МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное Государственное Бюджетное Образовательное учреждение высшего профессионального образования

«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

(ФГБОУ ВПО «КубГУ»)

Кафедра математического моделирования

КУРСОВАЯ РАБОТА

Прототип интерфейса пользователя для системы трансляции заданий с естественного языка в SQL

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.А. Бабичев

(подпись, дата)

Факультет ФКТиПМ курс 3

Направление 010400.62 – Прикладная математика и информатика

Научный руководитель,

канд. тех. наук, доцент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н.В. Бессарабов

Подпись Дата

Нормоконтролер,

канд. физ.-мат. наук, доцент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С. Е. Рубцов

Подпись Дата

Краснодар 2014

РЕФЕРАТ

Курсовая работа 30 с., 5 рис., 3 источников, 8 приложения.

ORM МОДЕЛЬ, TOMITA-ПАРСЕР, PHP, JAVASCRIPT, MYSQL, SQL, HTML, CSS, BOOTSTRAP, FONT-AWESOME, PSR-0, PSR-1, PSR-2, CSP, PHPMYADMIN.

Объектом исследования является Tomita-парсер и создание взаимодействия с ним через веб-интерфейс.

Цель работы:

* построение сервера взаимодействия с Tomita-парсером;
* разработка базы данных в СУБД MySQL;
* реализовать ORM модели для взаимодействия с объектами;
* реализация Web-интерфейса доступа к данным.

Исследования проводились с помощью стандартов PSR-0, PSR-1, PSR-2, технологии CSP в СУБД MySQL и технологий: PHP 5.3, JavaScript, HTML5, CSS3, Bootstrap 3, Font-awesome 4, PHPixie2, apache2, jQuery.

С использованием указанных стандартов и технологии:

* построен сервер взаимодействия с Томита-парсером;
* создана база данных в СУБД MySQL;
* созданы модели ORM для взаимодействия с объектами;
* реализован Web-интерфейс доступа к данным.

Результаты работы могут быть использованы для автоматизации обучения студентов языку SQL.

СОДЕРЖАНИЕ

[Введение 4](#_Toc428044106)

[1 Постановка задачи 5](#_Toc428044107)

[2 Выбор технологий 6](#_Toc428044108)

[3 Структура сервиса 11](#_Toc428044109)

[4 Древовидная структура в языке SQL 12](#_Toc428044110)

[5 JavaScript, построение дерева 14](#_Toc428044111)

[6 Взаимодействие с Томита-парсером 15](#_Toc428044112)

[7 Время реагирования 17](#_Toc428044113)

[Заключение 19](#_Toc428044114)

[Список используемых источников 20](#_Toc428044115)

[Приложение A Настройки PHPixie и Apache2 21](#_Toc428044116)

[Приложение Б Список терминов 22](#_Toc428044117)

[Приложение В Ответ от сервера 1 24](#_Toc428044118)

[Приложение Г Ответ от сервера 2 25](#_Toc428044119)

[Приложение Д JS создание дерева 26](#_Toc428044120)

[Приложение Е Обработка дерева 28](#_Toc428044121)

[Приложение Ж Конфигурация парсера 29](#_Toc428044122)

[Приложение З Результат парсера 30](#_Toc428044123)

# ВВЕДЕНИЕ

Задача трансляции инструкции с естественного (русского) языка на язык SQL.

Часть необходимых преобразований выполняется тривиально. Это перевод некоторых слов или словосочетаний естественного языка в термины SQL или термины схемы базы. Понятно, что эти действия осуществимы при наличии словаря SQL и словаря схем базы данных.

Проблема трансляции с ЕЯ в SQL разрабатывалась давно, но даже в таких серьезных работах, как [1] рассматривается упрощённый формат фразы естественного языка.

Существует подход, в котором формируется так называемое управляемое подмножество ЕЯ ограничивает допустимую структуру фраз языка. В настоящей работе этот подход не рассматривается.

Основные проблемы возникают из-за не определеннозначности естественного языка. В распространённых формулировках заданий, как минимум используются эллипсис и анафора.

Предлагаемая работа ориентирована на создание графического интерфейса программы для трансляции задания с ЕЯ в язык SQL, в которой предоставляет интерфейс к Томита-парсеру. В частности, решена задача повышения быстродействия системы интерфейс Томита-парсер.

# Постановка задачи

Создается интерфейс для преобразования заданий с естественного языка (ЕЯ) в специализированный декларативный язык SQL. Часть преобразований выполняется тривиально: перевод слов, словосочетаний. Что можно преобразовать, имея схему базы.

Существуют словари, SQL схемы c их помощью осуществима работа тривиальных преобразований. Трансляции более сложных заданий, использующих: эллипсис, анафора, должны выполняться выполнятся при помощи парсера, использующего грамматики, или наборов правил схем.

Желательно получить время реагирования меньше ½ секунды. Пользователь, использующий систему, не будет ждать несколько секунд, минут, часов. Трансляция должна выполнятся, как можно быстрее.

Предполагается что разработанная программа будет использована как для обучения языку SQL, так и в качестве рабочего инструмента. Интерфейс должен обеспечить удобную работу пользователя (UX).

Томита-парсер – создан для извлечения структурированных данных из текста на естественном языке. Вычленение фактов происходит при помощи контекстно-свободных грамматик и словарей ключевых слов.

Существует словарь ключевых слов и грамматика определенной базы данных. Именно для неё и будем решать задачу, используя Томита–парсер.

# Выбор технологий

Парсер является отдельным приложением, в которое нужно передавать данные, в этом поможет Hypertext Preprocessor (PHP). Для взаимодействия с клиентом есть технология Ajax, до его появления, интерактивные взаимодействия со страницей были тяжеловесными. Каждое из них требовало перезагрузки страницы, которая создавалась на сервере.

В этой модели основной единицей взаимодействия была страница. Неважно, какой объём информации отправлялся из браузера на сервер – результатом была новая страница.

Избежали трату лишнего трафика пользователя, ускорили взаимодействие с системой. В качестве точки опоры была выбрана структура Model View Controller (MVC). Выбрав открытый фреймворк, было сэкономлено время на написание собственной MVC структуры.

Фреймворков для выбора много: Symfony, Laravel, PHPixie, Yii и другие. У каждого есть свои достоинства и недостатки. Symfony – очень функциональный, но в это же время и медленный фреймворк. Laravel, к моей огромной радости, использует стандарт PSR-4, а значит каждый класс в папке App, у которого будет правильно задано пространство имен, но он, фреймворк, имеет большой недостаток, огромные объекты, которые хранятся в оперативной памяти, с Yii те же проблемы. Так был выбран PHPixie.

Быстрый и простой фреймворк, он и был выбран для разработки веб-сервиса, структура проста. Изначально запрос от пользователя идет на сервер.

PHPixie фреймворк использует менеджер пакетов Composer, занимающийся автоматической подгрузкой классов, которые должны использоваться в проекте. Если класс не используется, Composer его и не подключит, тем самым сэкономив оперативную память.

После инициализации класса, автоматической подгрузки, создается Bootstrap контейнер, который подгружает PHPixie-Core, устанавливает маршруты.

Фреймворк вызывает метод handle\_http\_request() и сохраняет данные из HTTP в параметр ядра $request. Происходит поиск и вызов маршрута. Создание контроллера и вызов метода before(), затем action(). В методе action() происходит рендер Haml, он же язык разметки для упрощённой генерации HTML.

Инициализация параметра $view, в котором и находится класс MtHaml. Так же контроллер устанавливает внутри шаблонные переменные. Контроллер вызывает after(), устанавливает все глобальные переменные $\_SESSION, $\_COOKIE. Отправляя $response, отрендеренный шаблон с установленным заголовком, пользователю. Подробности можно увидеть на рисунке 1.

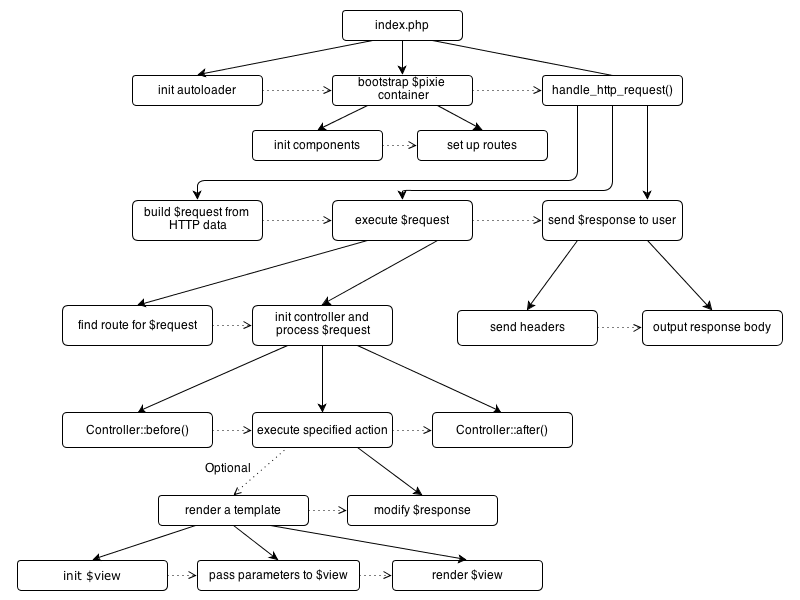


Рисунок 1 – Выполнение запроса пользователя

Так же у фреймворка есть модуль PHPixie-ORM, объектно–реляционное отображение, технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных».

Тем самым, можно работать уже не с привычными таблицами, а их моделями. Например, выбрать сотрудника из отдела номер пятьдесят. Сотрудники – это таблица базы данных, в которой находится информация конкретном сотруднике и существует информация за каким отделом он, сотрудник, закреплен.

В SQL, привычно, просим таблицу найти всех сотрудников из отдела номер пятьдесят, в случае же объектно-реляционного отображения, мы просим модель, или сущность, найти нам конкретного сотрудника или сотрудников. Данные баз данных, и отношения будут представлены в объектно-ориентированной абстракции, чтобы представить более просто, это позволит использовать таблицы, как если бы они были классы и опустить избыточные задачи, такие как обновление, удаление, вставка и выборка.

Взаимодействие с пользователем обычно сопровождается красивым интерфейсом: кнопки, фон, поле ввода данных. Все это оставляет след в памяти о сервисе. Создавать, продумывать свои шаблоны CSS, HTML задача не из легких, для этого нужно иметь определенные дизайнерские способности.

Так как помимо дизайна требуется создать взаимодействие с Томита-парсером, что займет не мало времени, было принято решение использовать HTML, CSS фреймворк. Один из первых, рассматриваемых фреймворков был 960 Grid System, но его давно никто не обновлял. В будущем с ним будут проблемы. Следующий рассмотренный фреймворк был Foundation 5, его основная проблема с браузерами Safari, он не верно генерировал страницу. Kube, большой и тянет за собой много JavaScript кода, что очень тормозило систему.

Для работы был выбран Twitter Bootstrap 3, создан в застенках компании Twitter. Данный набор инструментов для вёрстки позволяет адаптировать веб интерфейс и под мобильные приложения. В нём есть ряд преимуществ, благодаря которым BS считается самым популярным из себе подобных. Масштабируемость – добавление новых элементов не нарушает общую структуру. Лёгкая настройка – редактирование стилей производится путём создания новых CSS-правил, которые исполняются вместо стандартных.

Пользователю намного приятнее видеть иконку кнопки и её подпись, для этого используется Font–Awesome Framework. Набор шрифтов, в векторном виде, позволяют использовать иконки любого размера без потери качества. Полное управление шрифтами через язык стилей, CSS.

Технология Ajax не простая. Она имеет более трехсот видов уведомлений. Запрос к серверу бывает асинхронный и не асинхронный. Асинхронный запрос вызывает функцию и не ожидая результата функции, переходит к вызову следующей. Тем самым, если пользователь нажимал на кнопку для получения какой–либо информации, он просто ничего может не увидеть так, как запрос выполнился не мгновенно и асинхронный запрос работает уже с другими участками Document Object Model (DOM) дерева.

Согласно DOM-модели, документ является иерархией, деревом. Каждый HTML-тег образует узел дерева с типом «элемент». Вложенные в него теги становятся дочерними узлами. Для представления текста создаются узлы с типом «текст».

Для работы с Ajax технологией, DOM-деревом и отслеживанием событий используем фреймворк jQuery. С его помощью можно намного быстрее реализовывать события и обрабатывать их. Фреймворк по указателю строит DOM-дерево, с которым можно проводить различные действия. Пример простого DOM-дерева можно посмотреть на рисунке 2. Технология Ajax служит для взаимообмена данными с Hypertext Preprocessor без перезагрузки страницы. В проекте будем использовать не асинхронный $.ajax() запрос, чтобы дожидаться обмена данными и приступить к выполнению следующих операций.

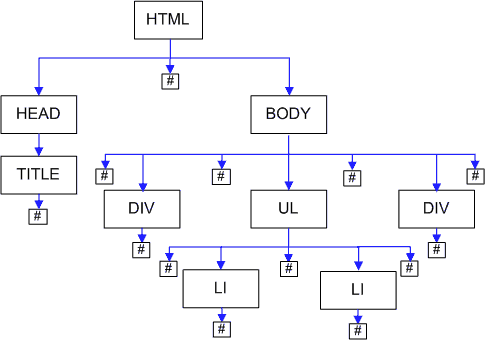


Рисунок 2 – Пример, DOM-дерево

# Структура сервиса

В корне сервиса находятся папки и файлы: assets/, classes/, exec/, vendor/, web/, .htaccess, composer.json, composer.phar. .htaccess – файл конфигураций для Apache2, в нем сохранены PHP и Apache2 настройки.

Код .htaccess вынесен в приложение A.

Composer.json – файл правил Composer.phar, какие библиотеки нужно подключить к проекту. Все библиотеки Composer сохраняются в vendor/, там и находятся настройки подключения.

Код файла composer.json вынесен в приложение A.

Папка assets/ – хранит отрендеренные файлы шаблонов, конфигурации классов, подключение к базе данных, время жизни кук, сессий и правила маршрутов сайта. Все конфигурации находятся в папке assets/config/, шаблоны в папке assets/views/.

В папка classes/ сохранены исходные Hypertext Preprocessor файлы, расположенные форматом автозагрузки PSR–0 [Приложение Б].

Главный namespace приложения является \App. Из правил, формата PSR–0 следует, что в папке classes должна находится папка App. Уже в находятся исполняемые контроллеры, модели базы и классы работы с сайтом.

Папка, в которой находится Томита–парсер и его конфигурации называется exec/. В папке web/ находится файл index.php, который в свою очередь и подгружает файл autoload.php из папки vendor/, дальше выделяет память под объект главный фреймворка PHPixiе и запускает handle\_http\_request() из рисунка 1, структура MVC. Так же в папке web находятся Bootstrap, Font–Awesome, jQuery, изображения и собственные CSS–файлы.

# Древовидная структура в языке SQL

Дерево — это связный ациклический граф. Связность означает наличие путей между любой парой вершин, ацикличность — отсутствие циклов и то, что между парами вершин имеется только по одному пути. Древовидная структура является одним из способов представления иерархической структуры в графическом виде. Древовидной структурой называется благодаря тому, что граф выглядит как перевернутое дерево. По этой же причине говорят, что корневой узел (корень) находится на самом верху, а листья — внизу.

Любой SQL запрос можно представить в виде древовидного дерева. Например, выбрать сотрудника с максимальной зарплатой.

Нет данных по выбору: фамилии, имени, зарплате. Нужно выбрать сотрудника, а значит в результат должны войти все данные сотрудника. Дальше идет групповая функция, в последствии она будет подзапросом с поиском максимальной зарплаты.

Групповые функции — это функции, работающие множеством строк возражающие один результат.

Построим дерево с помощью Hypertext Preprocessor. Код вынесен в приложение В.

С помощью построенного дерева должны получить:

SELECT \* FROM `EMP` WHERE `EMP`.`SALARY`=(SELECT MAX(`L1`.`SALARY`) FROM `EMP L1`)

Результатом этого запроса будут все сотрудники с заработной платой равной максимальной.

Чтобы избежать этого и других возможных ошибок, система работает в ручном режиме. Дерево строится по данным которые введет и выберет посетитель. Чтобы ограничить получаемых данных в базе данных Oracle добавляют параметр ROWNUM <= 1. Тем самым выбирают только одного сотрудника. Результат вынесен в приложение Г.

И данный запрос преобразуется уже в следующий код:

SELECT \* FROM `EMP` WHERE `EMP`.`SALARY`=(SELECT MAX(`L1`.`SALARY`) FROM `EMP L1`) AND `ROWNUM` <= 1;

# JavaScript, построение дерева

Система – это трехзвенное приложение, которое взаимодействует с пользователем. Первое с чем сталкиваются клиент-серверные приложения – производительность. Если массив, дерево, будет создаваться в PHP, его нужно будет получить в JS.

Пользователь, каждый раз, обновляя сайт, отправляет пакет данных, который будет нагружать сервер. Было принято решение в использовании технологии Ajax, потому что он поддерживает – асинхронность, но, каждый запрос – это выполнение большого участка кода в PHP и получение результата в JavaScript, поэтому реализация генератора дерева сделана на JavaScript.

С помощью Ajax отправляем данные на парсер. По окончанию сборки дерева. Оно отправляется в PHP, где уже и обрабатывается.

Код вынесен в приложение Д.

Обработка в PHP данных полученных из Ajax.

Код вынесен в приложение Е.

action\_struct\_code – является действием контроллера IO. Данный контроллер на прямую работает с классом Tomita для взаимодействия клиентской части с парсером. Данный метод уменьшает дерево которое пришло из JavaScript и возвращает результат назад.

# Взаимодействие с Томита-парсером

Парсер имеет конфигурационный файл, который считывается Томитой при работе. Файл содержит настройки наборы: расширений, словарей, грамматик, лемм. Так же в конфигурации хранится путь входного файла и результативного, и формат каждого из файлов.

Создание конфигурационного файла происходит только его не существует, или файл из которого строится конфигурационный файл изменился.

Код вынесен в приложение Ж.

Результатом работы парсера – построенное xml–дерево, вынесено в приложение З.

Дерево XML преобразуется в массив, который в последствии передается через Ajax пользователю, рисунок 3.

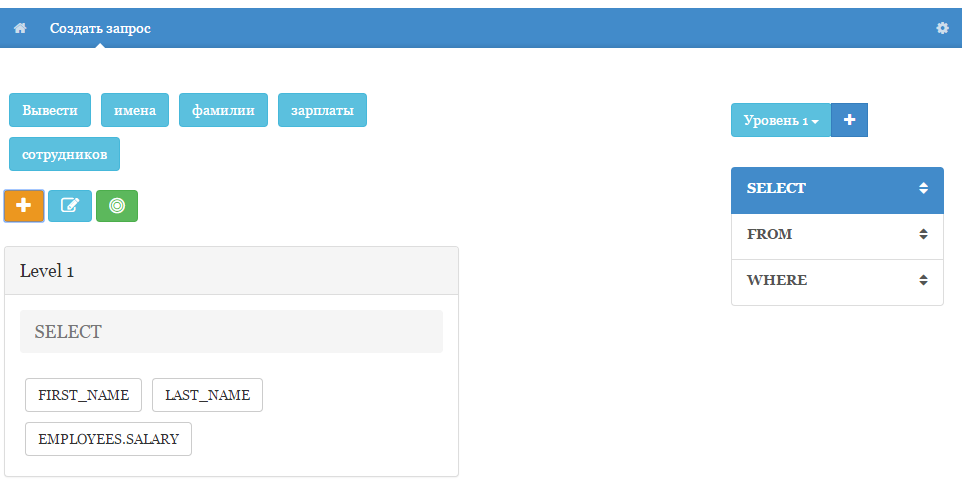


Рисунок 3 – Отображение DOM-дерева в пользовательском интерфейсе

JavaScript получил массив, создал объект объектов построенный по DOM–дереву, рисунок 4, которое пришло с сервера. Разбивает на отдельные части и передает данные синхронизатору. Синхронизатор проверяет валидность данных и выводит в браузер.

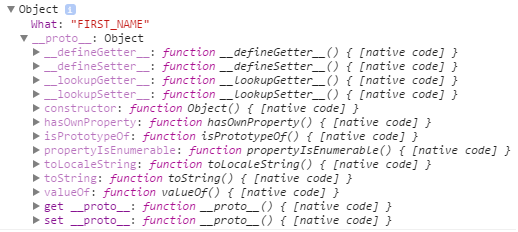


Рисунок 4 – Объект и его методы.

# Время реагирования

Основная задача, быстрое взаимодействие с пользователем. Во время запроса к сервису, пользователь, обращается к Томита-парсер, для обработки данных, если это требуется. Так, как парсер, является отдельным приложением, его нужно запускать отдельным процессом, тем самым сложность задачи увеличивается.

Томита-парсер, создает, результативный, файл, который нужен для обработки. Если после запуска считать сразу файл, можно получить старые данные. То есть, результативный файл нужно удалить перед запуском.

Для чтения файла, нужно сначала узнать, существует ли он. Мы не знаем, работает ли еще парсер или уже завершился. Поэтому создаем цикл, в котором обращаемся к файлу. Если файл существует, значит, парсер завершил работу и данные можно считывать.

Запрос занял 2,23 секунды, что очень много для простого запроса. Первая проблема, если Томита-парсер завершится принудительно операционной системой, файл не создастся, а скрипт будет ждать появления файла. Тем самым увеличивая время реагирования. Проверка существования, осуществляется после сканирования папки, что тоже является не простым процессом.

Решение, нужно запустить приложение, Томита-парсер, и дождаться, когда оно завершится. В ОС запущенный процесс имеет уникальный идентификатор (ID). Считаем список процессов и ищем Томита-парсер если его нет в списке, то проверяем существует ли результативный файл. Тем самым время реагирования можно сократить. Проверка файла осуществляется всего один раз, а не в цикле, причем, этот метод гарантирует обработку сразу после завершения Томита-парсера. Пользователь получит максимально быстрый ответ.

На рисунке 6 можно увидеть, что текущим алгоритмом время реагирования составило 0,053 секунды, что в 42 раза быстрее предыдущего запроса на рисунке 5.

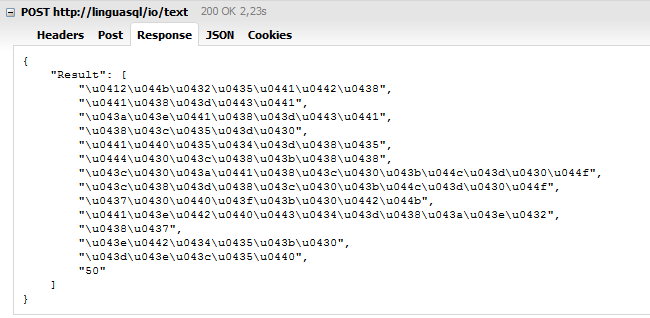


Рисунок 5 – Результат POST-запроса к сервису

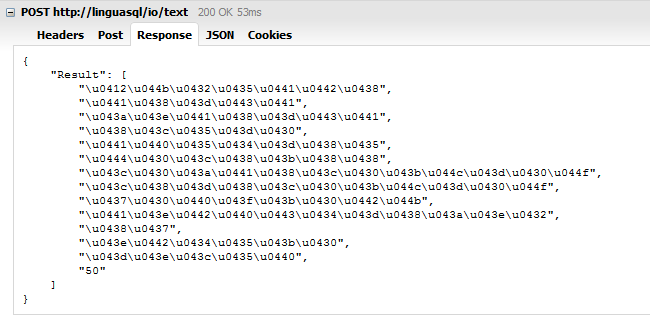


Рисунок 6 – Результат POST-запроса к сервису (улучшенный алгоритм)

# ЗАКЛЮЧЕНИЕ

Результатом данной работы являются прототип системы обучения. Система может быть применена как в обучении, так и в профессиональной сфере.

Изучены технологии: PHPixie, Bootstrap, jQuery, Haml, формат архитектуры приложения PSR–0.

Реализован программный интерфейс к парсеру, открывающий доступ к взаимодействию с ним по средствам технологий Hypertext Preprocessor и Ajax.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Найханова, Л.В. Методы и алгоритмы трансляции естественно–языковых запросов к базе данных в SQL-запросы / Л.В. Найханова, И.С. Евдокимова. – Улан-Удэ: ВСГТУ, 2004. – 148 с.
2. Бессарабов, Н.В. Реализация графовых моделей данных и знаний в Cache / Н.В. Бессарабов, А.Г. Коблов // Экологический вестник научных центров ЧЭС, 2006, Приложение. с. 64–68.
3. Бессарабов, Н. В. Гипотеза Сепира-Уорфа и инвариантные структуры данных // Обозрение прикладной и промышленной математики, 2002, т. 9. в. 2. с. 336-337.

# ПРИЛОЖЕНИЕ A

НАСТРОЙКИ PHPixie и Apache2

Код настройки .htaccess

RewriteEngine On

RewriteBase /

RewriteCond %{REQUEST\_FILENAME} !–f

RewriteRule .\* /web/index.php?$0 [PT,L,QSA]

JSON файл для Composer.

"require": {

"phpixie/core": "2.\*@dev",

"phpixie/db": "2.\*@dev",

"phpixie/orm": "2.\*@dev",

"phpixie/auth": "2.\*@dev",

"phpixie/haml": "2.\*@dev",

"mthaml/mthaml": "dev–master"

}

# ПРИЛОЖЕНИЕ Б

СПИСОК ТЕРМИНОВ, ВСТРЕЧАЮЩИХСЯ В РАБОТЕ

PSR–0 – Стандарт автозагрузки

* Полностью определённое пространство имён и имя класса должны иметь следующую структуру: \<Vendor Name>\(<Namespace>\)\*<Class Name>.
* Каждое пространство имён должно начинаться с пространства имён высшего уровня, указывающего на разработчика кода («имя производителя»).
* Каждое пространство имён может включать в себя неограниченное количество вложенных подпространств имён.
* Каждый разделитель пространства имён при обращении к файловой системе преобразуется в РАЗДЕЛИТЕЛЬ\_ИМЁН\_КАТАЛОГОВ.
* Каждый символ \_ («знак подчёркивания») в ИМЕНИ\_КЛАССА преобразуется в РАЗДЕЛИТЕЛЬ\_ИМЁН\_КАТАЛОГОВ. При этом символ \_ («знак подчёркивания») не обладает никаким особенным значением в имени пространства имён (и не претерпевает преобразований).
* При обращении к файловой системе полностью определённое пространство имён и имя класса дополняются суффиксом .php.
* В имени производителя, имени пространства имён и имени класса допускается использование буквенных символов в любых комбинациях нижнего и верхнего регистров.

Автозагрузка PSR–0 сопровождается следующим кодом.

function autoload($className)

{

    $className = ltrim($className, '\\');

    $fileName  = '';

    $namespace = '';

    if ($lastNsPos = strrpos($className, '\\')) {

        $namespace = substr($className, 0, $lastNsPos);

        $className = substr($className, $lastNsPos + 1);

        $fileName  = str\_replace('\\', DIRECTORY\_SEPARATOR, $namespace) . DIRECTORY\_SEPARATOR;

    }

    $fileName .= str\_replace('\_', DIRECTORY\_SEPARATOR, $className) . '.php';

    require $fileName;

}

# ПРИЛОЖЕНИЕ В

ОТВЕТ ОТ СЕРВЕРА

$SQL = [

“SELECT” => [“\*”],

“FROM” => [“EMP”],

“WHERE” => [

“EMP.SALARY” => [

“SELECT” => “MAX(L1.SALARY)”,

“FROM” => “EMP L1”

]

]

];

# ПРИЛОЖЕНИЕ Г

ОТВЕТ ОТ СЕРВЕРА

$SQL = [

“SELECT” => [“\*”],

“FROM” => [“EMP”],

“WHERE” => [

“EMP.SALARY” => [

“SELECT” => “MAX(L1.SALARY)”,

“FROM” => “EMP L1”

],

“AND” => [

“ROWNUM” => [“<=”, 1]

]

]

];

# ПРИЛОЖЕНИЕ Д

JAVASCRIPT СОЗДАНИЕ ДЕРЕВА

function *obj\_copy*(\_obj) {  
 NewObj = new *Object*();  
 for (var key in \_obj) {  
 if (\_obj.hasOwnProperty(key)) {  
 NewObj[key] = \_obj[key];  
 }  
 }  
 return NewObj;  
}

var current\_level = 0;  
  
*// array*var sql\_models\_for\_level = {  
 SELECT: [],  
 FROM: [],  
 WHERE: []};  
  
*// Initialize Tree*var modif\_sql\_models\_for\_level = *obj\_copy*(sql\_models\_for\_level);  
modif\_sql\_models\_for\_level.Active = 'SELECT';  
modif\_sql\_models\_for\_level.Data = [];  
var tree = *obj\_copy*(modif\_sql\_models\_for\_level);  
$.each(tree, function (key1, value1) {  
 if (key1 != 'Active' && key1 != 'Data') {  
 tree[key1] = *obj\_copy*(modif\_sql\_models\_for\_level);  
 $.each(tree[key1], function (key2, value2) {  
 if (key2 != 'Active' && key2 != 'Data') {  
 tree[key1][key2] = *obj\_copy*(modif\_sql\_models\_for\_level);  
 $.each(tree[key1][key2], function (key3, value3) {  
 if (key3 != 'Active' && key3 != 'Data') {  
 tree[key1][key2][key3] = *obj\_copy*(modif\_sql\_models\_for\_level);  
 $.each(tree[key1][key2][key3], function (key4, value4) {  
 if (key4 != 'Active' && key4 != 'Data')  
 tree[key1][key2][key3][key4] = *obj\_copy*(modif\_sql\_models\_for\_level);  
 });  
 }  
 });  
 }  
 });  
 }  
});

# ПРИЛОЖЕНИЕ Е

ОБРАБОТКА ДЕРЕВА

function handler\_sql\_code($array)  
{  
 foreach($array as $k => $v) {  
 if (*is\_array*($v) && $k != 'Data' && *count*($v))  
 $array[$k] = $this–>handler\_sql\_code($v);  
 if ($k != 'Data' && (!isset($v['Data']) || !*count*($v['Data']) || $k == 'Active'))  
 unset($array[$k]);  
 }  
 return $array;  
}  
  
public function action\_struct\_code()  
{  
 $array = $this–>request–>post('tree');  
 $this–>give\_api($this–>handler\_sql\_code($array));  
}

# ПРИЛОЖЕНИЕ Ж

КОНФИГУРАЦИЯ ТОМИТА

if (!*file\_exists*($path) || (*filectime*($path) > *filectime*($p))) {  
  
 $content = *strtr*($content, array(  
 '{PATH\_PRETTY}' => self::*change\_ds*(self::*path\_pretty*()),  
 '{PATH\_INPUT}' => self::*change\_ds*(self::*path\_input*()),  
 '{PATH\_OUTPUT}' => self::*change\_ds*(self::*path\_output*()),  
 '{PATH\_DICTIONARY}' => self::*change\_ds*(self::*path\_mydic*()),  
 '{TOMITA\_TYPE}' => self::*get\_type\_config*()  
 ));  
  
 *file\_put\_contents*($path, $content);  
}

Запуск парсера:

try {  
 *exec*(self::*path\_Tomita*() . " " . self::*path\_config*());  
}  
catch (\Exception $e) {  
 *passthru*(self::*path\_Tomita*() . " " . self::*path\_config*());  
}

# ПРИЛОЖЕНИЕ З

РЕЗУЛЬТАТ ПАРСЕРА

<Select FactID="0" LeadID="0" FieldsInfo="n0;" pos="8" len="5" sn="0" fw="1" lw="1">  
 <What val="EMPLOYEES.FIRST\_NAME"/>  
 <From val="EMPLOYEES.FIRST\_NAME"/>  
</Select>  
<Select FactID="1" LeadID="0" FieldsInfo="n1;" pos="14" len="7" sn="0" fw="2" lw="2">  
 <What val="EMPLOYEES.LAST\_NAME"/>  
 <From val="EMPLOYEES.LAST\_NAME"/>  
</Select>  
<Select FactID="2" LeadID="0" FieldsInfo="n2;" pos="22" len="8" sn="0" fw="3" lw="3">  
 <What val="EMPLOYEES.SALARY"/>  
 <From val="EMPLOYEES.SALARY"/>  
</Select>