**ТИТУЛ**

**ЗАДАНИЕ**

**РЕФЕРАТ**

Расчётно-пояснительная записка 66 с., X рис., X листингов., X источников, 3 прил.

АНАЛИТИЧЕСКАЯ СИСТЕМА, БАЗЫ ДАННЫХ, KIBANA, ELASTICSEARCH, NEO4J, HADOOP, HDFS, SPARK.

Целью курсового проекта является разработать макет аналитической

системы для редакции журналов с использованием ElasticSearch, Neo4j, Hadoop+Spark.

В ходе выполнения курсового проекта выполнялись следующие задачи:

− развертывание виртуальной машины;

− установка и настройка требуемых программных компонентов;

− генерация JSON-документов для заполнения;

− разработка и выполнение запросов к данным;

− анализ запросов посредством монитора.

**СОДЕРЖАНИЕ**

[Определения, обозначения и сокращения 5](#_Toc1)

[Введение 6](#_Toc2)

[1 Задание и описание варианта КП 7](#_Toc3)

[2 Установка пакетов Elasticsearch, Neo4j, Hadoop+Spark 9](#_Toc4)

[2.1 Установка ElasticSearch 9](#_Toc5)

[2.2 Установка Neo4j 10](#_Toc6)

[2.3 Установка Hadoop+Spark 11](#_Toc7)

[3 Elasticsearch 13](#_Toc8)

[3.1 Индексация документов 13](#_Toc9)

[3.2 Запросы 19](#_Toc10)

[4 Neo4j 22](#_Toc11)

[4.1 Создание и заполнение графовой БД 22](#_Toc12)

[4.2 Запрос 24](#_Toc13)

[5 Spark 25](#_Toc14)

[5.1 Создание и заполнение таблиц 25](#_Toc15)

[5.2 Запрос 26](#_Toc16)

[5.3 Монитор 27](#_Toc17)

[Заключение 33](#_Toc18)

[Список использованных источников 34](#_Toc19)

[ПРИЛОЖЕНИЕ А 35](#_Toc20)

[ПРИЛОЖЕНИЕ Б 45](#_Toc21)

[ПРИЛОЖЕНИЕ В 55](#_Toc22)

# **Определения, обозначения и сокращения**

В настоящей расчетно-пояснительной записке применяются следующие определения, сокращения и обозначения:

БД – база данных (сокр.)

ПО – программное обеспечение (сокр.)

СУБД – система управления базами данных (сокр.)

Стемминг – это процесс приведения слова к его корневой форме

Токенизация — процесс разбиения неструктурированного текста на более мелкие, доступные для поиска фрагменты, называемые токенами

ACID – набор требований к транзакциям в СУБД: атомарность, согласованность, изолированность, долговечность

DAG (англ. Directed Acyclic Graph) – направленный ациклический граф (сокр.)

ES – ElasticSearch (сокр.)

HDFS (Hadoop Distributed File System) – распределенная файловая система Hadoop для хранения файлов (сокр.)

YARN (Yet Another Resource Negotiator) – система планирования заданий и управления кластером (сокр.)

# **Введение**

Целью курсового проекта является разработка макета аналитической системы в предметной области “Ремонт бытовой техники” на основе баз данных NoSQL Elasticsearch и Neo4j в связке с системами Hadoop и Spark.

Макет аналитической системы состоит из заполненных данными БД Elasticsearch и графовой БД Neo4j, csv-файлов, сохраненных в распределенной файловой системе HDFS.

Elasticsearch — поисковая система, основанная на библиотеке Lucene. Предоставляет распределенную многопользовательскую полнотекстовую поисковую систему с веб-интерфейсом для работы с документами JSON без схем. Обеспечивает горизонтально масштабируемый поиск, поддерживает многопоточность.

Neo4j — это система управления графовой базой данных. Разработчики описывают ее как транзакционную базу данных, совместимую с ACID, с собственной системой хранения и обработки графов.

Apache Spark — фреймворк с открытым исходным кодом для реализации распределённой обработки неструктурированных и слабоструктурированных данных, входящий в экосистему проектов Hadoop.

# **1 Задание и описание варианта КП**

В рамках выполнения курсовой работы необходимо было выполнить следующие задачи:

1. Установить виртуальную машину с ubuntu 20.04.3 (ubuntu-20.04.3-desktop-amd64.iso [1б]) в VirtualBox [1а] (python 3 уже предустановлен в ubuntu 20.04.3).

Имя: Ubuntu-01. Тип: Linux. Версия: Ubuntu (64 bit) и т.д. Объём ОП – 4ГБ, Диск – 20ГБ.

2. Установить Elasticsearch, Neo4j, Hadoop+Spark, как это Вы делали на лабораторных работах в 1-м семестре [2а]. Можете использовать виртуальное окружение Python [1с] (в будущем может очень пригодиться).

3. В рамках основной части:

- вручную (или автоматически) создать два json-файла c 20¸30 json-документами каждого типа для предметной области, указанной в Вашем варианте; в варианте для каждого документа указаны его поля,

- в Elasticsearch: создать индекс с анализатором и маппингом [3], проиндексировать json-документы, разработать запросы с вложенной агрегацией [4], представить результаты в среде Kibana [5],

- в Neo4j: по данным из Elasticsearch заполнить графовую базу данных, разработать и реализовать запрос к этой БД,

- в Spark: по данным из Elasticsearch сформировать csv-файлы c таблицами и сохранить их в файловой системе HDFS, написать запрос и реализовать его в Spark, проанализировать процесс выполнения запроса с использованием монитора.

Вариант 1 описывает предметную область “Ремонт бытовой техники” и включает следующие блоки заданий:

**Elasticsearch.**

1. Типы документов (json):

Заказ:

{index, doc\_type, id, body: {id\_заказа, дата\_заказа, id\_заказчика, сведения\_о\_заказчике\*, данные\_о\_заказе\*, срок\_выполнения\_заказа, фактическая\_дата\_выполнения, [запчасть\*], [ремонт\*], стоимость, id\_мастера}}

Мастер:

{index, doc\_type, id, body: {сведения\_о\_мастере\*, [отзыв\_о\_работе\*]}}

1. Требование к анализатору:

Поля, отмеченные \*, разделить на слова, убрать пунктуацию с помощью токенизатора standart (русский), перевести все токены в нижний регистр, убрать токены, находящиеся в списке стоп-слов, выполнить стемминг оставшихся токенов с помощью фильтра snowball.

1. Запросы с вложенной агрегацией:

- разбить заказы по дате заказа с периодом 1 месяц, для каждой «корзины» определить суммарное число заказов по каждой запчасти;

- вывести мастеров, в сведениях которых указан стаж работы.

**Neo4j.**

1. По данным из Elasticsearch заполнить графовую базу данных Заказ(id\_заказа, дата\_заказа, сведения\_о\_заказчике, стоимость\_заказа) - Выполнил(срок\_выполнения\_заказа, фактическая\_дата\_выполнения) - Мастер(id\_мастера, сведения\_о\_мастере).
2. Разработать и реализовать запрос: найти мастера, который выполнил максимальное количество заказов.

**Spark.**

1. По данным из Elasticsearch сформировать csv-файлы (с внутренней схемой) таблиц «Заказчик», «Заказ», «Мастер» и сохранить их в файловой системе HDFS.
2. Написать запрос select: найти заказы и мастеров, которые не выполнили заказы в срок.
3. Реализовать этот запрос в Spark. Построить временную диаграмму его выполнения по результатам работы монитора.

# **2 Установка пакетов Elasticsearch, Neo4j, Hadoop+Spark**

## **2.1 Установка ElasticSearch**

Так, как в наличии имелся ПК с ОС Ubuntu, установка виртуальной машины не понадобилась и курсовая работа выполнялась на основной операционной системе.

Изматериалов к курсовой работе был скачан архив с исполняемым файлом и распакован командной “tar -xzf elasticsearch-7.17.0-linux-x86\_64.tar.gz”.

Далее, ElasticSearch был запущен из терминала (рисунок Х).

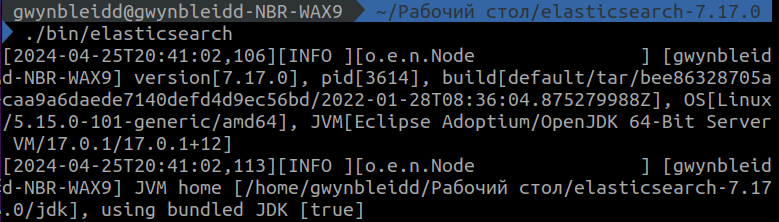


Рисунок Х — запуск ElasticSearch

Далее, доступность ElasticSeacrh была проверена с помощью GET-запроса к порту 9200 (рисунок Х).

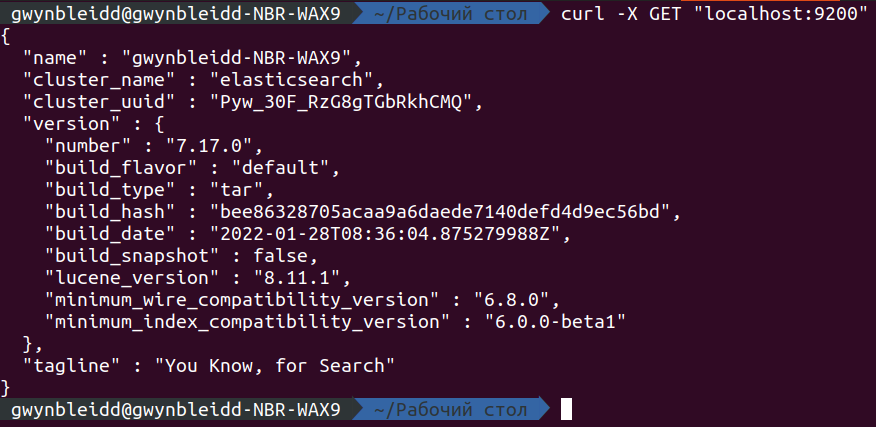


Рисунок Х — проверка доступа к ElasticSearch

Таким образом, был установлен и настроен Elasticsearch версии 7.17.0.

## **2.2 Установка Neo4j**

Далее было необходимо установить Neo4j. Для этого был импортирован открытый ключ GPG для Neo4j — “wget -O - https://debian.neo4j.org/neotechnology.gpg.key | sudo apt-key add –”

Затем Neo4j был добавлен в список пакетов менеджера apt — “echo "deb http://debian.neo4j.org/repo stable/" | sudo tee -a /etc/apt/sources.list.d/neo4j.list”.

Затем пакет был утсановлен — “sudo apt install neo4j=1:3.5.14” и запущен через systemctl (рисунок Х).

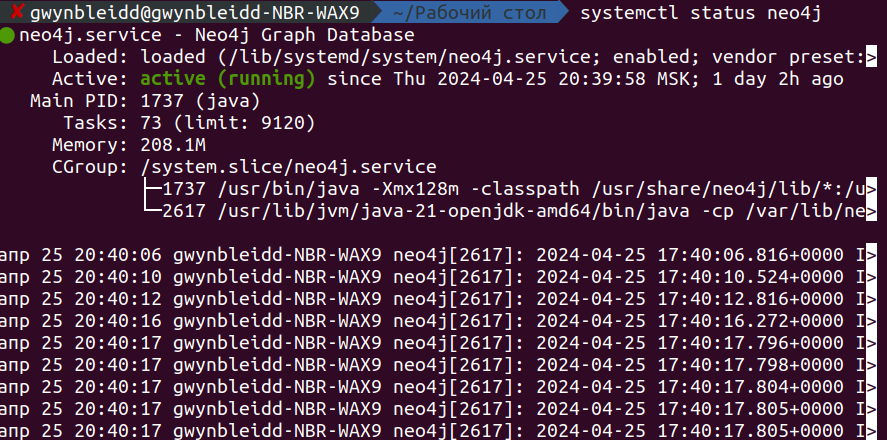


Рисунок Х — статус серписа Neo4j

Для проверки доступности Neo4j было выполнено подключение к его веб-интерфейсу (рисунок Х).

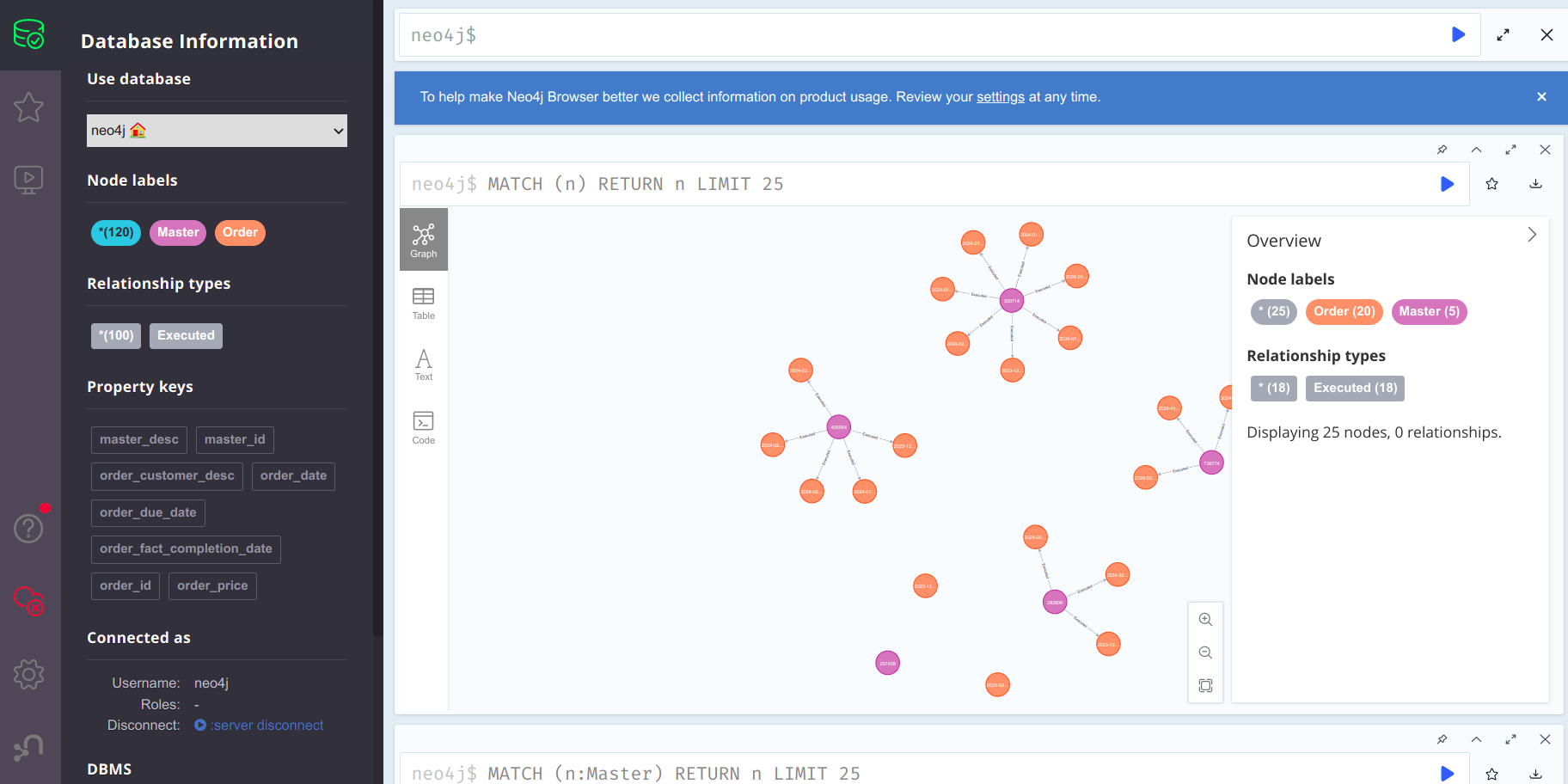


Рисунок Х — веб-интерфейс Neo4j

## **2.3 Установка Hadoop+Spark**

Для установки hadoop выполним ряд команд, приведенных в листинге Х.

Листинг Х — установка Hadoop

|  |
| --- |
| **# установка ssh**  sudo apt install openssh-server openssh-client -y  **# создание пользователя**  sudo adduser hduser  su - hduser  **# настройка ssh**  ssh-keygen -t rsa -P '' -f ~/.ssh/id\_rsa  cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys  chmod 600 ~/.ssh/authorized\_keys  chmod 700 ~/.ssh  **# загрузка hadoop**  wget <https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz>  tar xzf hadoop-3.3.6.tar.gz  sudo mv hadoop-3.3.6 /usr/local/hadoop  sudo chown -R hduser:hduser /usr/local/hadoop  **# настройка hadoop**  echo 'export JAVA\_HOME=$(readlink -f /usr/bin/java | sed "s:bin/java::")' | sudo tee -a /usr/local/hadoop/etc/hadoop/hadoop-env.sh |

Далее необходимо изменить ряд файлов конфигурации hadoop в соответствии с рисунками Х-Х.

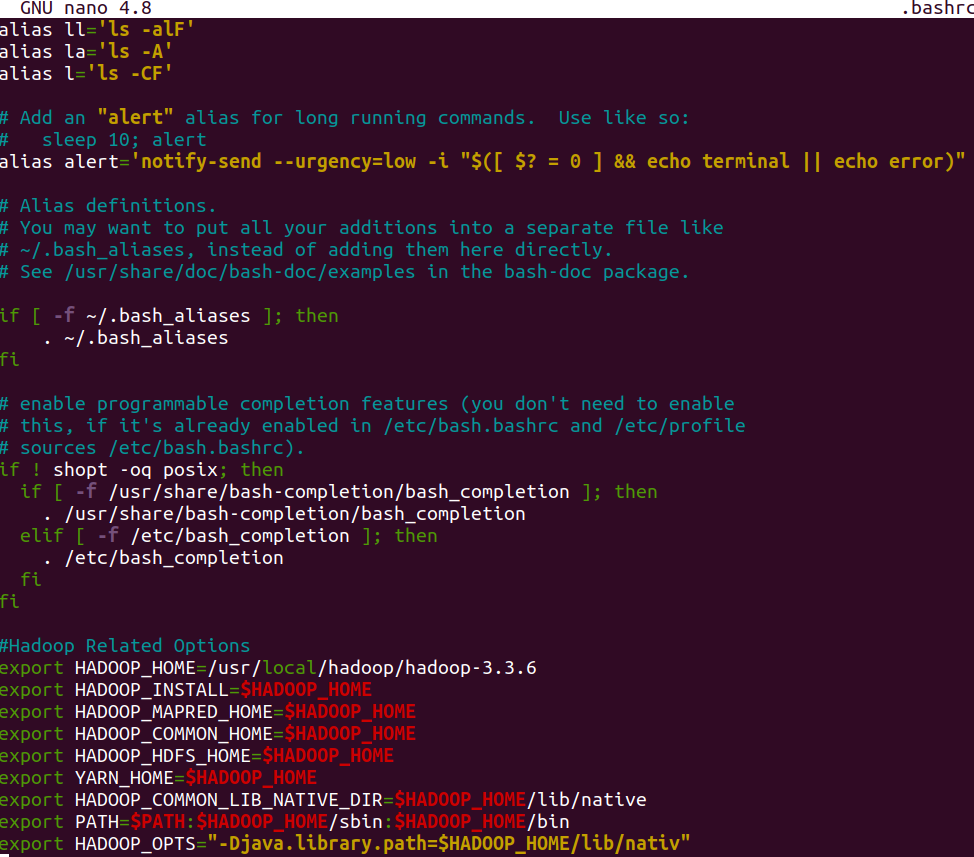


Рисунок Х — добавленные параметры .bashrc

Настроим доступ к hdfs через порт 9000.

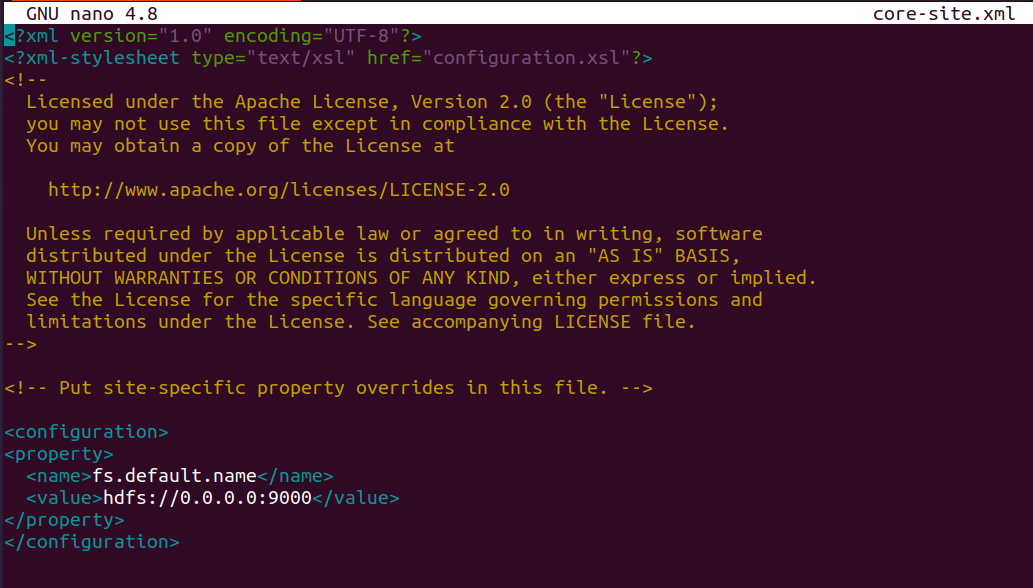


Рисунок Х — конфигурация core-site.xml

Настроим рабочие директории hadoop.

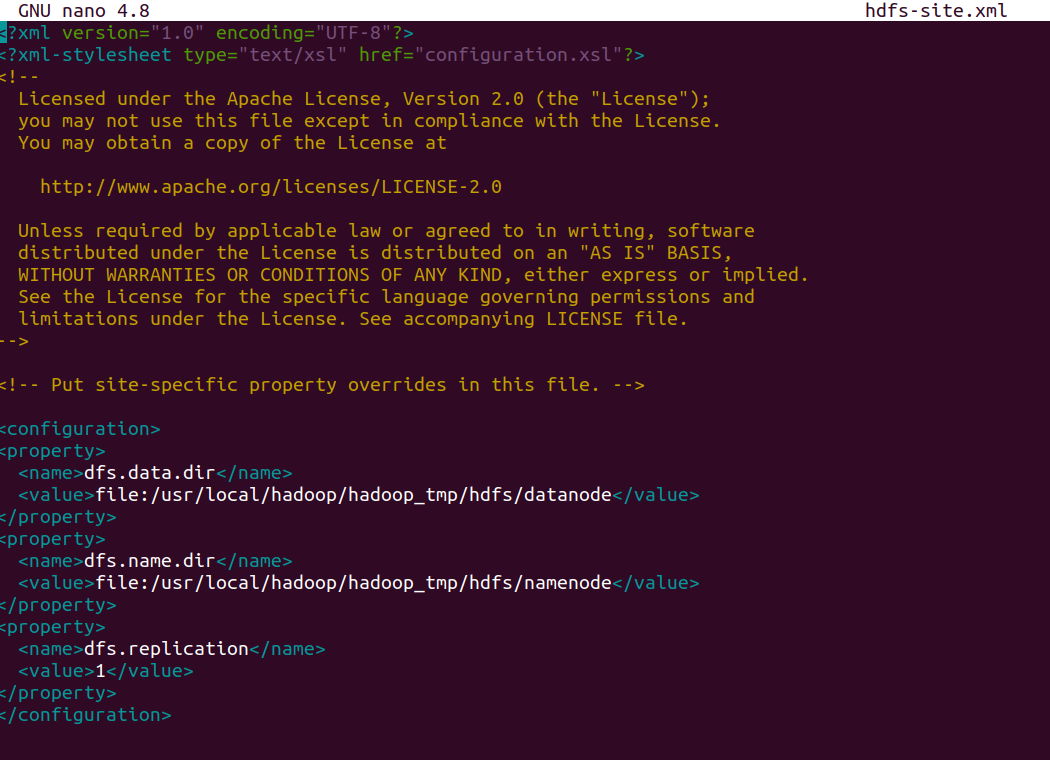


Рисунок Х — конфигурация hdfs-site.xml

Указываем, какие дополнительные сервисы должны быть запущены на каждом узле NodeManager в YARN. В данном случае сервис mapreduce\_shuffle, который отвечает за передачу данных между задачами Map и Reduce.



Рисунок Х — конфигурация yarn-site.xml

Указываем, какая фреймворк-среда будет использоваться для выполнения задач MapReduce в Hadoop. В данном случае, устанавливаем значение YARN.

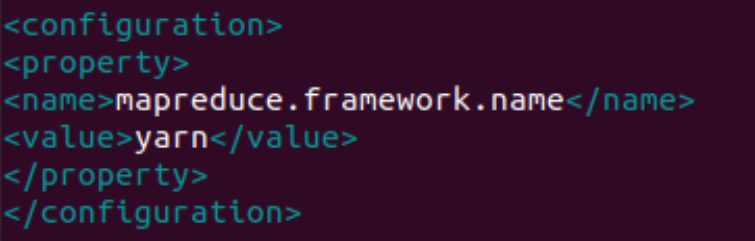


Рисунок Х — конфигурация mapred-site.xml

# **3 Elasticsearch**

## **3.1 Индексация документов**

Прежде чем выполнить индексацию документов было необходимо сгенерировать исходные данные. Для этого была использована python-библиотека Faker, позволяющая генерировать ФИО, тексты, даты и другие данные с заданными параметрами [ссылка]**.** В результате были сгенерированы данные со структурами, приведенными ниже**.**

Структура для записи о мастере:

* master\_id — уникальный номер мастера (пример: 163543);
* master\_desc — описание мастера — ФИО и данные об образовании/опыте (Пример: “Максимильян Виленович Гурьев, Стаж Работы: 11 Л./Г..”);
* master\_feedbacks — массив отзывов о мастере (пример: [“медлительный, аккуратный.”, “медлительный, ворчливый.”]).

Структура записи о заказе:

* order\_id — уникальный номер заказа (пример: 405064);
* order\_date — дата получения заказа (пример: 2024-03-15);
* order\_customer\_id — уникальный номер заказчика (пример: 257458);
* order\_customer\_desc — описание заказчика — ФИО и доп. информация (пример: “Имя: Комарова Дарья Борисовна; постоянный, премиальный.”);
* order\_details\_desc — свойства заказа (пример: “хрупкий, тяжелый.”);
* order\_due\_date — срок выполнения заказа (пример: 2024-04-20);
* order\_fact\_completion\_date — фактическая дата исполнения заказа (пример: 2024-04-25);
* order\_parts — массив со списком необходимых запчастей (пример: [“дисплей”, “usb-разъем”]);
* repair\_types — тип работ (пример: “мелкие детали”, “пайка”);
* order\_price — стоимость заказа (пример: 3866.44);
* order\_master\_id — уникальный номер мастера-исполнителя (пример: 163543)**.**

В приложении А приведен пример сгенерированного файла, в приложении Б представлен исходный код скрипта генерации.

Хотя Elasticsearch и является безсхемной базой данных, в ней все равно можно задать структуру документов, что особенно полезно для работы с большими объемами разнообразных данных. Для этого, перед загрузкой, были определены маппинги, описывающие сгенерированные документы.

Маппинг для мастеров приведен в листинге Х.

Листинг Х — маппинг для мастеров

|  |
| --- |
| mappings\_master = {  "mappings" : {  "properties" : {  "master\_desc" : { **# название поля**  "type" : "text", **# тип поля**  "analyzer": "custom\_analyzer" **# анализатор**  },  "master\_feedbacks" : {  "type" : "text",  "fields" : { **# поле-массив**  "keyword" : {  "type" : "keyword"  }  }  ,"analyzer": "custom\_analyzer"  },  "master\_id" : {  "type" : "long"  }  }  }  } |

Аналогично задан маппинг для заказов (листинг Х).

Листинг Х — маппинг заказов

|  |
| --- |
| mappings\_order = {  "mappings" : {  "properties" : {  "order\_customer\_desc" : {  "type" : "text",  "analyzer": "custom\_analyzer"  },  "order\_customer\_id" : {  "type" : "long"  },  "order\_date" : {  "type" : "date"  },  "order\_details\_desc" : {  "type" : "text",  "analyzer": "custom\_analyzer"  },  "order\_due\_date" : {  "type" : "date"  },  "order\_fact\_completion\_date" : {  "type" : "date"  },  "order\_id" : {  "type" : "long"  },  "order\_master\_id" : {  "type" : "long"  },  "order\_parts" : {  "type" : "text",  "fielddata" : True,  "fields" : {  "keyword" : {  "type" : "keyword"  }  }  ,"analyzer": "custom\_analyzer"  },  "order\_price" : {  "type" : "float"  },  "repair\_types" : {  "type" : "text",  "fields" : {  "keyword" : {  "type" : "keyword"  }  }  ,"analyzer": "custom\_analyzer"  }  }  }  } |

Для к составным полям, таким как, например, repair\_types, можно обращаться как к единому целому, применяя анализатор, либо же обращаться к их элементам через keyword.

В маппинге в том числе указывается к каким полям необходимо применить анализатор – алгоритм обработки текста, преобразующий его в формат, с которым поиск и агрегация работают намного быстрее, чем с исходным текстом.

Анализатор работает в три этапа:

* фильтрация – добавление, изменение или удаление символов по заданным правилам;
* токенизация – разбиение текста на отдельные слова;
* фильтрация токенов – фильтрация на уровне токенов. Например, производится стемминг – процесс нахождения основы слова для заданного исходного слова.

Конфигурация анализатора, использованного в ходе выполнения данной работы приведена в листинге Х.

Листинг Х — конфигурация анализатора

|  |
| --- |
| analyzer\_settings = {  "settings": {  "analysis": {  "filter": { **# описание фильтров**  "ru\_stop": {  "type": "stop", **# удалять стоп-слова**  "stopwords": "\_russian\_" **# из списка \_russian\_**  },  "snow\_ru\_stemmer": {  "type": "snowball", **# тип стеммера (ищет основы слов)**  "language": "russian" **# для русского языка**  }  },  "analyzer": {  "custom\_analyzer": {  "type": "custom",  "tokenizer": "standard", **# стандартный токенизатор**  "filter": [  "lowercase", **# переводить все в нижний регистр**  "ru\_stop",  "snow\_ru\_stemmer"  ]  }  }  }  }  } |

Приведенные выше маппинги и конфигурация анализатора были использованы в программе индексации, схема алгоритма которой приведена на рисунке Х, а исходный код — в приложении Б.



Рисунок Х — программа индексации документов

## **3.2 Запросы**

Первый запрос должен разбить заказы по дате заказа с периодом 1 месяц, для каждой «корзины» определить суммарное число заказов по каждой запчасти.

Итоговый запрос приведен в листинге Х.

Листинг Х — Запрос №1

|  |
| --- |
| GET order/\_search **# поиск по заказам**  {  **# внешняя агрегация**  "aggs": {  **# название внешней агрегации**  "over\_months": {  **# тип агрегации date histogram**  "date\_histogram": {  "field": "order\_date", **# по полю order date**  "calendar\_interval": "month", **# по месяцам**  "format": "yyyy-MM-dd" **# формат дат**  },  **# внутренняя агрегация**  "aggs":{  **# название внутренней агрегации**  "over\_parts": {  "terms": {  **# по полю order parts**  "field": "order\_parts.keyword"  }  }  }  }  }  } |

Таким образом, внутренняя агрегация выполняет подсчет повторений для каждого типа запчастей, внешняя — организует внутреннюю агрегацию по датам заказа (с округлением до месяцев). Было использовано поле keyword, т.к. без него анализатор разобьет названия некоторых типов запчастей (например, “usb-разъем”)на несколько частей.

Второй запрос должен вывести информацию о мастерах, в сведениях о которых указан стаж работы. Данный запрос приведен в листинге Х.

Листинг Х — запрос №2

|  |
| --- |
| GET master/\_search **# поиск по мастерам**  {  "query": {  "match": { **# поиск записей с вхождением подстроки**  "master\_desc": "стаж" **# "стаж" в поле master\_desc**  }  }  } |

Данный запрос позволяет выполнить поиск записей с вхождением заданной подстроки. Исходный текст документа проходит через анализатор перед выполнением сравнения.

Результаты выполнения запросов приведены в приложении А.

Кроме того, результаты запросов были визуализированы с помощью ПО Kibana (рисунки Х-Х)**.**

Так как второй запрос подразумевает текстовый результат, вместо него было визуализировано соотношение мастеров с указанием стажа и без указания стажа.



Рисунок Х — визуализация запроса №1

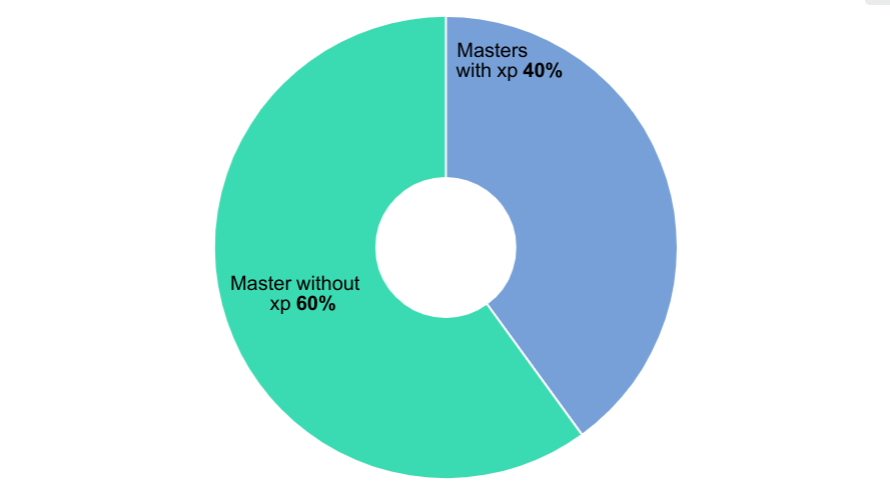


Рисунок Х — визуализация запроса №2

# **4 Neo4j**

## **4.1 Создание и заполнение графовой БД**

Согласно заданию необходимо по данным из Elasticsearch заполнить графовую базу данных узлами вида Заказ(id\_заказа, дата\_заказа, сведения\_о\_заказчике, стоимость\_заказа), Мастер(id\_мастера, сведения\_о\_мастере). А также, внести в Neo4j отношения вида Выполнил(срок\_выполнения\_заказа, фактическая\_дата\_выполнения).



Рисунок Х — схема алгоритма заполнения графовой базы данных

Для заполнения графовой базы данных был разработан алгоритм, схема которого приведена на рисунке Х.

Согласно данному алгоритму сначала выполняется подключение к ES, Neo4j и очистка Neo4j. Затем выполняется запрос, получающий все записи из индекса мастеров.

Для каждого мастера создается узел, затем во вложенном цикле производится поиск соответствующих мастеру заказов и создание узлов заказов и отношений между мастером и его заказами.

Перейдя в веб-интерфейс Neo4j можно увидеть визуализацию созданного графа (рисунок Х).

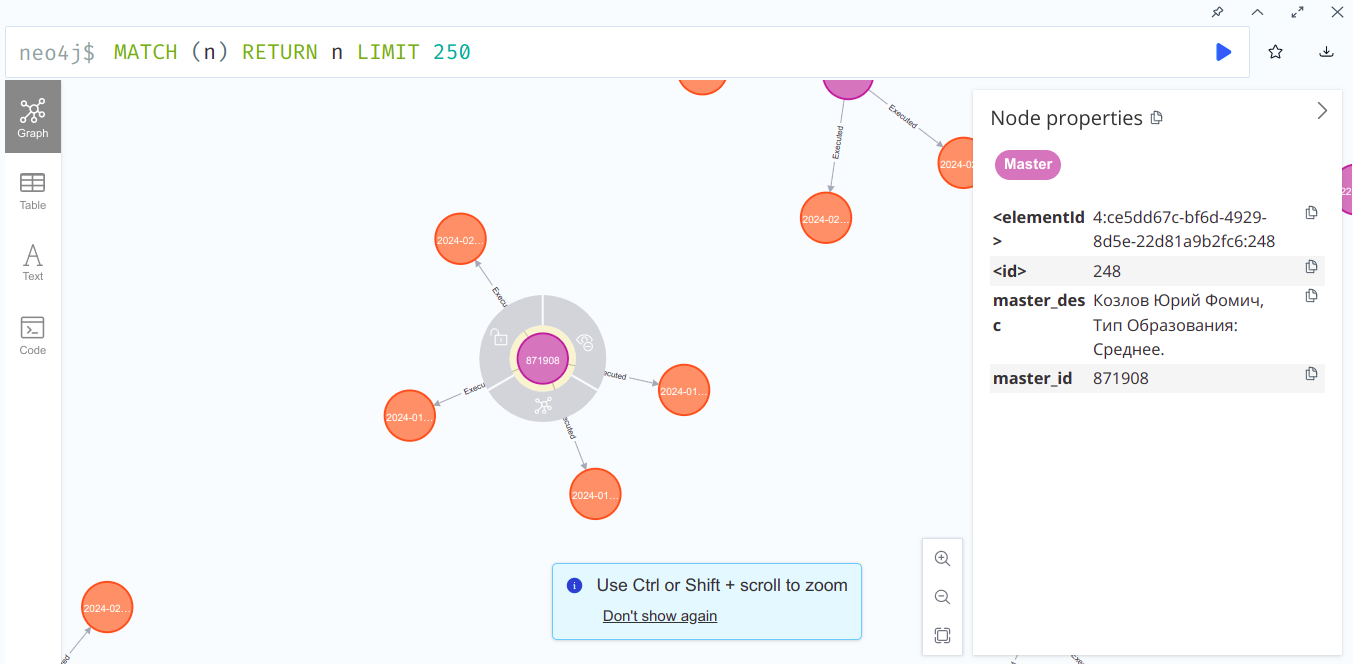


Рисунок Х — визуализация созданного графа

Исходный код программы заполнения графовой базы данных приведен в приложении Б.

## **4.2 Запрос**

В ходе выполнения курсовой работы для Neo4j было необходимо разработать и реализовать запрос: найти мастера, который выполнил максимальное количество заказов.

Для решения данной задачи был написан запрос на языке Cypher. Cypher является декларативным графовым языком запросов, который позволяет писать выразительные и эффективные запросы на получение данных из хранилища графов и их изменение. Cypher является относительно простым, но весьма мощным языком.

Исходный код данного запроса приведен в листинге Х.

Листинг Х — исходный код запроса

|  |
| --- |
| **#соединить узлы мастеров и заказов через отношения “выполнил”**  MATCH (MAS:Master)-[r:Executed]->(ORD:Order)  **# посчитать количество заказов на мастера**  WITH MAS, count(r) AS num  **# вернуть описания мастеров и кол-во заказов**  RETURN MAS.master\_desc as master\_desc, num  **# по убыванию числа заказов**  ORDER BY num DESC  **# лимит 1**  LIMIT 1;  **# результат:**  **# 'Воробьев Ювеналий Измаилович, Тип Образования: Среднее.' 10** |

Данный запрос аналогичен join мастеров и заказов, с последующей группировкой по мастерам и сортировкой в обычном SQL.

# **5 Spark**

## **5.1 Создание и заполнение таблиц**

Согласно заданию необходимо по данным из Elasticsearch сформировать csv-файлы (с внутренней схемой) таблиц «Заказчик», «Заказ», «Мастер» и сохранить их в файловой системе HDFS.

Схема алгоритма формирования таблиц приведена на рисунке Х.



Рисунок Х — схема алгоритма формирования таблиц

## **5.2 Запрос**

Согласно заданию было необходимо написать запрос select: найти заказы и мастеров, которые не выполнили заказы в срок.

Для этого соединим таблицы мастеров и заказов и отфильтруем только те строки, где дата завершения заказа больше планируемой.

Кроме того, присоединим таблицу заказчиков для получения доп. информации, используем left join на случай, если таблица заказчиков не полна (например, изначально в нашей системе не был предусмотрен учет информации о заказчиках и он был добавлен позднее).

Код запроса приведен в листинге Х.

Листинг Х — код запроса

|  |
| --- |
| select  m.master\_id, m.master\_desc, o.order\_id, o.order\_date, o.order\_due\_date, o.order\_fact\_completion\_date, c.order\_customer\_id, c.order\_customer\_desc  **# соединим мастеров и заказы**  from master m JOIN order o  ON o.order\_master\_id = m.master\_id  **# присоединим заказчиков, если они есть**  LEFT JOIN customer c  ON o.order\_customer\_id = c.order\_customer\_id  **# только просроченные**  where o.order\_fact\_completion\_date > o.order\_due\_date  **# Фрагмент результата**  +---------+---------------+--------+----------+----------+---------------+-----------+---------------+ |master\_id| master\_desc|order\_id|order\_date| due\_date|fact\_compl\_date|customer\_id| customer\_desc| +---------+---------------+--------+----------+----------+---------------+-----------+---------------+ | 257458|Шестакова На...| 131919|2024-01-06|2024-04-10| 2024-04-27| 227431|Имя: Маргари...| | 814777|Шарова Жанна...| 141550|2024-01-10|2024-04-03| 2024-04-27| 683623|Имя: Лора Бо...| | 871908|Козлов Юрий ...| 993858|2024-01-16|2024-04-08| 2024-04-27| 433504|Имя: Лукьян ...|   <...> |

## **5.3 Монитор**

Посмотрим показатели выполнения запросов через веб-интрефейс Spark. В нем видно, что в процессе исполнения программы было выполнено 10 jobs (рисунок Х).

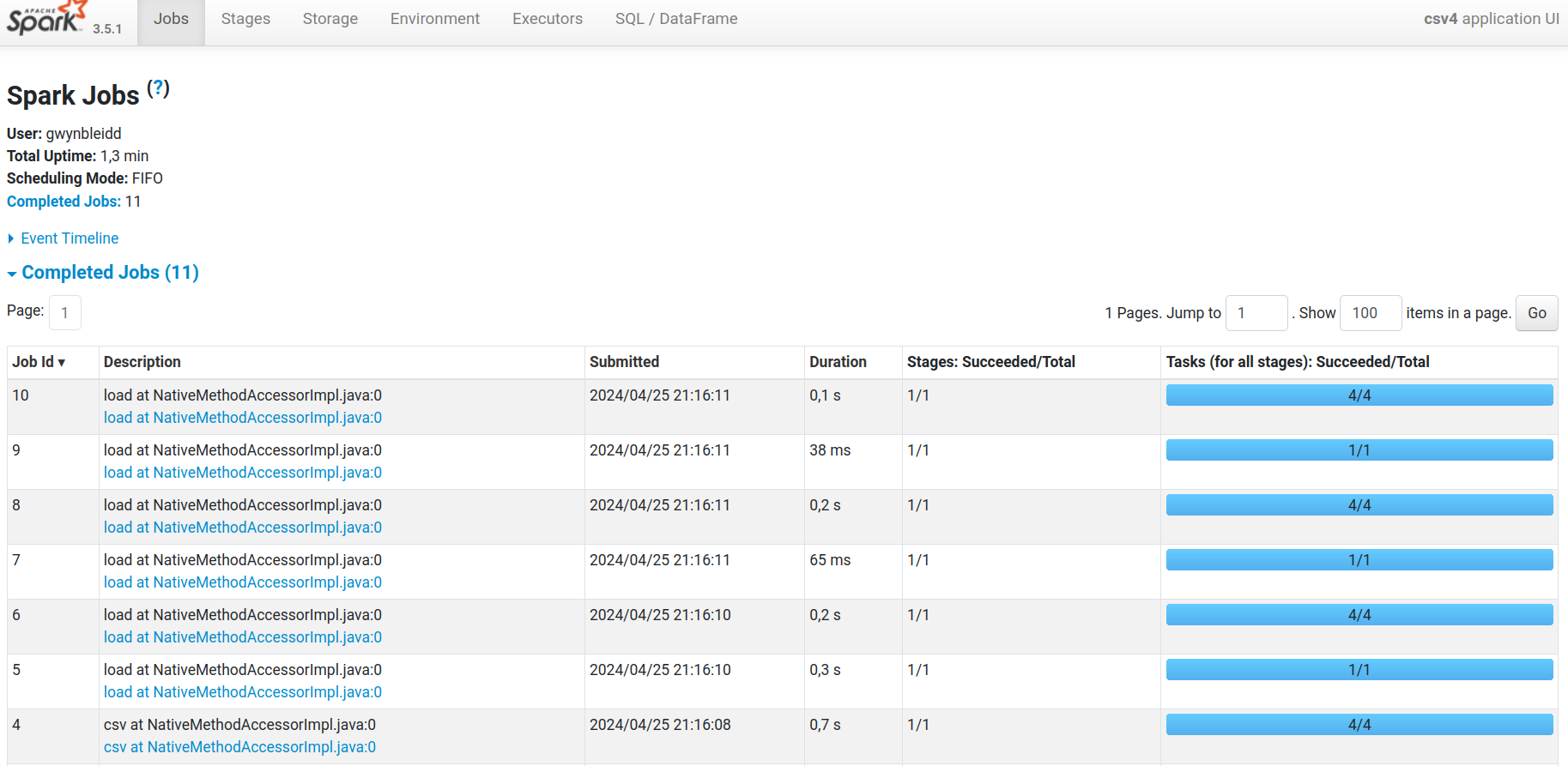


Рисунок Х — Spark jobs

Во вкладке SQL/Dataframe (рисунок Х) виден список SQL-запросов:

1-3) запись csv-файлов;

4-9) чтение файлов и создание временных view;

10) исполнение целевого sql-запроса

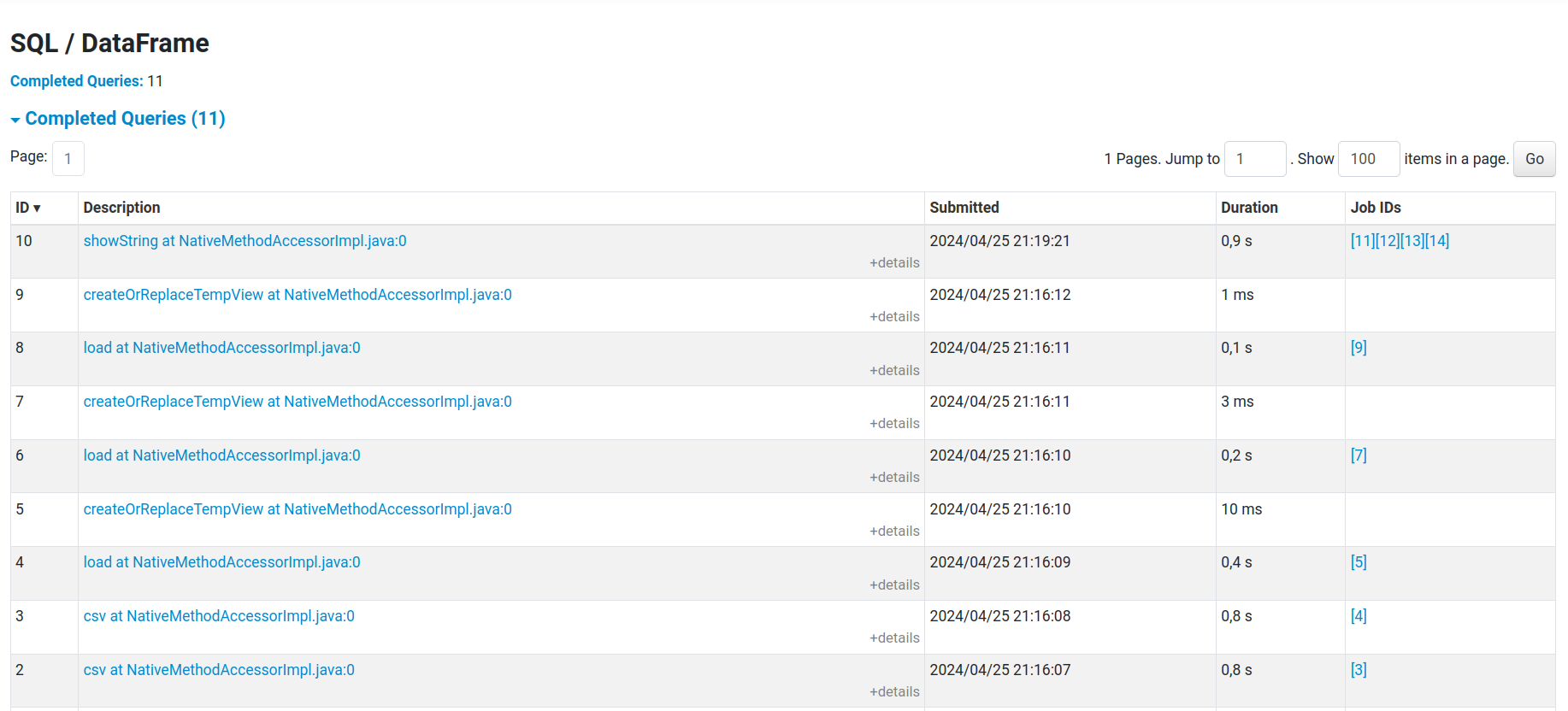


Рисунок Х — список SQL-запросов

Из данного риснука видно, что запрос состоит из 4 jobs, каждая по 1 stage. В кастве примера приведена временная характеристика для stage 11 (рисунок Х).

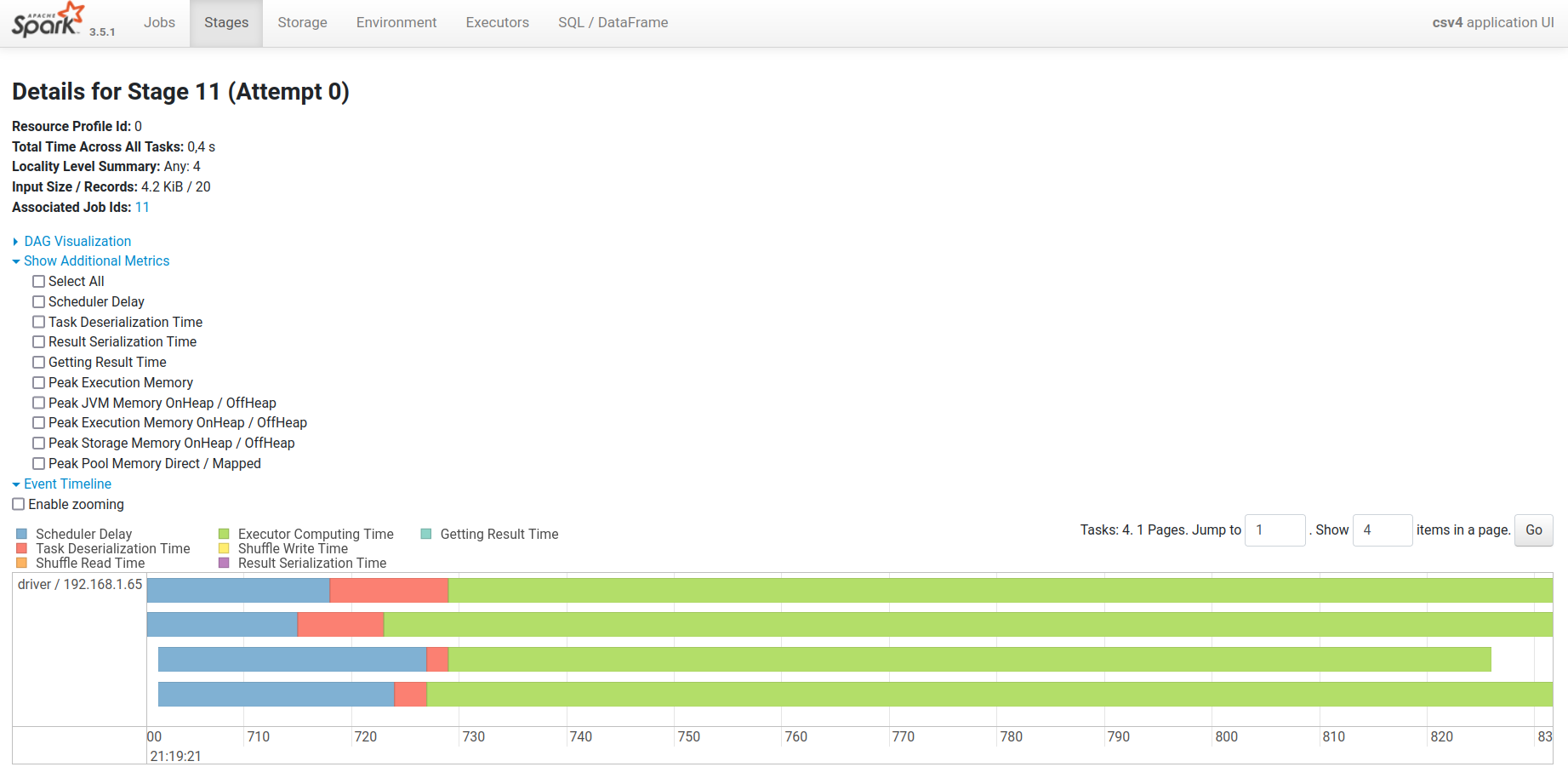


Рисунок Х — временная характеристика stage 11

Граф выполнения запроса приведен на рисунка Х-Х.



Рисунок Х — граф выполнения запроса



Рисунок Х — граф выполнения запроса

Сначала сканируется каждый из 3 csv-файлов, затем для каждого из них осуществляется фильтрация. После этого происходит широковещательная рассылка и hash-соединение таблицы мастеров и таблицы заказов. Затем, промежуточный результат соединяется с таблицей заказчиков аналогичным образом. После осуществляется проекция запрашиваемых колонок.

Так же ход выполнения запроса можно отследить по его планам. Построенные планы приведены в листинге Х.

Листинг Х — планы запроса

|  |
| --- |
| == Parsed Logical Plan ==  'Project ['m.master\_id, 'm.master\_desc, 'o.order\_id, 'o.order\_date, 'o.order\_due\_date AS due\_date#139, 'o.order\_fact\_completion\_date AS fact\_compl\_date#140, 'c.order\_customer\_id AS customer\_id#141, 'c.order\_customer\_desc AS customer\_desc#142]  +- 'Filter ('o.order\_fact\_completion\_date > 'o.order\_due\_date)  +- 'Join LeftOuter, ('o.order\_customer\_id = 'c.order\_customer\_id)  :- 'Join Inner, ('o.order\_master\_id = 'm.master\_id)  : :- 'SubqueryAlias m  : : +- 'UnresolvedRelation [master], [], false  : +- 'SubqueryAlias o  : +- 'UnresolvedRelation [order], [], false  +- 'SubqueryAlias c  +- 'UnresolvedRelation [customer], [], false  == Analyzed Logical Plan ==  master\_id: int, master\_desc: string, order\_id: int, order\_date: date, due\_date: date, fact\_compl\_date: date, customer\_id: int, customer\_desc: string  Project [master\_id#75, master\_desc#76, order\_id#98, order\_date#99, order\_due\_date#102 AS due\_date#139, order\_fact\_completion\_date#103 AS fact\_compl\_date#140, order\_customer\_id#135 AS customer\_id#141, order\_customer\_desc#136 AS customer\_desc#142]  +- Filter (order\_fact\_completion\_date#103 > order\_due\_date#102)  +- Join LeftOuter, (order\_customer\_id#100 = order\_customer\_id#135)  :- Join Inner, (order\_master\_id#107 = master\_id#75)  : :- SubqueryAlias m  : : +- SubqueryAlias master  : : +- View (`master`, [master\_id#75,master\_desc#76,master\_feedbacks#77])  : : +- Relation [master\_id#75,master\_desc#76,master\_feedbacks#77] csv  : +- SubqueryAlias o  : +- SubqueryAlias order  : +- View (`order`, [order\_id#98,order\_date#99,order\_customer\_id#100,order\_details\_desc#101,order\_due\_date#102,order\_fact\_completion\_date#103,order\_parts#104,repair\_types#105,order\_price#106,order\_master\_id#107])  : +- Relation [order\_id#98,order\_date#99,order\_customer\_id#100,order\_details\_desc#101,order\_due\_date#102,order\_fact\_completion\_date#103,order\_parts#104,repair\_types#105,order\_price#106,order\_master\_id#107] csv  +- SubqueryAlias c  +- SubqueryAlias customer  +- View (`customer`, [order\_customer\_id#135,order\_customer\_desc#136])  +- Relation [order\_customer\_id#135,order\_customer\_desc#136] csv  == Optimized Logical Plan ==  Project [master\_id#75, master\_desc#76, order\_id#98, order\_date#99, order\_due\_date#102 AS due\_date#139, order\_fact\_completion\_date#103 AS fact\_compl\_date#140, order\_customer\_id#135 AS customer\_id#141, order\_customer\_desc#136 AS customer\_desc#142]  +- Join LeftOuter, (order\_customer\_id#100 = order\_customer\_id#135)  :- Project [master\_id#75, master\_desc#76, order\_id#98, order\_date#99, order\_customer\_id#100, order\_due\_date#102, order\_fact\_completion\_date#103]  : +- Join Inner, (order\_master\_id#107 = master\_id#75)  : :- Project [master\_id#75, master\_desc#76]  : : +- Filter isnotnull(master\_id#75)  : : +- Relation [master\_id#75,master\_desc#76,master\_feedbacks#77] csv  : +- Project [order\_id#98, order\_date#99, order\_customer\_id#100, order\_due\_date#102, order\_fact\_completion\_date#103, order\_master\_id#107]  : +- Filter (((isnotnull(order\_fact\_completion\_date#103) AND isnotnull(order\_due\_date#102)) AND (order\_fact\_completion\_date#103 > order\_due\_date#102)) AND isnotnull(order\_master\_id#107))  : +- Relation [order\_id#98,order\_date#99,order\_customer\_id#100,order\_details\_desc#101,order\_due\_date#102,order\_fact\_completion\_date#103,order\_parts#104,repair\_types#105,order\_price#106,order\_master\_id#107] csv  +- Filter isnotnull(order\_customer\_id#135)  +- Relation [order\_customer\_id#135,order\_customer\_desc#136] csv  == Physical Plan ==  AdaptiveSparkPlan isFinalPlan=false  +- Project [master\_id#75, master\_desc#76, order\_id#98, order\_date#99, order\_due\_date#102 AS due\_date#139, order\_fact\_completion\_date#103 AS fact\_compl\_date#140, order\_customer\_id#135 AS customer\_id#141, order\_customer\_desc#136 AS customer\_desc#142]  +- BroadcastHashJoin [order\_customer\_id#100], [order\_customer\_id#135], LeftOuter, BuildRight, false  :- Project [master\_id#75, master\_desc#76, order\_id#98, order\_date#99, order\_customer\_id#100, order\_due\_date#102, order\_fact\_completion\_date#103]  : +- BroadcastHashJoin [master\_id#75], [order\_master\_id#107], Inner, BuildLeft, false  : :- BroadcastExchange HashedRelationBroadcastMode(List(cast(input[0, int, false] as bigint)),false), [plan\_id=336]  : : +- Filter isnotnull(master\_id#75)  : : +- FileScan csv [master\_id#75,master\_desc#76] Batched: false, DataFilters: [isnotnull(master\_id#75)], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://localhost:9000/master.csv], PartitionFilters: [], PushedFilters: [IsNotNull(master\_id)], ReadSchema: struct<master\_id:int,master\_desc:string>  : +- Filter (((isnotnull(order\_fact\_completion\_date#103) AND isnotnull(order\_due\_date#102)) AND (order\_fact\_completion\_date#103 > order\_due\_date#102)) AND isnotnull(order\_master\_id#107))  : +- FileScan csv [order\_id#98,order\_date#99,order\_customer\_id#100,order\_due\_date#102,order\_fact\_completion\_date#103,order\_master\_id#107] Batched: false, DataFilters: [isnotnull(order\_fact\_completion\_date#103), isnotnull(order\_due\_date#102), (order\_fact\_completion..., Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://localhost:9000/order.csv], PartitionFilters: [], PushedFilters: [IsNotNull(order\_fact\_completion\_date), IsNotNull(order\_due\_date), IsNotNull(order\_master\_id)], ReadSchema: struct<order\_id:int,order\_date:date,order\_customer\_id:int,order\_due\_date:date,order\_fact\_completi...  +- BroadcastExchange HashedRelationBroadcastMode(List(cast(input[0, int, false] as bigint)),false), [plan\_id=340]  +- Filter isnotnull(order\_customer\_id#135)  +- FileScan csv [order\_customer\_id#135,order\_customer\_desc#136] Batched: false, DataFilters: [isnotnull(order\_customer\_id#135)], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://localhost:9000/customer.csv], PartitionFilters: [], PushedFilters: [IsNotNull(order\_customer\_id)], ReadSchema: struct<order\_customer\_id:int,order\_customer\_desc:string> |

Всего система строит четыре плана выполнения запроса:

* Parsed Logical Plan — разобранный план выполнения запроса. На этом этапе проверяется только синтаксическая корректность запроса; ̶
* Analyzed Logical Plan —логический план выполнения запроса; ̶
* Optimized Logical Plan — оптимизированный логический план выполнения запроса;
* Physical Plan — физический план выполнения запроса.

# **Заключение**

В ходе выполнения курсовой работы был разработан макет аналитической системы в предметной области “Ремонт бытовой техники” на основе баз данных NoSQL ElasticSearch и Neo4j в связке с системами Hadoop и Spark.

Все необходимые программные компоненты были установлены на ПК под управлением 64-ых разрядной ОС Ubuntu версии 20.04.

Был развернут кластер ElasticSearch, который был заполнен при помощи сгенерированных JSON файлов. Для заполнения данными в кластере ElasticSearch были созданы два индекса. Также для индексов документов в ElasticSearch разработаны маппинги и анализаторы текстовых полей.

Графовая БД Neo4j была заполнена данными из кластера ElasticSearch. В Neo4j были созданы узлы типов «Заказ» и «Мастер», связи типа «Выполнил».

Были заполнены данными из Elasticsearch таблицы «Заказ», «Мастер» и «Заказчик». Эти таблицы затем были сохранены в виде CSV-файлов в распределенную файловую систему HDFS.

В каждой настроенной системе были написаны запросы и проанализированы результаты их работы.

# **Список использованных источников**

# **ПРИЛОЖЕНИЕ А**

Фрагменты JSON и результаты выполнения запросов

Листов 9

**1. Фрагменты JSON с сущностями “Мастер” (первые 10 записей)**

|  |
| --- |
| [  {  "index": "master",  "id": 563268,  "body": {  "master\_id": 563268,  "master\_desc": "Милица Руслановна Фомина, Тип Образования: Среднее.",  "master\_feedbacks": [  "ворчливый, медлительный.",  "ворчливый, медлительный.",  "медлительный, ворчливый."  ]  }  },  {  "index": "master",  "id": 871908,  "body": {  "master\_id": 871908,  "master\_desc": "Козлов Юрий Фомич, Тип Образования: Среднее.",  "master\_feedbacks": [  "ворчливый, аккуратный.",  "медлительный, аккуратный.",  "аккуратный, ворчливый."  ]  }  },  {  "index": "master",  "id": 164475,  "body": {  "master\_id": 164475,  "master\_desc": "Юлий Дмитриевич Константинов, Тип Образования: Среднее.",  "master\_feedbacks": [  "аккуратный, ворчливый."  ]  }  },  {  "index": "master",  "id": 464209,  "body": {  "master\_id": 464209,  "master\_desc": "Ия Тарасовна Степанова, Тип Образования: Высшее.",  "master\_feedbacks": [  "аккуратный, медлительный."  ]  }  },  {  "index": "master",  "id": 22653,  "body": {  "master\_id": 22653,  "master\_desc": "Акулина Рудольфовна Никитина, Стаж Работы: 15 Л./Г..",  "master\_feedbacks": [  "ворчливый, аккуратный.",  "ворчливый, медлительный."  ]  }  },  {  "index": "master",  "id": 282806,  "body": {  "master\_id": 282806,  "master\_desc": "Маслов Емельян Дмитриевич, Тип Образования: Высшее.",  "master\_feedbacks": [  "ворчливый, медлительный."  ]  }  },  {  "index": "master",  "id": 736774,  "body": {  "master\_id": 736774,  "master\_desc": "Владимирова Анжелика Эдуардовна, Тип Образования: Среднее.",  "master\_feedbacks": [  "аккуратный, ворчливый."  ]  }  },  {  "index": "master",  "id": 300714,  "body": {  "master\_id": 300714,  "master\_desc": "Вероника Петровна Силина, Стаж Работы: 12 Л./Г..",  "master\_feedbacks": [  "ворчливый, медлительный.",  "ворчливый, аккуратный."  ]  }  },  {  "index": "master",  "id": 405064,  "body": {  "master\_id": 405064,  "master\_desc": "Хохлов Олег Харлампьевич, Стаж Работы: 3 Л./Г..",  "master\_feedbacks": [  "ворчливый, медлительный.",  "ворчливый, аккуратный.",  "ворчливый, аккуратный."  ]  }  },  {  "index": "master",  "id": 257458,  "body": {  "master\_id": 257458,  "master\_desc": "Шестакова Наина Владиславовна, Стаж Работы: 9 Л./Г..",  "master\_feedbacks": [  "аккуратный, ворчливый.",  "ворчливый, аккуратный."  ]  }  },  ...  ] |

**2. Фрагменты JSON с сущностями “Заказ” (первые 10 записей)**

|  |
| --- |
| [  {  "index": "order",  "id": 131919,  "body": {  "order\_id": 131919,  "order\_date": "2024-01-06",  "order\_customer\_id": 227431,  "order\_customer\_desc": "Имя: Маргарита Валентиновна Чернова; премиальный, постоянный.",  "order\_details\_desc": "хрупкий, ценный.",  "order\_due\_date": "2024-04-10",  "order\_fact\_completion\_date": "2024-04-27",  "order\_parts": [  "аккумулятор"  ],  "repair\_types": [  "замена компонентов"  ],  "order\_price": 2296.89215138,  "order\_master\_id": 257458  }  },  {  "index": "order",  "id": 664730,  "body": {  "order\_id": 664730,  "order\_date": "2024-02-11",  "order\_customer\_id": 656432,  "order\_customer\_desc": "Имя: Комарова Дарья Борисовна; постоянный, премиальный.",  "order\_details\_desc": "хрупкий, тяжелый.",  "order\_due\_date": "2024-04-30",  "order\_fact\_completion\_date": "2024-04-20",  "order\_parts": [  "дисплей",  "дисплей",  "usb-разъем"  ],  "repair\_types": [  "мелкие детали",  "пайка"  ],  "order\_price": 3866.441,  "order\_master\_id": 785701  }  },  {  "index": "order",  "id": 141550,  "body": {  "order\_id": 141550,  "order\_date": "2024-01-10",  "order\_customer\_id": 683623,  "order\_customer\_desc": "Имя: Лора Борисовна Жукова; должник, постоянный.",  "order\_details\_desc": "хрупкий, тяжелый.",  "order\_due\_date": "2024-04-03",  "order\_fact\_completion\_date": "2024-04-27",  "order\_parts": [  "дисплей"  ],  "repair\_types": [  "анализ неисправности",  "мелкие детали"  ],  "order\_price": 4879.1271000384,  "order\_master\_id": 814777  }  },  {  "index": "order",  "id": 99676,  "body": {  "order\_id": 99676,  "order\_date": "2023-12-25",  "order\_customer\_id": 444528,  "order\_customer\_desc": "Имя: Василий Эдуардович Жданов; должник, премиальный.",  "order\_details\_desc": "тяжелый, ценный.",  "order\_due\_date": "2024-04-20",  "order\_fact\_completion\_date": "2024-04-04",  "order\_parts": [  "usb-разъем"  ],  "repair\_types": [  "замена компонентов",  "замена компонентов"  ],  "order\_price": 2103.6634502,  "order\_master\_id": 397809  }  },  {  "index": "order",  "id": 277607,  "body": {  "order\_id": 277607,  "order\_date": "2024-02-22",  "order\_customer\_id": 972535,  "order\_customer\_desc": "Имя: Гурий Анисимович Евдокимов; премиальный, постоянный.",  "order\_details\_desc": "хрупкий, ценный.",  "order\_due\_date": "2024-04-22",  "order\_fact\_completion\_date": "2024-04-10",  "order\_parts": [  "дисплей",  "корпус",  "корпус"  ],  "repair\_types": [  "замена компонентов",  "пайка"  ],  "order\_price": 5652.6299329,  "order\_master\_id": 163543  }  },  {  "index": "order",  "id": 832420,  "body": {  "order\_id": 832420,  "order\_date": "2024-03-01",  "order\_customer\_id": 872571,  "order\_customer\_desc": "Имя: Тимофеев Савва Григорьевич; постоянный, премиальный.",  "order\_details\_desc": "тяжелый, ценный.",  "order\_due\_date": "2024-04-14",  "order\_fact\_completion\_date": "2024-04-11",  "order\_parts": [  "usb-разъем",  "аккумулятор"  ],  "repair\_types": [  "мелкие детали"  ],  "order\_price": 5427.53678234973,  "order\_master\_id": 405064  }  },  {  "index": "order",  "id": 880172,  "body": {  "order\_id": 880172,  "order\_date": "2024-02-08",  "order\_customer\_id": 460465,  "order\_customer\_desc": "Имя: Лыткин Игнатий Валерьянович; должник, премиальный.",  "order\_details\_desc": "тяжелый, объемный.",  "order\_due\_date": "2024-04-23",  "order\_fact\_completion\_date": "2024-04-20",  "order\_parts": [  "usb-разъем"  ],  "repair\_types": [  "пайка",  "замена компонентов"  ],  "order\_price": 7003.855612,  "order\_master\_id": 871908  }  },  {  "index": "order",  "id": 993858,  "body": {  "order\_id": 993858,  "order\_date": "2024-01-16",  "order\_customer\_id": 433504,  "order\_customer\_desc": "Имя: Лукьян Венедиктович Богданов; премиальный, должник.",  "order\_details\_desc": "объемный, ценный.",  "order\_due\_date": "2024-04-08",  "order\_fact\_completion\_date": "2024-04-27",  "order\_parts": [  "дисплей",  "корпус"  ],  "repair\_types": [  "пайка"  ],  "order\_price": 7472.5287,  "order\_master\_id": 871908  }  },  {  "index": "order",  "id": 683490,  "body": {  "order\_id": 683490,  "order\_date": "2024-01-22",  "order\_customer\_id": 269146,  "order\_customer\_desc": "Имя: Алина Оскаровна Орехова; постоянный, должник.",  "order\_details\_desc": "ценный, тяжелый.",  "order\_due\_date": "2024-04-15",  "order\_fact\_completion\_date": "2024-04-20",  "order\_parts": [  "корпус",  "корпус"  ],  "repair\_types": [  "анализ неисправности",  "анализ неисправности"  ],  "order\_price": 2959.11938941,  "order\_master\_id": 300714  }  },  {  "index": "order",  "id": 243483,  "body": {  "order\_id": 243483,  "order\_date": "2024-02-01",  "order\_customer\_id": 408481,  "order\_customer\_desc": "Имя: Филатов Карл Алексеевич; должник, премиальный.",  "order\_details\_desc": "объемный, тяжелый.",  "order\_due\_date": "2024-04-05",  "order\_fact\_completion\_date": "2024-04-27",  "order\_parts": [  "дисплей",  "корпус",  "дисплей"  ],  "repair\_types": [  "анализ неисправности",  "пайка"  ],  "order\_price": 2323.37152,  "order\_master\_id": 736774  }  },  ...  ] |

**3. Результат запроса №1 к ElasticSearch**

|  |
| --- |
| [{'key\_as\_string': '2023-12-01',  'key': 1701388800000,  'doc\_count': 13,  'over\_parts': {'doc\_count\_error\_upper\_bound': 0,  'sum\_other\_doc\_count': 0,  'buckets': [{'key': 'usb-разъем', 'doc\_count': 5},  {'key': 'дисплей', 'doc\_count': 4},  {'key': 'корпус', 'doc\_count': 4},  {'key': 'аккумулятор', 'doc\_count': 3}]}},  {'key\_as\_string': '2024-01-01',  'key': 1704067200000,  'doc\_count': 48,  'over\_parts': {'doc\_count\_error\_upper\_bound': 0,  'sum\_other\_doc\_count': 0,  'buckets': [{'key': 'usb-разъем', 'doc\_count': 23},  {'key': 'дисплей', 'doc\_count': 22},  {'key': 'корпус', 'doc\_count': 18},  {'key': 'аккумулятор', 'doc\_count': 16}]}},  {'key\_as\_string': '2024-02-01',  'key': 1706745600000,  'doc\_count': 37,  'over\_parts': {'doc\_count\_error\_upper\_bound': 0,  'sum\_other\_doc\_count': 0,  'buckets': [{'key': 'дисплей', 'doc\_count': 19},  {'key': 'usb-разъем', 'doc\_count': 14},  {'key': 'аккумулятор', 'doc\_count': 14},  {'key': 'корпус', 'doc\_count': 14}]}},  {'key\_as\_string': '2024-03-01',  'key': 1709251200000,  'doc\_count': 2,  'over\_parts': {'doc\_count\_error\_upper\_bound': 0,  'sum\_other\_doc\_count': 0,  'buckets': [{'key': 'usb-разъем', 'doc\_count': 1},  {'key': 'аккумулятор', 'doc\_count': 1},  {'key': 'корпус', 'doc\_count': 1}]}}] |

**4. Результат запроса №2 к ElasticSearch**

|  |
| --- |
| {'total': {'value': 8, 'relation': 'eq'},  'max\_score': 0.8435577,  'hits': [{'\_index': 'master',  '\_type': '\_doc',  '\_id': '22653',  '\_score': 0.8435577,  '\_source': {'master\_id': 22653,  'master\_desc': 'Акулина Рудольфовна Никитина, Стаж Работы: 15 Л./Г..',  'master\_feedbacks': ['ворчливый, аккуратный.',  'ворчливый, медлительный.']}},  {'\_index': 'master',  '\_type': '\_doc',  '\_id': '300714',  '\_score': 0.8435577,  '\_source': {'master\_id': 300714,  'master\_desc': 'Вероника Петровна Силина, Стаж Работы: 12 Л./Г..',  'master\_feedbacks': ['ворчливый, медлительный.',  'ворчливый, аккуратный.']}},  {'\_index': 'master',  '\_type': '\_doc',  '\_id': '405064',  '\_score': 0.8435577,  '\_source': {'master\_id': 405064,  'master\_desc': 'Хохлов Олег Харлампьевич, Стаж Работы: 3 Л./Г..',  'master\_feedbacks': ['ворчливый, медлительный.',  'ворчливый, аккуратный.',  'ворчливый, аккуратный.']}},  {'\_index': 'master',  '\_type': '\_doc',  '\_id': '257458',  '\_score': 0.8435577,  '\_source': {'master\_id': 257458,  'master\_desc': 'Шестакова Наина Владиславовна, Стаж Работы: 9 Л./Г..',  'master\_feedbacks': ['аккуратный, ворчливый.', 'ворчливый, аккуратный.']}},  {'\_index': 'master',  '\_type': '\_doc',  '\_id': '814777',  '\_score': 0.8435577,  '\_source': {'master\_id': 814777,  'master\_desc': 'Шарова Жанна Аркадьевна, Стаж Работы: 1 Л./Г..',  'master\_feedbacks': ['ворчливый, медлительный.',  'медлительный, аккуратный.',  'ворчливый, медлительный.']}},  {'\_index': 'master',  '\_type': '\_doc',  '\_id': '847121',  '\_score': 0.8435577,  '\_source': {'master\_id': 847121,  'master\_desc': 'Жуков Ладимир Всеволодович, Стаж Работы: 12 Л./Г..',  'master\_feedbacks': ['медлительный, ворчливый.',  'аккуратный, медлительный.']}},  {'\_index': 'master',  '\_type': '\_doc',  '\_id': '163543',  '\_score': 0.8435577,  '\_source': {'master\_id': 163543,  'master\_desc': 'Максимильян Виленович Гурьев, Стаж Работы: 11 Л./Г..',  'master\_feedbacks': ['аккуратный, медлительный.']}},  {'\_index': 'master',  '\_type': '\_doc',  '\_id': '785701',  '\_score': 0.8435577,  '\_source': {'master\_id': 785701,  'master\_desc': 'Панфилова Пелагея Игоревна, Стаж Работы: 8 Л./Г..',  'master\_feedbacks': ['медлительный, аккуратный.',  'медлительный, ворчливый.']}}]} |

# **ПРИЛОЖЕНИЕ Б**

Исходные коды программ

Листов 9

**1. Программа генерации JSON документов**

|  |
| --- |
| from faker import Faker  import json  import random  fake = Faker('ru\_RU') **# русскоязычные ФИО**  master\_id\_set = set() **# существующие id мастеров**  **# функция генерации записи о заказе**  def generate\_order\_data():  order\_id = fake.pyint(min\_value=1, max\_value=1000000)  order\_customer\_id = fake.pyint(min\_value=1, max\_value=1000000)  order\_customer\_name = fake.name() # ФИО  **# описание: ФИО + доп. информация**  smp = random.sample(["премиальный", "постоянный", "должник"], 2)  order\_customer\_desc = f'Имя: {order\_customer\_name}; {smp[0]}, {smp[1]}.'  smp = random.sample(["хрупкий", "тяжелый", "объемный", "ценный"], 2)  order\_details = f'{smp[0]}, {smp[1]}.'  **# генерация дат в заданном промежутке**  order\_date = fake.date\_between(start\_date="-70d",  end\_date="today")  order\_due\_date = fake.date\_between(start\_date="+30d", end\_date="+60d")  order\_completion\_date = fake.date\_between(start\_date="+30d", end\_date="+60d")  order\_parts = [fake.word(ext\_word\_list=['корпус', 'usb-разъем', 'дисплей', 'аккумулятор']) for \_ in range(fake.pyint(min\_value=1, max\_value=3))]  repair\_types = [fake.word(ext\_word\_list=['замена компонентов', 'пайка', 'анализ неисправности', 'мелкие детали']) for \_ in range(fake.pyint(min\_value=1, max\_value=2))]  order\_price = fake.pyfloat(min\_value=1000, max\_value=10000)  **# исполнитель - из множества существующих**  order\_master\_id = random.choice(list(master\_id\_set))  return {  "index": "order",  "id": order\_id,  "body": {  "order\_id": order\_id, **#id\_заказа**  "order\_date": order\_date.strftime("%Y-%m-%d"), **#дата\_заказа**  "order\_customer\_id": order\_customer\_id, **#id\_заказчика**  "order\_customer\_desc": order\_customer\_desc, **#сведения\_о\_заказчике**  "order\_details\_desc": order\_details, **#данные\_о\_заказе**  "order\_due\_date": order\_due\_date.strftime("%Y-%m-%d"), **#срок\_выполнения\_заказа**  "order\_fact\_completion\_date": order\_completion\_date.strftime("%Y-%m-%d"), **#фактическая\_дата\_выполнения**  "order\_parts": order\_parts, **#запчасть**  "repair\_types": repair\_types, **#ремонт**  "order\_price": order\_price, **#стоимость**  "order\_master\_id": order\_master\_id **#id\_мастера**  }  }  **# генерация записи о мастере**  def generate\_master\_data():  master\_id = fake.pyint(min\_value=1, max\_value=1000000)  **# пополнение множества существующих мастеров**  master\_id\_set.add(master\_id)  master\_name = fake.name()  master\_feedbacks = []    for \_ in range(fake.pyint(min\_value=1, max\_value=3)):  smp = random.sample(["аккуратный", "медлительный", "ворчливый"], 2)  master\_feedback = f'{smp[0]}, {smp[1]}.'  master\_feedbacks.append(master\_feedback)    master\_xp = f'Стаж работы: {fake.random\_int(min=1, max=15)} л./г.'  if fake.random\_int(min=1, max=5) == 2:  master\_edu = 'Тип образования: высшее'  else :  master\_edu = 'Тип образования: среднее'  return {  "index": "master",  "id": master\_id,  "body": {  "master\_id": master\_id,  "master\_desc": f'{master\_name}, {fake.sentence(ext\_word\_list=[master\_xp, master\_edu], nb\_words=1, variable\_nb\_words=True)}', **#сведения\_о\_мастере**  "master\_feedbacks": master\_feedbacks #отзывы\_о\_работе  }  }  **# сгенерировать 20 и 100 записей, записать в файлы**  masters = [generate\_master\_data() for \_ in range(20)]  orders = [generate\_order\_data() for \_ in range(100)]  with open('../masters.json','w') as f:  json.dump(masters,f)  with open('../orders.json','w') as f:  json.dump(orders,f) |

**2. Программа заполнения ES**

|  |
| --- |
| import json  from elasticsearch import Elasticsearch  **# подключение**  client = Elasticsearch([{"host": "127.0.0.1", "scheme": "http", "port": 9200}])  **# настройки анализатора**  analyzer\_settings = {  "settings": {  "analysis": {  "filter": {  "ru\_stop": {  "type": "stop",  "stopwords": "\_russian\_"  },  "snow\_ru\_stemmer": {  "type": "snowball",  "language": "russian"  }  },  "analyzer": {  "custom\_analyzer": {  "type": "custom",  "tokenizer": "standard",  "filter": [  "lowercase",  "ru\_stop",  "snow\_ru\_stemmer"  ]  }  }  }  }  }  **# маппинг заказов**  mappings\_order = {  "mappings" : {  "properties" : {  "order\_customer\_desc" : {  "type" : "text",  "analyzer": "custom\_analyzer"  },  "order\_customer\_id" : {  "type" : "long"  },  "order\_date" : {  "type" : "date"  },  "order\_details\_desc" : {  "type" : "text",  "analyzer": "custom\_analyzer"  },  "order\_due\_date" : {  "type" : "date"  },  "order\_fact\_completion\_date" : {  "type" : "date"  },  "order\_id" : {  "type" : "long"  },  "order\_master\_id" : {  "type" : "long"  },  "order\_parts" : {  "type" : "text",  "fielddata" : True,  "fields" : {  "keyword" : {  "type" : "keyword"  }  }  ,"analyzer": "custom\_analyzer"  },  "order\_price" : {  "type" : "float"  },  "repair\_types" : {  "type" : "text",  "fields" : {  "keyword" : {  "type" : "keyword"  }  }  ,"analyzer": "custom\_analyzer"  }  }  }  }  **# (пере)создание индекса**  indexName = "order"  if client.indices.exists(index=indexName):  client.indices.delete(index=indexName)  client.indices.create(index=indexName,  body={\*\*analyzer\_settings  , \*\*mappings\_order  }  )  **# заполнение индекса**  with open('../orders.json', 'r') as f:  dataStore = json.load(f)  for data in dataStore:  try:  client.index(  index=data["index"],  # doc\_type=data["doc\_type"],  id=data["id"],  body=data["body"]  )  except Exception as e:  print(e, end="")  print("index 'order' data inserted")  **# маппинг мастеров**  mappings\_master = {  "mappings" : {  "properties" : {  "master\_desc" : {  "type" : "text",  "analyzer": "custom\_analyzer"  },  "master\_feedbacks" : {  "type" : "text",  "fields" : {  "keyword" : {  "type" : "keyword"  }  }  ,"analyzer": "custom\_analyzer"  },  "master\_id" : {  "type" : "long"  }  }  }  }  **# (пере)создание индекса**  indexName = "master"if client.indices.exists(index=indexName):  client.indices.delete(index=indexName)  client.indices.create(index=indexName,  body={\*\*analyzer\_settings  , \*\*mappings\_master  }  )  **# заполнение индекса**  with open('../masters.json', 'r') as f:  dataStore = json.load(f)  for data in dataStore:  try:  client.index(  index=data["index"],  id=data["id"],  body=data["body"]  )  except Exception as e:  print(e, end="")  print("index 'master' data inserted") |

**3. Программа заполнения Neo4j**

|  |
| --- |
| import json  from elasticsearch import Elasticsearch  from py2neo import Graph, Node, Relationship  **# подключение к ES**  client = Elasticsearch([{"host": "127.0.0.1", "scheme": "http", "port": 9200}])  **# подключение к Neo4j**  graph\_db = Graph("bolt://localhost:7687", user='neo4j', password='my\_password')  searchBody = {  "size": 9999,  "\_source": True,  "query": {  "match\_all": {}  }  }  **# получение всех мастеров**  result = client.search(index="master", body=searchBody)  masters = result['hits']['hits']  **# очистка Neo4j**  graph\_db.run("MATCH (n) OPTIONAL MATCH (n)-[r]-() DELETE n,r;")  i = 0  **# для каждого мастера: ...**  for master in masters:  try:  print(f'{i})', end='')  **# создать узел**  MasterNode = Node("Master",master\_id=master['\_source']['master\_id'], master\_desc=master['\_source']['master\_desc'])  graph\_db.create(MasterNode)  **# найти соответствующие заказы**  searchBody = {  "size": 9999,  "\_source": True,  "query": {  "term": {  "order\_master\_id": master['\_source']['master\_id']  }  }  }  result = client.search(index="order", body=searchBody)  orders = result['hits']['hits']  **# создать узлы заказов и связи с ними**  for order in orders:  OrderNode = Node("Order",order\_id=order['\_source']['order\_id'], order\_date=order['\_source']['order\_date'], order\_customer\_desc=order['\_source']['order\_customer\_desc'], order\_price=order['\_source']['order\_price'])  graph\_db.create(OrderNode)  NodesRelationship = Relationship(MasterNode, "Executed", OrderNode, order\_due\_date=order['\_source']['order\_due\_date'], order\_fact\_completion\_date=order['\_source']['order\_fact\_completion\_date'])  graph\_db.create(NodesRelationship)  print('.', end='')  print("+")  i += 1  except Exception as e:  print(f'Exception: {e}')  continue  print("data written successfully") |

**4. Программа формирования csv-файлов в HDFS**

|  |
| --- |
| from pyspark.sql import SparkSession  from elasticsearch import Elasticsearch  from pyspark.sql.types import \*  import findspark  findspark.init()  **# подключение к ES и Spark**  client = Elasticsearch("http://localhost:9200")  sparkSession = SparkSession.builder.appName("csv4").getOrCreate()  **# получение всех мастеров и заказов из ES**  searchBody = {  "size": 9999,  "\_source": True,  "query": {  "match\_all": {}  }  }  result = client.search(index="master", body=searchBody)  masters = result['hits']['hits']  result = client.search(index="order", body=searchBody)  orders = result['hits']['hits']  **# описание схем**  MasterSchema = StructType([  StructField("master\_id", IntegerType(), False),  StructField("master\_desc", StringType(), False),  StructField("master\_feedbacks", StringType(), False)  ])  CustomerSchema = StructType([  StructField("order\_customer\_id", IntegerType(), False),  StructField("order\_customer\_desc", StringType(), False)  ])  OrderSchema = StructType([  StructField("order\_id", IntegerType(), False),  StructField("order\_date", StringType(), False),  StructField("order\_customer\_id", IntegerType(), False),  StructField("order\_details\_desc", StringType(), False),  StructField("order\_due\_date", StringType(), False),  StructField("order\_fact\_completion\_date", StringType(), False),  StructField("order\_parts", StringType(), False),  StructField("repair\_types", StringType(), False),  StructField("order\_price", FloatType(), False),  StructField("order\_master\_id", IntegerType(), False)  ])  **# заполнение массивов**  MasterTable = []  CustomerTable = []  OrderTable = []  CustomerSet = set()  for master in masters:  MasterTable.append((  master['\_source']['master\_id'],  master['\_source']['master\_desc'],  str(master['\_source']['master\_feedbacks'])  ))  for order in orders:  OrderTable.append((  order['\_source']['order\_id'],  order['\_source']['order\_date'],  order['\_source']['order\_customer\_id'],  order['\_source']['order\_details\_desc'],  order['\_source']['order\_due\_date'],  order['\_source']['order\_fact\_completion\_date'],  str(order['\_source']['order\_parts']),  str(order['\_source']['repair\_types']),  order['\_source']['order\_price'],  order['\_source']['order\_master\_id']  ))  if not order['\_source']['order\_customer\_id'] in CustomerSet:  CustomerSet.add(order['\_source']['order\_customer\_id'])  CustomerTable.append((  order['\_source']['order\_customer\_id'],  order['\_source']['order\_customer\_desc']  ))  **# преоброазование в dataframe**  MasterDF = sparkSession.createDataFrame(MasterTable, MasterSchema)  OrderDF = sparkSession.createDataFrame(OrderTable, OrderSchema)  CustomerDF = sparkSession.createDataFrame(CustomerTable,CustomerSchema)  **# запись в HDFS**  MasterDF.write.csv(path='hdfs://0.0.0.0:9000/master.csv',mode='overwrite', header=True)  OrderDF.write.csv(path='hdfs://0.0.0.0:9000/order.csv',mode='overwrite', header=True)  CustomerDF.write.csv(path='hdfs://0.0.0.0:9000/customer.csv',mode='overwrite', header=True)  print('data is written successfully')  sparkSession.stop() **# закрытие сессии** |

**5.  Программа реализации запроса для работы монитора**

|  |
| --- |
| import findspark  from pyspark.sql import SparkSession  findspark.init()  **# подключение к Spark**  sparkSession = SparkSession.builder.appName("csv4").getOrCreate()  **# считывание csv и регистрация временных таблиц**  data = sparkSession.read.load("hdfs://localhost:9000/master.csv", format="csv",sep=",", inferSchema="true", header="true")  data.registerTempTable("master")  data = sparkSession.read.load("hdfs://localhost:9000/order.csv", format="csv",sep=",", inferSchema="true", header="true")  data.registerTempTable("order")  data = sparkSession.read.load("hdfs://localhost:9000/customer.csv", format="csv",sep=",", inferSchema="true", header="true")  data.registerTempTable("customer")  **# исполнение запроса и вывод результата**  result = sparkSession.sql("""select m.master\_id, m.master\_desc, o.order\_id, o.order\_date, o.order\_due\_date, o.order\_fact\_completion\_date, c.order\_customer\_id, c.order\_customer\_desc  from master m JOIN order o ON o.order\_master\_id = m.master\_id LEFT JOIN customer c ON o.order\_customer\_id = c.order\_customer\_id  where o.order\_fact\_completion\_date > o.order\_due\_date  """)  result.show()  Input()  sparkSession.stop() **# закрытие сессии** |

# **ПРИЛОЖЕНИЕ В**

Гафический материал

Листов 11

