УДК 004.415.2
DOI: 10.15827/0236-235X.124.653-658

Дата подачи статьи: 21.09.18
2018. Т. 31. № 4. С. 653–658

Система корпус-менеджер: архитектура и модели корпусных данных

Д.Р. Мухамедшин [1], *аспирант, damirmuh@gmail.com*
Д.Ш. Сулейманов [1, 2], *д.т.н., профессор, ipsanrt@mail.ru*

[1] *Институт прикладной семиотики Академии наук Республики Татарстан, г. Казань, 420111, Россия*
[2] *Казанский федеральный университет, г. Казань, 420008, Россия*

Современные системы управления корпусными данными позволяют решать широкий спектр задач, связанных с компьютерной лингвистикой. Часто такие системы основаны на готовых решениях, что приводит к проблемам со скоростью выполнения поисковых запросов (выборок данных), гибкостью, масштабируемостью и расширяемостью системы. В данной статье рассмотрена архитектура системы корпус-менеджер, разработанной для управления корпусом татарского языка, и предложена модель корпусных данных, которая решает ряд проблем существующих систем корпус-менеджер.

В статье подробно описана архитектура программной части системы, которая позволяет быстро разрабатывать дополнительные модули и интегрировать в систему сторонние приложения. Представленная модель корпусных данных разработана для решения задач быстрого поиска по словоформам и леммам (прямого поиска), а также по морфологическим свойствам (обратного поиска). Она позволяет также осуществлять фразовый и синтаксический поиск, таким образом обеспечивая решение задач, связанных с более сложными выборками данных.

Авторами были проведены эксперименты с целью выбора наиболее оптимального набора технологий хранения данных, который позволяет решать поставленные перед системой задачи и имеет возможности для расширения ее функционала в будущем. В статье представлены результаты экспериментов, в том числе тестирования производительности различных СУБД и хранилищ данных.

Описанные архитектура системы и модель корпусных данных применены в поисковом модуле системы управления корпусом татарского языка, который показывает существенный прирост скорости выполнения поисковых запросов по сравнению с аналогичными модулями в других системах.

Ключевые слова: *корпус-менеджер, корпусные данные, архитектура системы, модель данных, поисковая система, СУБД.*

Разработка архитектуры и БД поисковой системы корпус-менеджер предполагает анализ и тестирование всего необходимого функционала различных СУБД. Функционирующая корректно достаточно быстрая и оптимальная с точки зрения потребления машинных ресурсов поисковая система практически полностью зависит от правильной архитектуры системы и БД.

Разработка системы управления татарским корпусом (http://tugantel.tatar) направлена на расширение поисковых возможностей для БД Татарского национального корпуса [1]. Исследования татарского языка в значительной степени зависят от использования корпус-менеджера татарского языка в гуманитарных и образовательных приложениях и в сфере компьютерной лингвистики.

Для разработки национальных корпус-менеджеров часто используются готовые решения, которые имеют свои плюсы и минусы. В частности, такие решения нередко являются проприетарными, соответственно, малодоступными для использования в качестве инструментария или части инструментария для работы с корпусными данными. Некоторые из них будут рассмотрены далее.

Обзор работ

Существует много работ, связанных с разработкой систем управления корпусами.

Многие поисковые системы, работающие с национальными языковыми корпусами, используют готовые технологии (движки). Например, поисковая система Яндекс.Сервер используется для Национального корпуса русского языка [2]. Подобные системы представляют собой комплекс различных подсистем, обеспечивающих быстрый и многофункциональный поиск. Яндекс.Сервер является проприетарной системой, а его полная версия распространяется на коммерческой основе. Поисковая система позволяет выполнять прямые и обратные поисковые запросы, использовать логические операции И, ИЛИ при поиске по морфологическим свойствам. Система не оптимизирована для выполнения поисковых запросов по морфологическим свойствам, так как изначально разработана для выполнения прямых поисковых запросов и поиска по леммам. Использование подобных систем требует тонкой настройки оборудования и программного окружения для достижения наилучших результатов.

Еще одной важной работой является Чешский национальный корпус [3]. Он разработан на основе системы Sketch Engine [4], которая обладает следующими возможностями: поддерживает произвольные метаданные документов, использует собственный язык запросов (CQL – Corpus Query Language), поддерживает обратный и фразовый поиск, позволяет просматривать статистику корпуса и произво-

дить выборки списков словоформ и n-грамм по параметрам. Также система может работать с документами в различных форматах и использует NoSQL – БД для их хранения [5]. Вместе с тем Sketch Engine также имеет и ряд ограничений: в поисковой выдаче не показывается морфологическая разметка, использование обратного поиска затруднительно, выполнение некоторых поисковых запросов может занимать продолжительное время, система не оптимизирована для выполнения сложных поисковых запросов, что делает ее потенциально уязвимой для атак типа DDoS.

Архитектура системы

Для программной реализации корпус-менеджера татарского языка используется концепция MVC (Model-View-Controller) [6], которая была несколько изменена для решения задач системы. Основная схема выполнения любой задачи, поставленной системе, показана на рисунке 1.

Более детализированная схема элементов системы и связей между ними (архитектура системы) показана на карте на рисунке 2. Работа системы начинается с запроса (Request), который приходит в основной контроллер (MainControl). В первую очередь, основной контроллер обеспечивает безопасность при помощи компонента ShieldModel, который, в свою очередь, использует объект очереди (Queue). Если ShieldModel считает запрос безопасным, контроль передается специальному контроллеру в зависимости от типа запрашиваемой задачи. Всего в системе семь специальных контроллеров: SinglePageControl для вывода статичных страниц, SearchControl для обработки поисковых запросов, ContextControl для расширения контекстов, SinglePageEditControl для управления статичными страницами, StatisticsControl для просмотра статистики системы, DataManagementControl для управления корпусными данными, SecurityControl для управления настройками безопасности. Первые три используют абстрактный контроллер страницы PageControl и доступны любому пользователю. PageControl объединяет в себе основной функционал для страниц с публичным доступом. Остальные контроллеры применяются для управления и используют абстрактный контроллер страниц управления AdminControl. Эти контроллеры доступны только для пользователей с правами на управление (администраторы и редакторы). AdminControl объединяет в себе основной функционал для страниц управления и наследует функционал PageControl. PageControl и AdminControl используют вспомогательный контроллер RequestControl для быстрой проверки пользовательских запросов.

После проверки и фильтрации данных система передает контроль в модель, соответствующую запрошенному действию: SinglePageModel, SearchModel, QueryModel (не вызывается непосредственно из контроллера, используется в SearchModel), ContextModel, SinglePageEditModel, StatisticsModel, DataManagementModel (использует модели, которые не показаны на карте: DocumentModel, SentenceModel, WordModel), SecurityModel. Все модели наследуют абстрактную модель PageModel, которая объединяет функционал страниц. Последние четыре модели используют модель страницы управления AdminPageModel, объединяющую функционал страниц управления. Все мо-

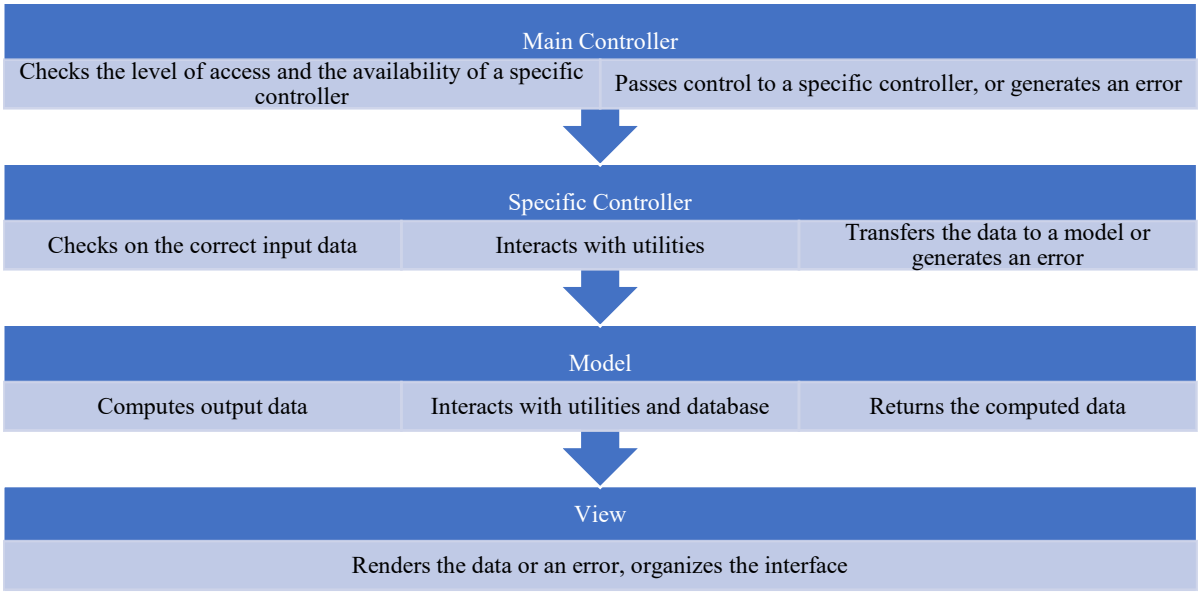

Рис. 1. Абстрактная схема выполнения задач в системе

Fig. 1. An abstract scheme for performing tasks in the system

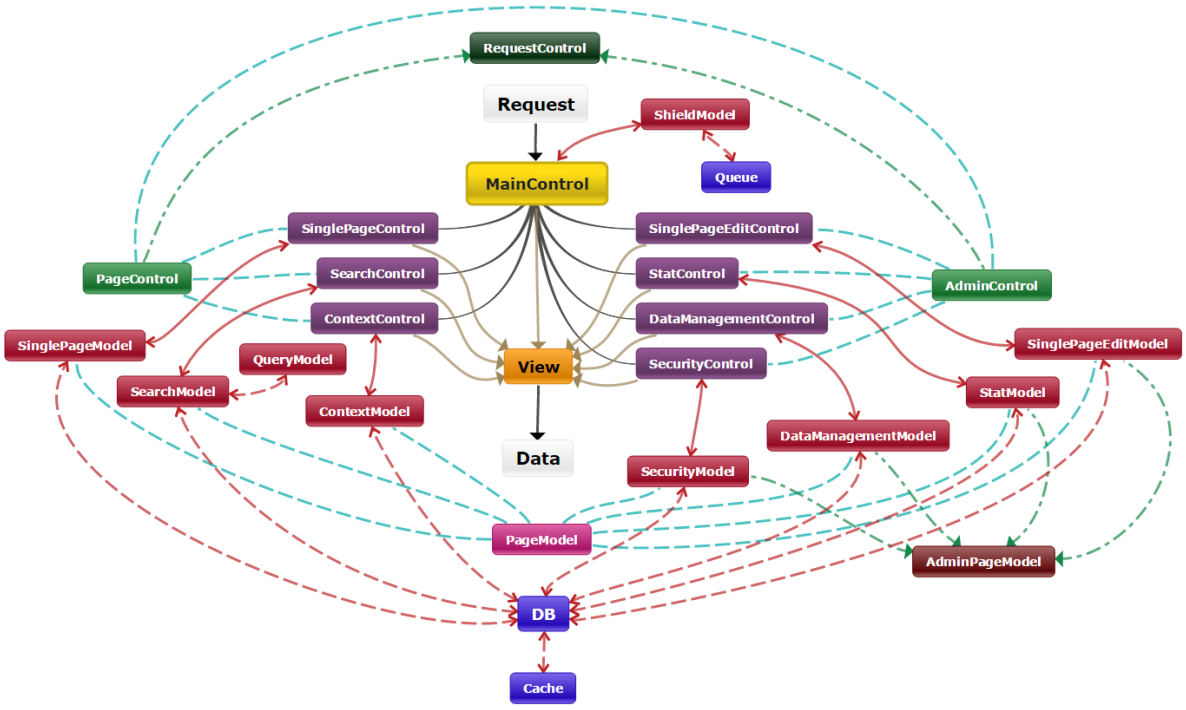

Рис. 2. Архитектура системы

Fig. 2. System architecture

дели используют модель БД DB, которая, в свою очередь, использует модель Cache для кэширования.

После выполнения моделью действия контроль возвращается к контроллеру, откуда данные передаются в Вид (View). Последний использует данные и соответствующие шаблоны страниц для генерации документа HTML или возвращает данные в формате JSON в зависимости от запрошенного формата выходных данных.

Выбор БД и хранилища данных

Поиск и анализ БД и хранилища данных. На первом этапе разработки системы важно выбрать СУБД и хранилище данных для использования в поисковой системе. Возможные для использования СУБД и хранилища данных должны обеспечивать быстрый и надежный доступ к большому объему данных в режиме реального времени и соответствовать следующим критериям:

– производительность (скорость поиска по БД, включающей таблицу с 100 млн строк, не менее 1 запроса в секунду);

– масштабируемость (соответствие требованиям к функциональности системы с распределением процессов на несколько физических машин);

– стоимость (анализ включает бесплатные и коммерческие СУБД и хранилища данных);

– совместимость с ПО (поддержка возможности работы с PHP и Unix-подобными операционными системами);

– доступность документации (доступность полной документации на русском, английском или татарском языках);

– перспективы развития (динамика разработки проекта, существующее сообщество пользователей, планы разработчиков).

Были произведены поиск и анализ СУБД и хранилищ данных, соответствующих критериям, на основе информации из открытых источников. Выбраны одиннадцать возможных СУБД и хранилищ данных:

– Memcached/MemcacheDB (http://memcached.org/, http://memcachedb.org/);
– Redis (http://redis.io/);
– Tarantool (http://tarantool.org/);
– Aerospike (http://www.aerospike.com/);
– FoundationDB (https://foundationdb.com/);
– Apache HBase (http://hbase.apache.org/);
– Hypertable (http://hypertable.com/);
– Cassandra (http://planetcassandra.org/);
– MariaDB (https://mariadb.org/);
– Sphinx (http://sphinxsearch.com/);
– ElasticSearch (http://www.elasticsearch.org/).

Для каждой из перечисленных систем была разработана структура обратного индекса с учетом их особенностей.

Тестирование производительности БД и хранилищ данных. Каждая система была установлена на виртуальную машину для проведения тестирования производительности. Виртуальная машина имела следующие технические характеристики:

4 ядра CPU (2.7GHz на каждое), 4GB DDR3 RAM, HDD 20GB (5400RPM), SSD 1GB для swap и работала под управлением Debian 7.5. Скорость записи и чтения была замерена для каждой системы хранения данных. Записываемые данные (обратный индекс документов) были сгенерированы из словоформ, случайно выбранных из словаря, морфологические свойства сгенерированы случайно для каждой словоформы.

Авторы разработали специальное ПО для генерации, записи и чтения данных, а также для замера производительности для каждой системы хранения данных. Это ПО позволило автоматизировать тестирование и обеспечить необходимую нагрузку на каждую систему. В процессе тестирования производительности были получены результаты, показанные в таблице 1.

Таблица 1

Результаты тестирования производительности

Table 1

Performance test results

Система	Запись, словоформ/сек.	Чтение, словоформ/сек.
Memcached	3283	13,2
MemcacheDB	560,7	7,8
Redis	3792	14
FoundationDB	692,2	5,8
Hypertable	2655	15,8
MySQL (MariaDB)	1031,5	9,9
Redis+MySQL (MariaDB)	551,3	14,1
Sphinx	1368,7	0,2
ElasticSearch	3546,7	11,4

Наилучшую скорость чтения показала система Hypertable. Однако для дальнейшей работы была выбрана комбинация Redis + MySQL, так как при незначительной потере в скорости эта комбинация может существенно расширить поисковый функционал только за счет изменения запроса к БД.

Архитектура БД

Типы поисковых запросов. Архитектура системы и БД разработана для того, чтобы отвечать на следующие типы запросов:

– прямой поиск по словоформе или лемме;

– обратный поиск по морфологическим свойствам, которые могут быть представлены как формула с использованием конъюнкции, дизъюнкции, отрицания, произвольного набора логических операций И, ИЛИ, НЕ;

– гибридный поиск по морфологическим свойствам и словоформе или лемме.

Размеченный текст [7] разбирается на отдельные словоформы и их морфологическую разметку для записи в БД в виде обратного индекса [8], списка предложений (контекстов) и списка документов. Поиск производится по обратному индексу, что существенно увеличивает скорость выполнения запросов к БД.

Так как морфологические свойства записаны в БД в форме бинарных векторов, поисковые запросы с конъюнкцией могут быть представлены как наложение битовой маски вектора морфологических свойств из поискового запроса на вектор морфологических свойств из таблицы обратного индекса. Результат данной операции должен быть равен вектору морфологических свойств из поискового запроса. Запросы с дизъюнкцией формируются так же, но результат должен представлять собой непустой вектор (должен содержать хотя бы одну 1). При запросе с отрицанием наложение битовой маски должно вернуть пустой вектор (все нули). Наложение битовой маски является простой операцией, и, несмотря на то, что индексы из таблицы не могут быть применены для этой операции, запросы исполняются менее чем за 1 секунду.

Структура таблицы обратного индекса показана в таблице 2.

Таблица 2

Структура таблицы обратного индекса

Table 2

The structure of the reverse index table

Имя столбца	Тип	Описание
id	INT (AUTO-INCREMENT)	Уникальный идентификатор морфологического анализа
word	MEDIUMINT UNSIGNED	Уникальный идентификатор словоформы. Связи между идентификаторами и словоформами хранятся в Redis
lemma	MEDIUMINT UNSIGNED	Уникальный идентификатор леммы. Связи между идентификаторами и леммами хранятся в Redis
doc	MEDIUMINT UNSIGNED	Уникальный идентификатор документа. Документы хранятся в отдельной таблице
pos	MEDIUMINT UNSIGNED	Позиция словоформы в документе
sentence	INT UNSIGNED	Уникальный идентификатор контекста. Контексты хранятся в отдельной таблице
sentence_pos	SMALLINT UNSIGNED	Позиция словоформы в контексте
morph1	BIGINT UNSIGNED	Первые 64 морфологических свойства
morph2	BIGINT UNSIGNED	Последующие 64 (до 64) морфологических свойства

Таблица обратного индекса также содержит несколько индексов для оптимизации запросов (табл. 3).

Таблица 3

Индексы в таблице обратного индекса

Table 3

Indeces in the reverse index table

Название	Тип и компоненты	Описание
PRIMARY	PRIMARY KEY (id, word, lemma, morph1)	Составной первичный ключ для быстрого поиска словоформы и для организации секционирования
docs	KEY (doc)	Индекс по идентификаторам документов для быстрого поиска в документе и для группировки по документам
sentences	KEY (sentence)	Индекс по идентификаторам контекстов для быстрой группировки по контекстам и для нескольких запросов в одном контексте (фразовый поиск)
context	KEY (id, doc, sentence)	Составной индекс для расширения контекстов
word	KEY (word)	Индекс по идентификаторам словоформ для быстрого прямого и гибридного поиска по словоформе
lemma	KEY (lemma)	Индекс по идентификаторам лемм для быстрого прямого и гибридного поиска по лемме

Также в таблице обратного индекса были созданы секции и подсекции [9], которые существенно увеличили скорость выполнения запросов. Первая часть вектора морфологических свойств (класс) определяет, в какую секцию должен быть записан разбор, а идентификаторы словоформ и лемм определяют подсекцию.

Секции p1–p14 определены на основе морфологического свойства из первой части (класса); секция class_not_found включает все разборы, у которых не определено морфологическое свойство в первой части вектора; секция p0 включает словоформы без разбора; знаки препинания в конце предложений (точки, восклицательные знаки, вопросительные знаки) хранятся в секции end; секция syntax включает все остальные знаки препинания; слова, состоящие из одной буквы, хранятся в секции letter; в секции latin хранятся все словоформы, которые содержат латинские символы; все другие сущности, включая специальные символы, хранятся в секции sign.

Таким образом, таблица размером 12,5 ГБ была разделена на небольшие таблицы, каждая из которых может разместиться в оперативной памяти для быстрого доступа к данным. Это существенно увеличило скорость выполнения запросов к БД. При накоплении достаточной истории запросов секции начинают заполнять оперативную память, в которой хранятся только наиболее востребованные из них.

Заключение

Архитектура системы и БД, описанная в данной статье, применена в поисковом модуле системы управления корпусными данными, работающем с электронным корпусом татарского языка.

Для проверки правильности, точности, согласованности и характера изменения во времени был проведен комплекс мероприятий для тестирования. Тестирование корректности показало, что представленный метод позволяет полностью выполнить поставленные задачи. Тестирование точности, основанное на сравнении с другими методами представления и обработки запросов, показало корректность работы представленных методов.

Тестирование согласованности и характера изменения во времени показало, что описанный синтаксис лексической и морфологической составляющих запроса был интерпретирован системой корректно, а время, необходимое для обработки и выполнения поискового запроса, не превышает 0,05 секунды в 98,71 % случаев для прямого поиска по словоформам и в 98,08 % случаев для гибридного поиска по словоформам и морфологическим свойствам.

Использование описанной архитектуры для корпус-менеджеров позволяет решить целый ряд проблем. В данной статье описаны лишь некоторые из них. В будущем планируется использовать эту архитектуру для интеграции различных сервисов, направленных на обработку лингвистических данных, таких как морфологический анализатор, модуль разрешения морфологической многозначности [10] и других.

Представленный подход к решению проблем поисковых систем для лингвистических корпусов позволяет использовать разработанную систему не только для электронного корпуса текстов на татарском языке, но и для корпусов других языков без существенных изменений в системе.

Литература

1. Suleymanov D., Nevzorova O., Gatiatullin A., Gilmullin R., Khakimov B. National corpus of the Tatar language “Tugan Tel”: grammatical annotation and implementation. Procedia-Social and Behavioral Sciences, 2013, vol. 95, pp. 68–74. DOI: 10.1016/j.sbspro.2013.10.623.

2. Аброскин А.А. Поиск по корпусу: проблемы и методы их решения. Национальный корпус русского языка: 2006–2008. Новые результаты и перспективы. СПб: Нестор-История, 2009. С. 277–282.

3. Křen M. Recent Developments in the Czech National Corpus. Proc. 3rd Workshop on CMLC-3. Mannheim, 2015, URL: https://ids-pub.bsz-bw.de/frontdoor/deliver/index/docId/3826/file/K%c5%99en_Recent_developments_in_the_czech_national_corpus_2015.pdf/ (дата обращения: 21.07.2018).

4. Kilgarriff A., Baisa V., Bušta J., Jakubíček M., Kovář V., Michelfeit J., Suchomel V. The Sketch Engine: ten years on. Lexicography, 2014, no. 1, pp. 7–36. DOI: 10.1093/ijl/ecw029.

5. Han J., Haihong E., Le G., & Du J. Survey on NoSQL database. Pervasive computing and applications, Proc. 6th IEEE Intern. Conf. ICPCA., 2011, pp. 363–366. DOI: 10.1007/978-981-10-6520-0_14.

6. Leff A., Rayfield J.T. Web-application development using the model/view/controller design pattern. Proc. 5th IEEE Intern. Conf. EDOC'01, 2001, pp. 118–127. DOI: 10.1109/EDOC.2001.950428.

7. Галиева А.М., Хакимов Б.Э., Гатиатуллин А.Р. Метаязык описания структуры татарской словоформы для корпусной грамматической аннотации // Ученые записки Казанского ун-та: Гуманитарные науки. 2013. № 5. Т. 155. С. 287–296.

8. Croft W.B., Metzler D., Strohman T. Search engines: Information retrieval in practice. Addison-Wesley, 2010, pp. 131–141. DOI: 10.4236/jsea.2016.97024.

9. MySQL 5.6 Reference Manual. URL: http://dev.mysql.com/doc/refman/5.6/en/partitioning.html (дата обращения: 19.09.2018).

10. Хакимов Б.Э., Гильмуллин Р.А., Гатауллин Р.Р. Разрешение грамматической многозначности в корпусе татарского языка // Ученые записки Казанского ун-та: Гуманитарные науки. 2014. Т. 156. № 5. С. 236–244.

Software & Systems Received 21.09.18
DOI: 10.15827/0236-235X.124.653-658 2018, vol. 31, no. 4, pp. 653–658

A corpus manager system: corpus data architecture and models

D.R. Mukhamedshin [1], *Postgraduate Student, damirmuh@gmail.com*
D.Sh. Sulejmanov [1, 2], *Dr.Sc. (Engineering), Professor, ipsanrt@mail.ru*

[1] *Institute of Applied Semiotics of the Tatarstan Academy of Sciences, Kazan, 420111, Russian Federation*
[2] *Kazan Federal University, Kazan, 420008, Russian Federation*

Abstract. Modern corpus data management systems allow solving a wide range of computer linguistics problems. Such systems are often based on ready-made solutions, which causes the problems with search query execution speed (data extraction), flexibility, scalability and extensibility of the system. This article considers the architecture of the corpus manager system for the Tatar Corpus. It also proposes a corpus data model that solves a number of problems of the existing corpus manager systems.

The paper describes in detail the architecture of the software part of the system, which allows quick developing of additional modules and integrating third-party applications into the system. The corpus data model presented in the article is designed to solve quick search tasks using wordforms and lemmas (direct search), as well as morphological properties (reverse search). The developed model also allows phrasal and syntactic search, thus providing a solution for tasks related to more complex data extraction.

The authors performed experiments in order to find the most optimal set of data storage technologies that allows solving the required problems and has the capability to expand the system functionality in the future. The paper presents the results of experiments, including the results of testing the performance of various CAD systems and data storages.

The search module of the corpus manager system of the Tatar language corpus includes the described system architecture and the corpus data model. It shows a significant increase in the execution speed of search queries in comparison with similar modules in other systems.

Keywords: corpus manager, corpus data, system architecture, data model, search engine, DBMS.

References

1. Suleymanov D., Nevzorova O., Gatiatullin A., Gilmullin R., Khakimov B. National corpus of the Tatar language “Tugan Tel”: grammatical annotation and implementation. *Procedia-Social and Behavioral Sciences*. 2013, no. 95, pp. 68–74. DOI: 10.1016/j.sbspro.2013.10.623.

2. Abroskin A.A. Search by body: problems and methods to solve them. *Russian National Corpus: 2006–2008. New Results and Prospects*. St. Petersburg, Nestor-Istoriya Publ., 2009, pp. 277–282 (in Russ.).

3. Křen M. *Recent Developments in the Czech National Corpus*. Proc. 3rd Workshop on CMLC-3. Mannheim, 2015. Available at: https://ids-pub.bsz-bw.de/frontdoor/deliver/index/docId/3826/file/K%c5%99en_Recent_developments_in_the_czech_national_corpus_2015.pdf/ (accessed July 21, 2018).

4. Kilgarriff A., Baisa V., Bušta J., Jakubíček M., Kovář V., Michelfeit J., Suchomel V. The Sketch Engine: ten years on. *Lexicography*. 2014, no. 1, pp. 7–36. DOI: 10.1093/ijl/ecw029.

5. Han J., Haihong E., Le G., Du J. Survey on NoSQL database. *IEEE 6th Intern. Conf. on Pervasive Computing and Applications (ICPCA)*. 2011, pp. 363–366. DOI: 10.1007/978-981-10-6520-0_14.

6. Leff A., Rayfield J.T. Web-application development using the model/view/controller design pattern. *Proc. 5th IEEE Intern. IEEE Enterprise Distributed Object Computing Conf., EDOC'01*. 2001, pp. 118–127. DOI: 10.1109/EDOC.2001.950428.

7. Galieva A.M., Khakimov B.E., Gatiatullin A.R. Meta-language describing the structure of Tatar wordform for corpus grammatical annotation. *Proc. of Kazan Univ. Humanities Series*. 2013, vol. 155, no. 5, pp. 287–296 (in Russ.).

8. Croft W.B., Metzler D., Strohman T. *Search Engines: Information Retrieval in Practice*. Addison-Wesley Publ., 2010, pp. 131–141. DOI: 10.4236/jsea.2016.97024.

9. *MySQL 5.6 Reference Manual*. Available at: http://dev.mysql.com/doc/refman/5.6/en/partitioning.html (accessed September 19, 2018).

10. Khakimov B.E., Gilmullin R.A., Gataullin R.R. Resolution of grammatical ambiguity in the Tatar language corpus. *Proc. of Kazan Univ. Humanities Series*. 2014, vol. 156, no. 5, pp. 236–244.