrayList<>();
26
27      @XmlElement(name = "VirtualCube", type = VirtualCube.class)
28      private List<VirtualCube> virtualCubes = new ArrayList<>();
29
30      public void addCube(Cube cube) {
```

```
31          cubes.add(cube);
32      }
33
34      public Cube getCube(String name) {
35          for (Cube cube : cubes) {
36              if (cube.getName().equals(name)) {
37                  return cube;
38              }
39          }
40          return null;
41      }
42
43      public void addDimension(Dimension dimension, String cubeName) {
44          dimensions.add(dimension);
45      }
46
47      public void addVirtualCube(VirtualCube cube) {
48          virtualCubes.add(cube);
49      }
50
51      public Schema(String name) {
52          this.name = name;
53      }
54
55      /**
56       * Конструктор для JAXB
57       */
58      public Schema() {
59      }
60
61      public Dimension getDimension(String name) {
62          for (Dimension dim : dimensions) {
63              if (dim.getName().equals(name)) {
64                  return dim;
65              }
66          }
67          return null;
68      }
69  }
```

Листинг A.5. VTigerXML.java

```
1   package org.xerocry.mondrian.xml_elements;
2
3   import org.xerocry.mondrian.Schema;
4   import org.xerocry.mondrian.mondrian_classes.Cube;
5   import org.xerocry.mondrian.mondrian_classes.Dimension;
6   import org.xerocry.mondrian.mondrian_classes.VirtualCube;
7   import org.xerocry.mondrian.mondrian_classes.VirtualCubeDimension;
8
9   import java.util.ArrayList;
10  import java.util.HashMap;
11
12  public class VTigerXML {
13      private ArrayList<ModuleXML> xmlModules = new ArrayList<>();
14      HashMap<String, String> sourceInfo = new HashMap<>();
15
```

```
16
17      /**
18       * Основа формирования Mondrian схемы
19       *
20       * @param name — имя схемы
21       * @return JAXB-объект содержащий массив кубов и вирт.кубов
22       */
23      public Schema generateSchema(String name) {
24          Schema schema = new Schema(name);
25
26          /*
27          Для каждого модуля создаётся отдельный куб
28           */
29          for (ModuleXML module : xmlModules) {
30              Cube cube = new Cube(module.getModuleName());
31              cube.setTable(module.getFactTable());
32              cube.setForeignKey(module.getKey());
33              /*
34              Сначала создаются измерения, которые будут включены в ку
                    б.
35               */
36              for (FieldXML field : module.getFieldXMLList()) {
37                  Dimension dim;
38                  if (schema.getDimension(field.getFieldName()) !=
                         null) {
39                      dim = new Dimension(cube.getName() + field.
                             getFieldName(),
40                                 field.getFieldCaption()).generate(field)
                                       ;
41                  } else dim = new Dimension(field.getFieldName(),
                         field.getFieldCaption()).generate(field);
42                  schema.addDimension(dim, cube.getName());
43                  cube.addDimension(dim);
44              }
45              cube.setRelated(module.getRelatedModules());
46              cube.generate();
47              schema.addCube(cube);
48          }
49
50          /*
51          Затем виртуальный куб на основе относящихся к модулю модулям
52           */
53          for (Cube cube : schema.getCubes()) {
54              VirtualCube virCube = new VirtualCube();
55              if (cube.getRelated().size() == 0) {
56                  continue;
57              }
58              for (String related : cube.getRelated()) {
59                  if(schema.getCube(related) == null) continue;
60                  for (Dimension dimension : schema.getCube(related).
                         getDimensions()) {
61                      virCube.addVirtualDimension(new
                             VirtualCubeDimension(dimension, cube.
                             getName()));
62                  }
63              }
64              virCube.setName(virCube.generateName());
65              schema.addVirtualCube(virCube);
```

```
66          }
67          return schema;
68      }
69
70      void addModule(ModuleXML moduleXML) {
71          xmlModules.add(moduleXML);
72      }
73
74      public HashMap<String, String> getSourceInfo() {
75          return sourceInfo;
76      }
77  }
```

Листинг А.6. ModuleXML.java

```
1   package org.xerocry.mondrian.xml_elements;
2
3   import lombok.Getter;
4   import lombok.Setter;
5
6   import java.util.ArrayList;
7
8   @Getter
9   @Setter
10  public class ModuleXML {
11      private String moduleName;
12      private String moduleCaption;
13      private Integer sequenceNum;
14      private String key;
15      private ArrayList<FieldXML> fieldXMLList;
16
17      private boolean crmEntity;
18      private String factTable;
19
20      private boolean related;
21      private ArrayList<String> relatedModules;
22
23      public ModuleXML() {
24          this.fieldXMLList = new ArrayList<>();
25          this.relatedModules = new ArrayList<>();
26      }
27
28      public void addNewField(FieldXML field) {
29          fieldXMLList.add(field);
30      }
31
32      public void addRelatedModule(String module) {
33          relatedModules.add(module);
34      }
35
36  }
```

Листинг А.7. FieldXML.java

```
1   package org.xerocry.mondrian.xml_elements;
```

```
2
3  import lombok.Getter;
4  import lombok.Setter;
5
6  import java.util.ArrayList;
7
8  @Getter
9  @Setter
10 public class FieldXML {
11     private String columnName;
12     private String fieldName;
13     private String columnType;
14     private String tableName;
15     private String fieldCaption;
16     private String uiType;
17     private String primaryKey;
18     private ArrayList<String> relatedModules = new ArrayList<>();
19
20     private static final String REL_MODULE_TYPE = "10";
21
22     private boolean crmEntity;
23
24     public void addRelatedModule(String module) {
25         relatedModules.add(module);
26     }
27
28     public boolean isRelated() {
29         return uiType.equals(REL_MODULE_TYPE);
30     }
31
32
33 }
```

Листинг А.8. TableAdapter.java

```
1  package org.xerocry.mondrian.mondrian_classes;
2
3  import javax.xml.bind.annotation.XmlAttribute;
4  import javax.xml.bind.annotation.adapters.XmlAdapter;
5
6  public class TableAdapter extends XmlAdapter<TableAdapter.
       AdaptedTable, String> {
7
8      @Override
9      public String unmarshal(AdaptedTable v) throws Exception {
10         return v.name;
11     }
12
13     @Override
14     public AdaptedTable marshal(String v) throws Exception {
15         AdaptedTable amf = new AdaptedTable();
16         if (v == null) {
17             return null;
18         }
19         amf.name = v;
20         return amf;
21     }
```

```
22
23      public static class AdaptedTable {
24
25          @XmlAttribute
26          public String name;
27
28      }
29  }
```

Листинг А.9. Dimension.java

```
1   package org.xerocry.mondrian.mondrian_classes;
2
3   import org.xerocry.mondrian.xml_elements.FieldXML;
4   import lombok.Getter;
5   import lombok.Setter;
6
7   import javax.xml.bind.annotation.*;
8   import java.util.ArrayList;
9   import java.util.List;
10
11  @XmlAccessorType(XmlAccessType.FIELD)
12  @Getter
13  @Setter
14  @XmlRootElement(name="Dimension")
15  public class Dimension {
16      @XmlAttribute(name = "name")
17      private String name;
18      @XmlAttribute
19      private String caption;
20      @XmlAttribute
21      private String type="StandardDimension";
22      @XmlElement(name = "Hierarchy", type = Hierarchy.class)
23      private List<Hierarchy> hierarchy = new ArrayList<>();
24
25
26      public Dimension(String name, String caption) {
27          this.name = name;
28          this.caption = caption;
29      }
30
31      public Dimension() {
32      }
33
34      public Dimension(Dimension other) {
35          this.name = other.name;
36          this.caption = other.caption;
37          this.type = other.type;
38          this.hierarchy = other.hierarchy;
39      }
40
41      public Dimension generate(FieldXML field) {
42          Hierarchy hierarchy = new Hierarchy();
43          hierarchy.setName(field.getFieldName());
44
45          Level level = new Level();
46          level.setColumn(field.getColumnName());
```

```
47          level.setName(field.getFieldName());
48          level.setCaption(field.getFieldCaption());
49          level.setType(field.getColumnType());
50          hierarchy.addLevel(level);
51          if (field.getTableName().equals("vtiger_crmentity")) {
52              hierarchy.setTable("vtiger_crmentity");
53          } else hierarchy.setTable(field.getTableName());
54          hierarchy.setCaption(field.getFieldCaption());
55          hierarchy.setPrimaryKey(field.getPrimaryKey());
56          this.hierarchy.add(hierarchy);
57
58          return this;
59      }
60
61
62
63
64  }
```

Листинг А.10. Cube.java

```
1   package org.xerocry.mondrian.mondrian_classes;
2
3
4   import org.xerocry.mondrian.Schema;
5   import org.xerocry.mondrian.xml_elements.FieldXML;
6   import org.xerocry.mondrian.xml_elements.ModuleXML;
7   import lombok.Getter;
8   import lombok.Setter;
9
10  import javax.xml.bind.annotation.*;
11  import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter;
12  import java.util.ArrayList;
13  import java.util.List;
14
15  @XmlAccessorType(XmlAccessType.FIELD)
16  @Getter
17  @Setter
18  public class Cube {
19      @XmlAttribute
20      private String name;
21      @XmlAttribute
22      private String source;
23      @XmlAttribute
24      private String caption;
25      @XmlTransient
26      private String foreignKey;
27      @XmlJavaTypeAdapter(TableAdapter.class)
28      private String table;
29      @XmlElement(name = "DimensionUsage", type = DimensionUsage.class
            )
30      private List<DimensionUsage> dimensionUsages;
31      @XmlTransient
32      private List<String> related = new ArrayList<>();
33      @XmlTransient
34      private List<Dimension> dimensions;
35
```

```
36      public Cube generate() {
37          for (Dimension dimension : dimensions) {
38              DimensionUsage dimensionGen = new DimensionUsage();
39              dimensionGen = dimensionGen.generate(dimension);
40              dimensionGen.setForeignKey(foreignKey);
41              dimensionUsages.add(dimensionGen);
42          }
43          return this;
44      }
45
46      public Cube(String name) {
47          dimensions = new ArrayList<>();
48          dimensionUsages = new ArrayList<>();
49          this.name = name;
50      }
51
52      public Cube() {
53          dimensionUsages = new ArrayList<>();
54          dimensions = new ArrayList<>();
55      }
56
57      public void addDimension(Dimension dimension) {
58          dimensions.add(dimension);
59      }
60  }
```

Листинг A.11. VirtualCube.java

```
1   package org.xerocry.mondrian.mondrian_classes;
2
3   import lombok.Getter;
4   import lombok.Setter;
5
6   import javax.xml.bind.annotation.XmlAccessType;
7   import javax.xml.bind.annotation.XmlAccessorType;
8   import javax.xml.bind.annotation.XmlAttribute;
9   import javax.xml.bind.annotation.XmlElement;
10  import java.util.ArrayList;
11  import java.util.List;
12
13  @Getter
14  @Setter
15  @XmlAccessorType(XmlAccessType.FIELD)
16  public class VirtualCube {
17      @XmlElement(name = "VirtualCubeDimension", type =
            VirtualCubeDimension.class)
18      private List<VirtualCubeDimension> dimensionUsages;
19
20      @XmlAttribute
21      private String name;
22
23      /**
24       * Конструктор для JAXB
25       */
26      public VirtualCube() {
27          dimensionUsages = new ArrayList<>();
28      }
```

```
29
30      public void addVirtualDimension(VirtualCubeDimension dimension)
            {
31          dimensionUsages.add(dimension);
32      }
33
34      public String generateName() {
35          VirtualCubeDimension dim = dimensionUsages.get(1);
36          return dim.getCaption() + "Virtual";
37      }
38  }
```

Листинг А.12. Код на Java

```
1   <!DOCTYPE html>
2   <html lang="en"><head>
3   <meta charset="utf-8">
4   <title>JavaScript Form Validation using a sample registration form</
        title>
5   <link rel='stylesheet' href='js-form-validation.css' type='text/css'
        />
6   <script src="main.js"></script>
7   </head>
8   <body onload="document.registration.url.focus();">
9   <h1>Mondrian publish Form</h1>
10  <p>Use tab keys to move from one input field to the next.</p>
11  <form name='registration' onSubmit="return publish();">
12  <ul>
13  <li><label for="url">VTiger URL:</label></li>
14  <li><input type="text" name="url" size="50" /></li>
15  <li><label for="username">Username:</label></li>
16  <li><input type="text" name="username" size="12" /></li>
17  <li><label for="passid">Password:</label></li>
18  <li><input type="password" name="passid" size="12" /></li>
19  <li><label for="source">Datasource:</label></li>
20  <li><select name="source" id = "sources">
21  <option selected="" value="Default">(Please select a datasource)</
        option>
22  </select></li>
23  <li><input type="submit" name="submit" value="Submit" /></li>
24  </ul>
25  </form>
26  <script src="sample-registration-form-validation.js"></script>
27  <script>
28  var select = document.getElementById("sources");
29  var myobject = getSources();
30  for(var index in myobject) {
31      select.options[select.options.length] = new Option(myobject[
            index].$, index);
32  }
33  </script>
34  </body>
35  </html>
```

Листинг А.13. main.js

```
function getSources() {
    var e = new XMLHttpRequest;
    e.open("GET", "/pentaho/plugin/data-access/api/datasource/jdbc/
        connection", !1), e.send();
    var t = JSON.parse(e.responseText),
        n = t.Item;
    return n }

function getMondrian() {
    var e = new XMLHttpRequest;
    e.open("GET", "/pentaho/plugin/pentaho-vtiger-plugin/api/
        mondrian", !1), e.send();
    var t = e.responseXML;
    return t }

function publish() {
    var e = (new XMLSerializer).serializeToString(getMondrian()),
        t = { uploadInput: "Content-Type: text/xml\r\n" + e,
            parameters: "overwrite=true;DataSource=" + document.
            getElementById("sources").options[document.
            getElementById("sources").selectedIndex].text,
            schemaFileInfo: "firstVer.xml", xmlaEnabledFlag: "false
            " },
        n = String(Math.random()).slice(2),
        r = "--" + n + "\r\n",
        i = "--" + n + "--\r\n",
        s = ["\r\n"];
    for (var o in t) s.push('Content-Disposition: form-data; name="'
        + o + '"\r\n\r\n' + t[o] + "\r\n");
    s = s.join(r) + i, console.log(s);
    var u = new XMLHttpRequest;
    u.open("PUT", "/pentaho/plugin/data-access/api/datasource/
        analysis/import", !0), u.withCredentials = !0, u.
        setRequestHeader("Authorization", "Basic " + btoa("admin:
        password")), u.setRequestHeader("Content-Type", "multipart/
        form-data; boundary=" + n), u.onreadystatechange = function
        () {
        if (this.readyState != 4) return;
        alert(this.responseText) }, u.send(s);
    var a = new XMLHttpRequest;
    a.open("GET", "pentaho/plugin/data-access/api/datasource/
        analysis/catalog/firstVer", !0), a.send();
    var f = a.responseText;
    console.log(f) }

function getAnalysis() {
    var e = new XMLHttpRequest;
    e.open("GET", "pentaho/plugin/data-access/api/datasource/
        analysis/ids", !0), e.send();
    var t = e.responseXML;
    console.log(t) }
define("../main", function() {});
```

Листинг А.14. Код на Javajs-form-validation.cs

```
h1 {
```

```
    margin-left: 70px;
}
form li {
    list-style: none;
    margin-bottom: 5px;
}

form ul li label{
    float: left;
    clear: left;
    width: 100px;
    text-align: right;
    margin-right: 10px;
    font-family:Verdana, Arial, Helvetica, sans-serif;
    font-size:14px;
}

form ul li input, select, span {
    float: left;
    margin-bottom: 10px;
}

form textarea {
    float: left;
    width: 350px;
    height: 150px;
}

[type="submit"] {
    clear: left;
    margin: 20px 0 0 230px;
    font-size:18px
}

p {
    margin-left: 70px;
    font-weight: bold;
}
```