

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«МИРЭА – Российский технологический университет»**

**РТУ МИРЭА**

Институт: «Институт кибербезопасности и цифровых технологий»

Направление: 09.03.02 «Информационные системы и технологии»

Кафедра: КБ-14 «Цифровые технологии обработки данных»

Дисциплина: «Методы Big Data»

**ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3**

**«Обработка реляционных данных с применением Hive, Pig, MapReduce»**

Выполнил студент БСБО-08-18 Шайко Т.И.

Руководитель работы: Лебедев А.С.

Москва 2021 г.

Оглавление

[Цель работы 3](#_Toc93031731)

[Задача работы 3](#_Toc93031732)

[Теоретическое введение 3](#_Toc93031733)

[Hive 3](#_Toc93031734)

[Pig 4](#_Toc93031735)

[Ход выполнения 7](#_Toc93031736)

[Запросы Pig 10](#_Toc93031737)

[Запросы Hive 17](#_Toc93031738)

[Собственная реализация запросов на MapReduce 18](#_Toc93031739)

[Список использованной литературы 25](#_Toc93031740)

# Цель работы

Научиться использовать системы Hive и Pig для работы с базами данных. Написать несколько запросов на языках PigLatin, HiveQL и также напрямую написать их с помощью модели MapReduce. Сравнить полученные результаты выполнения этих запросов.

# Задача работы

− Разработать базу данных (не менее 3 сущностей). Разработать не менее 5 запросов к БД, из них не менее 3 с применением JOIN.

− Реализовать БД и запросы на Hive и Pig.

− Реализовать запросы с применением паттерна MapReduce.

− Продемонстрировать, что результаты выполнения запросов на Hive, Pig совпадают с результатами, полученными с помощью собственной реализации MapReduce.

# Теоретическое введение

## Hive

Одним из главных компонентов информационной платформы, построенной груп­пой Джеффа Хаммербахера в Facebook, была система Hive, предназначенная для организации информационных хранилищ на базе Hadoop. Технология Hive выросла из по­требности в управлении и извлечении информации из огромных объемов данных, ежедневно производимых Facebook в стремительно растущей социальной сети. Опробовав несколько разных систем, группа выбрала Hadoop для хранения и об­работки информации, так как эта технология была экономичной и удовлетворяла их потребности в масштабировании.

Система Hive создавалась для того, чтобы аналитики, хорошо владеющие SQL (но слабо разбирающиеся в программировании на Java), могли выполнять запро­сы к гигантским объемам данных, хранимых Facebook в HDFS. Сегодня Hive — успешный проект Apache, используемый во многих организациях как универсаль­ная и масштабируемая платформа обработки данных.

Конечно, язык SQL не идеален для некоторых задач больших данных — например, он плохо подходит для построения сложных алгоритмов машинного самообучения. Тем не менее во многих аналитических областях он прекрасно работает; еще одно огромное преимущество — его известность в отрасли. Кроме того, SQL является общепринятым языком инструментов бизнес-аналитики, так что интеграция Hive с этими продуктами абсолютно уместна.

## Pig

Pig повышает уровень абстракции при обработке больших наборов данных. MapReduce позволяет программисту определить функцию map, за которой следует функция reduce, но вписать обработку данных в этот шаблон, который часто требует нескольких этапов MapReduce, может быть сложно. При использовании Pig, структуры данных становятся намного сложнее, как правило, многозначными и вложенными, и преобразования, которые можно применить к данным, намного мощнее. Например, они включают соединения, которые сложны в реализации в MapReduce.

Pig состоит из двух частей:

* Язык, используемый для описания потоков данных, называется Pig Latin.
* Среда выполнения для запуска программ Pig Latin. В настоящее время существует две среды: локальное выполнение в одной JVM и распределенное выполнение в кластере Hadoop.

Программа Pig Latin состоит из последовательности операций или преобразований, которые применяются к входным данным для получения выходных данных. В целом, операции описывают поток данных, который среда выполнения Pig преобразует в исполняемое представление и затем запускает. Pig превращает преобразования в серию задач MapReduce, но, программист в основном не знает об этом, что позволяет ему сосредоточиться на обработке данных, а не на механизме исполнения.

Pig — это скриптовый язык для анализа больших наборов данных. В основном MapReduce критикуют в том, что цикл разработки очень длинный. Написание классов mappers и reducers, компиляция и упаковка кода, отправка заданий и получение результатов — трудоемкое дело, и даже с потоковой передачей, которая исключает этап компиляции и упаковки, опыт по-прежнему необходим. Привлекательная сторона Pig — способность обрабатывать терабайты данных в ответ на полдюжины строк Pig Latin, введенных с консоли. Действительно, он был создан в Yahoo!, чтобы исследователям и инженерам было легче анализировать огромные наборы данных. Pig очень поддерживает программиста, пишущего запрос, так как предоставляет несколько команд для анализа структур данных в вашей программе во время ее написания. Еще более полезно то, что он может выполнить пробный прогон для репрезентативного подмножества ваших входных данных, чтобы можно было увидеть, есть ли ошибки в обработке, прежде чем запускать ее на полном наборе данных.

Pig была разработана, чтобы быть масштабируемой. Фактически все стадии обработки настраиваемы: загрузка, хранение, фильтрация, группировка и объединение могут быть изменены пользовательскими функциями (UDF). Эти функции работают с моделью вложенных данных Pig, поэтому они могут очень глубоко интегрироваться с операторами Pig. Еще одним преимуществом является то, что UDF имеют тенденцию быть более пригодными для повторного использования, чем библиотеки, разработанные для написания программ MapReduce.

В некоторых случаях Pig работает не так хорошо, как программы, написанные в MapReduce. Однако с каждым выпуском его отставание сокращается, поскольку команда Pig реализует сложные алгоритмы для применения реляционных операторов Pig. Справедливо сказать, что если программист не готов вкладывать много усилий в оптимизацию кода Java MapReduce, написание запросов на языке Pig Latin сэкономит его время.

# Ход выполнения

Для начала нужно скачать и установить Hive и Pig в систему CentOS и после запускать уже вместе с Hadoop.

Первым делом, как написано в методичке преподавателя [1], устанавливается программный комплекс Hive (см. рисунок 1).

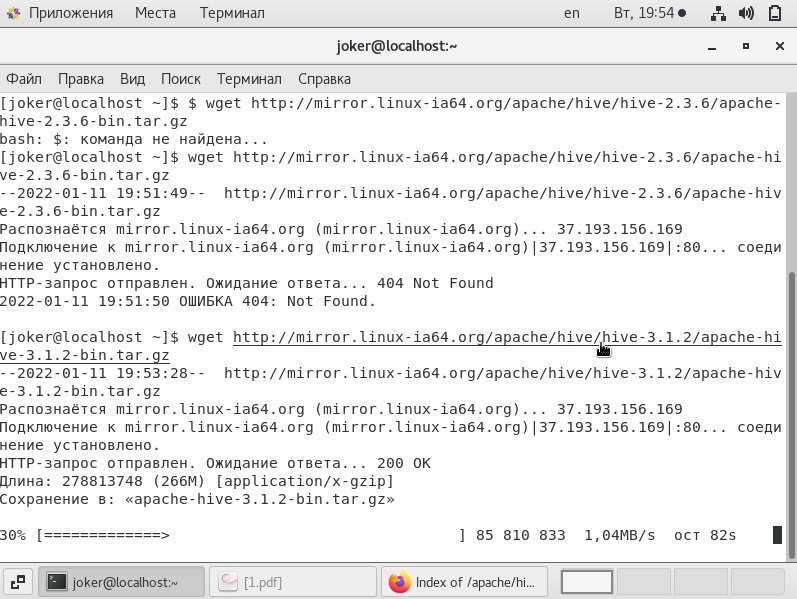


Рисунок 1 — Скачивание Hive с официального сервера с помощью «wget»

Затем после того, как отредактированы все нужные конфигурационные файлы, в том числе и файл .bash\_profile, необходимо протестировать работу программы: должны успешно выполняться все операции в HiveQL — создание таблицы, вставка в неё значений и отображение её содержимого. Для тестов нужно при первом запуске исполнить вспомогательный bash-скрипт «hiveMeta.sh» (см. рисунок 2) и затем после успешного его выполнения скрипт «hiveStart.sh».

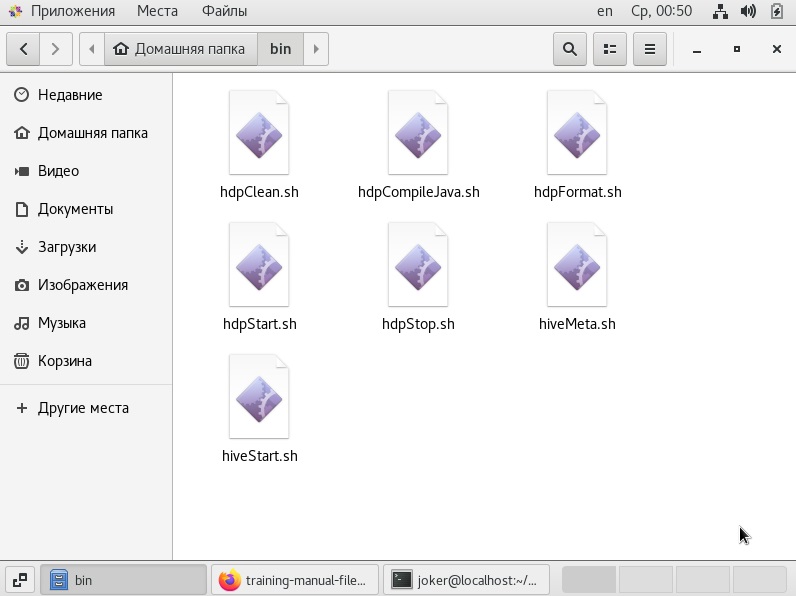


Рисунок 2 — Каталог со вспомогательными bash-скриптами от преподавателя

После установки и настройки Hive следует приступить к скачиванию и установке похожим образом Pig.

Все созданные отношения по мере исполнения кода на PigLatin должны успешно сохраняться в систему и в само хранилище HDFS (см. рисунок 3).

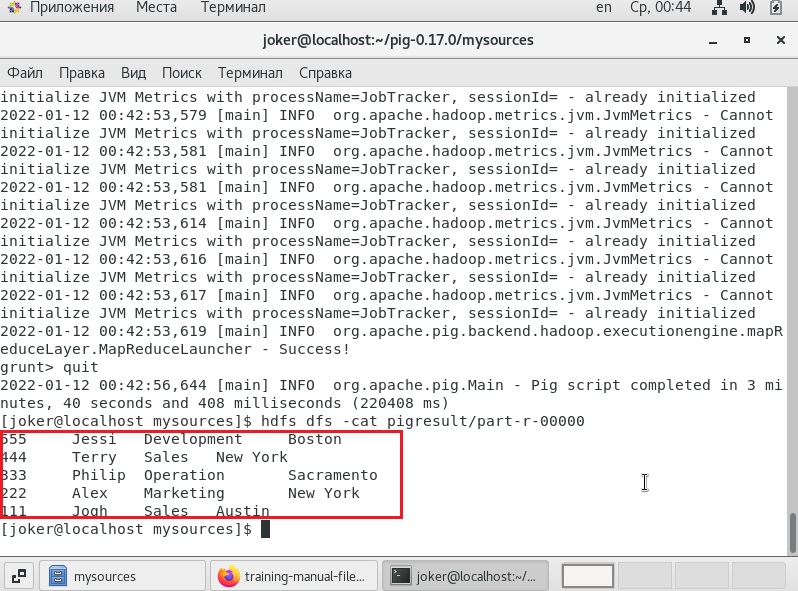


Рисунок 3 — После работы с таблицей в примере от преподавателя она успешно сохранилась в HDFS и отобразилась при чтении соответствующего файла

Теперь настало время написать пять запросов к базе данных, которую нужно сначала создать. Пусть она состоит из трёх отношений, которые указаны ниже.

Таблица 1 — Сущность «Книга»

|  |  |  |  |
| --- | --- | --- | --- |
| Книга | | | |
| **Идентификатор** | **Название** | **Автор** | **Год издания** |
| 1 | Язык программирования C++ | Бьерн Страуструп | 2010 |
| 2 | Монах, который продал свой «Феррари» | Робин Шарма | 2020 |

Таблица 1 — Сущность «Книга» (Продолжение)

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | Теория музыки для чайников | Майкл Пилхофер | 2009 |
| 4 | Методы BigData | Артём Сергеевич Лебедев | 2021 |

Таблица 2 — Сущность «Заказ»

|  |  |  |
| --- | --- | --- |
| Заказ | | |
| **Идентификатор[[1]](#footnote-1)** | **Заказчик** | **Идентификатор товара** |
| 1 | 1233 | 1 |
| 2 | 2145 | 4 |

Таблица 3 — Сущность «Заказчик»

|  |  |  |
| --- | --- | --- |
| Заказчик | | |
| **Паспорт** | **ФИО** | **Возраст** |
| 1234 | Петров А.И. | 18 |
| 1233 | Иванов И.С. | 35 |
| 2145 | Зелёный А.В. | 21 |

Теперь, реализовав в виде входных текстовых файлов эту базу данных (для собственной реализации MapReduce) и в виде непосредственно отношений в Pig и Hive, можно с ней работать.

## Запросы Pig

Сначала обрабатываются запросы в системе Pig. Создаётся сама БД, а затем к ней применяются любые пять операций, типа, выборки, вставки записей, изменения полей и тому подобные действия.

Список запросов, которые писались в Pig:

1. SELECT \* FROM books;
2. INSERT INTO orders VALUES (3, 1234, 3);
3. SELECT books.name, orders.customer\_id FROM books INNER JOIN orders ON books.id = orders.product\_id;
4. SELECT \* FROM customers LEFT OUTER JOIN orders ON customers.id = orders.customer\_id;
5. SELECT \* FROM customers, books WHERE customers.age < books.year;

По условию, нужно написать пять запросов, из которых не менее трёх — содержат операцию JOIN. Так вот, пятый запрос тоже содержит JOIN, но неявно, потому что он, не смотря ни на что, всё же объединяет две таблицы.

Итак, заполняются текстовые файлы с начальными данными обсуждаемой базы и запускается Pig (см. рисунок 4).

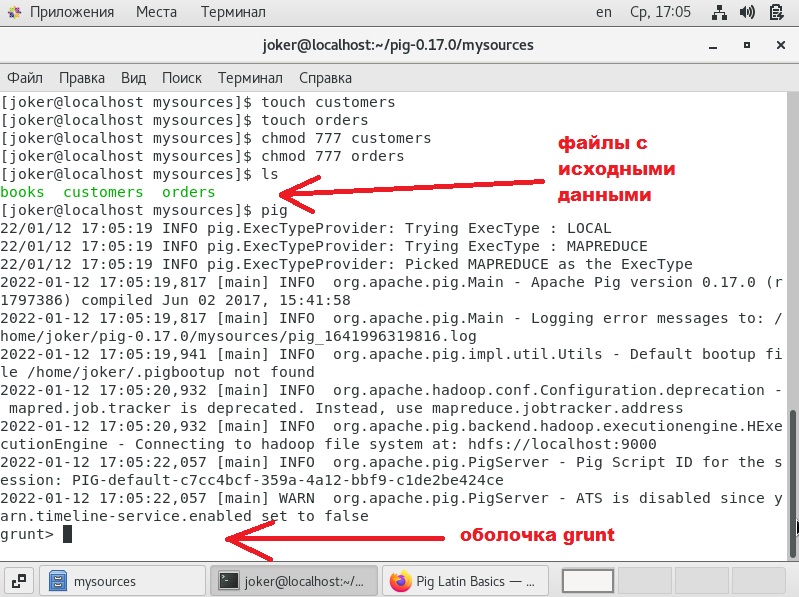


Рисунок 4 — Подготовка к выполнению запросов в Pig

Но нельзя забыть положить все эти три исходника в HDFS командой «put» (см. рисунок 5).

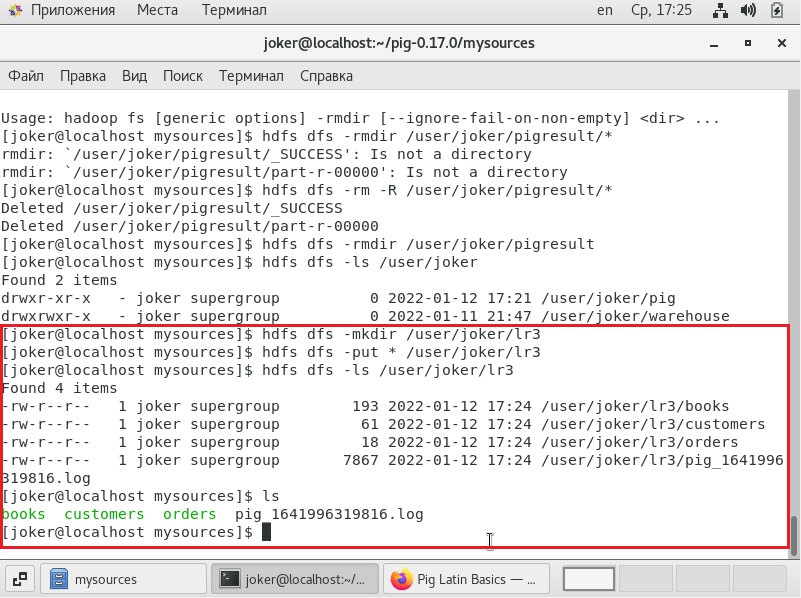


Рисунок 5 — Отправка исходных данных в HDFS

Теперь дело подходит к загрузке базы данных с помощью оператора LOAD в PigLatin и работе с ней (см. рисунок 6).

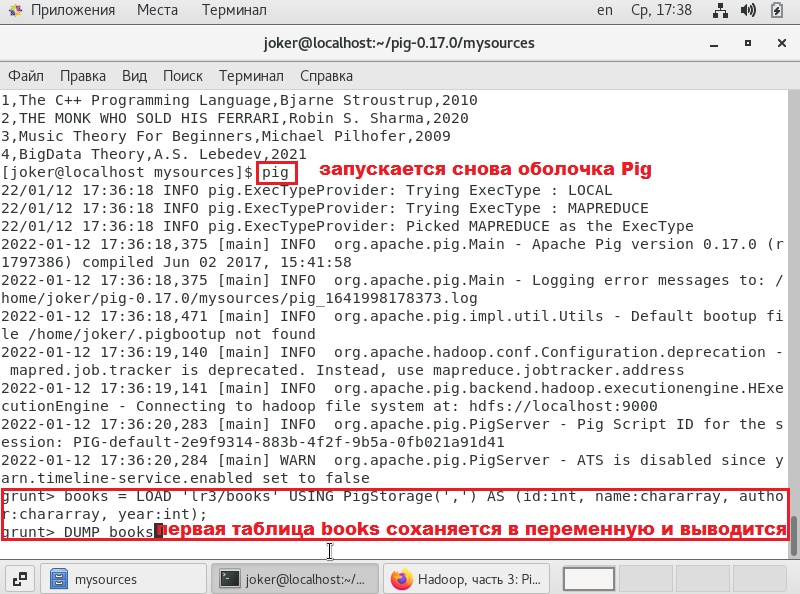


Рисунок 6 — Загрузка данных в Pig из первой таблицы books

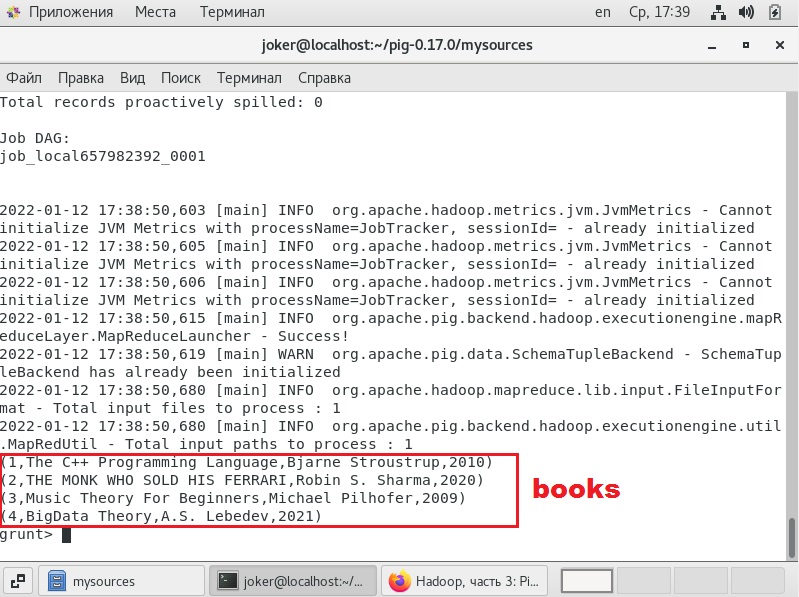


Рисунок 7 — DUMP первой таблицы books

Таким образом, уже получен результат выполнения первого запроса «SELECT \* FROM books;» (см. список пяти запросов выше). Нужно сохранить текущие данные в соответствующий новый каталог в HDFS командой «STORE» (см. рисунок 8).

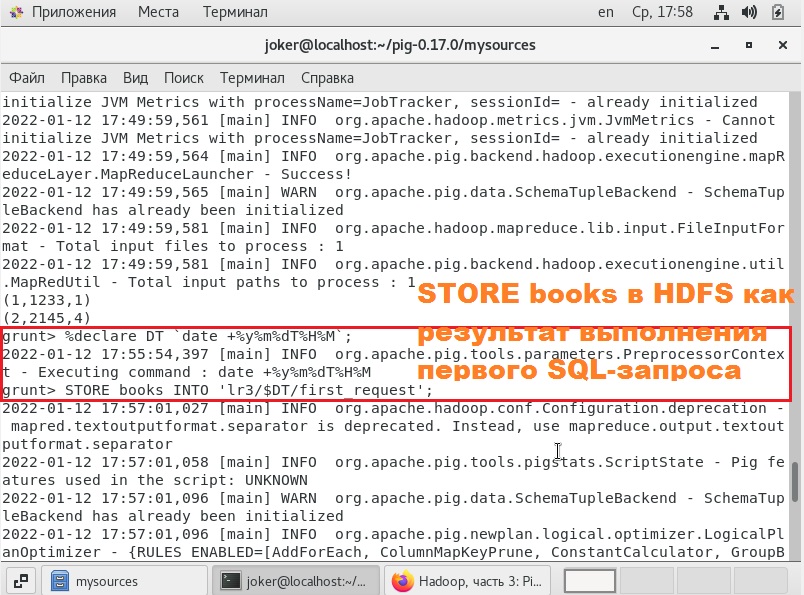


Рисунок 8 — Сохранение «STORE» в HDFS первого результата

После этого выполняется следующий запрос вставки данных в таблицу «Заказы» (см. рис. 10). Здесь применяется операция объединения двух Pig-сумок —UNION — для вставки новой записи в таблицу «Заказы».

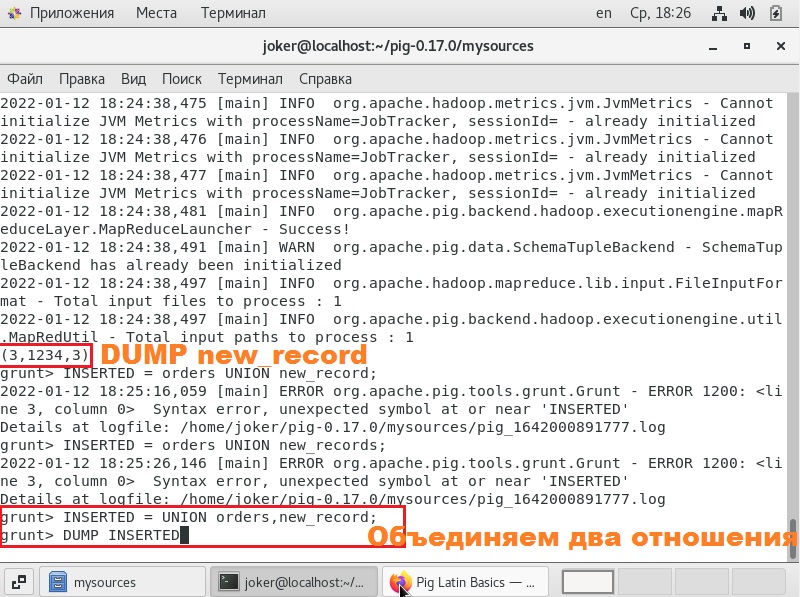


Рисунок 9 — Операция UNION

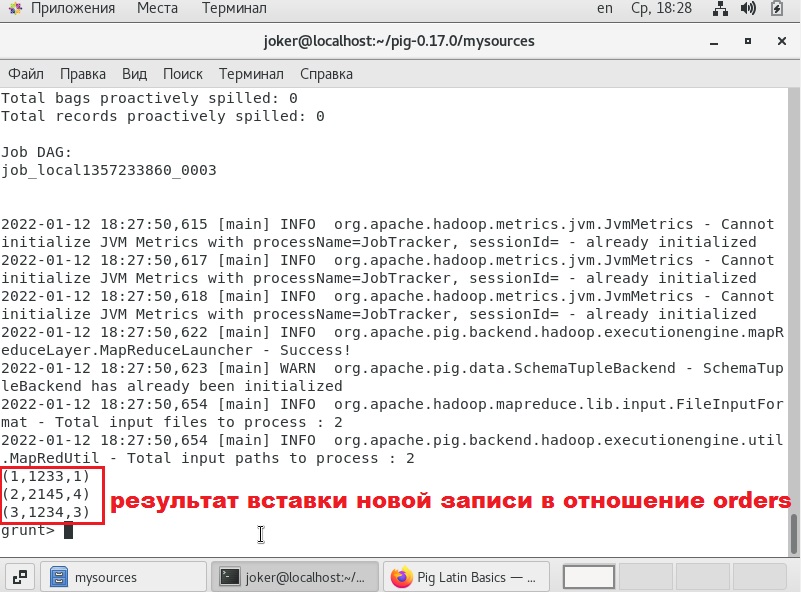


Рисунок 10 — DUMP результирующего отношения (INSERTED)

Для реализации третьего запроса:

SELECT **books**.name, **orders**.customer\_id

FROM **books**

INNER JOIN **orders**

ON **books**.id = **orders**.product\_id;

где определяются покупатели и купленные ими книги, необходимо последовательно исполнить примерно такие действия:

1. inner\_join = **JOIN** books **BY** id, orders **BY** product\_id;
2. **DUMP** inner\_join
3. inner\_join2 = **FOREACH** inner\_join **GENERATE** $1, $5;
4. **DUMP** inner\_join2

Результат представлен на рисунке ниже.

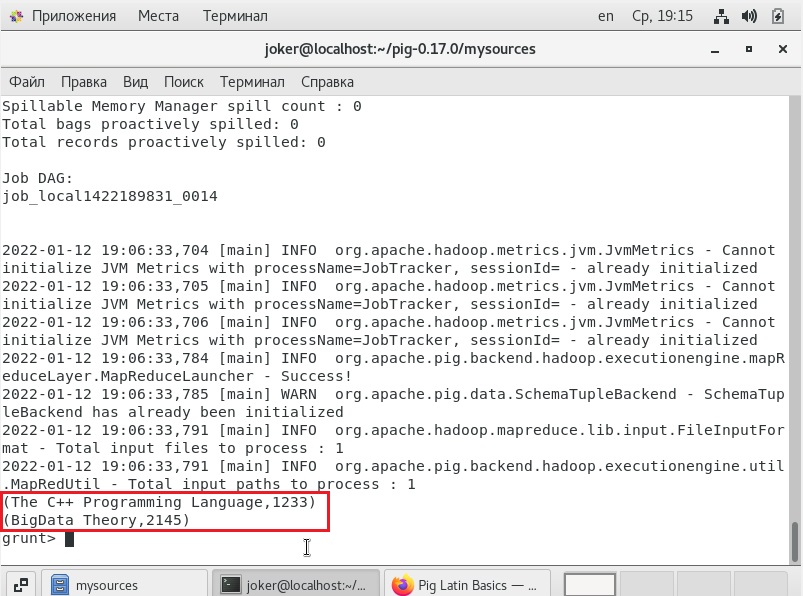


Рисунок 11 — Результат выполнения третьего запроса

Четвёртый запрос выглядит так:

SELECT \* FROM **customers**

LEFT OUTER JOIN **orders**

ON **customers**.id = **orders**.customer\_id;

Реализуется он ещё проще, чем третий, предыдущий запрос — всего одной командой:

1. outer\_join = **JOIN** customers **BY** id **LEFT OUTER**, orders **BY** customer\_id;
2. **DUMP** outer\_join

В итоге должен вывестись следующий результат:



То есть заказчик Петров А.И. ещё ничего не покупал, в отличие от остальных двух человек.

И пятый запрос

SELECT \* FROM **customers**, **books**

WHERE **customers**.age < **books**.year;

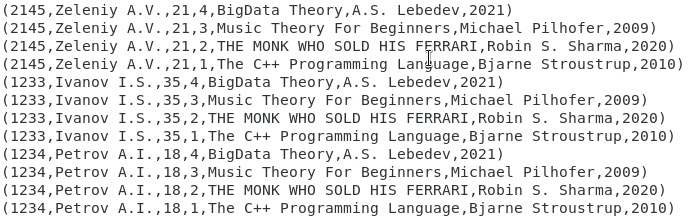
можно реализовать в три строчки:

1. cross\_join = **CROSS** customers, books;
2. **ASSERT** cross\_join **BY** $2 < $6;
3. **DUMP** cross\_join

Этот запрос представляет собой как раз неявный оператор CROSS JOIN и в результате даст декартово произведение первого отношения на второе, то есть, в данном случае, в итоге будет кол-во строк, оцениваемое как три записи в «Заказчиках», умноженные на четыре записи в «Книгах» — двенадцать различных записей в результирующем отношении.

В Pig Latin есть специальный оператор для выполнения подобного действия «CROSS». Следующий за ним оператор «ASSERT» просто удостоверит тот факт, что все записи в результате окажутся удовлетворяющими условию «возраст заказчика меньше года издания книги». Это, как должно быть ясно из контекста, всегда верно для каждой записи.

Затем результат выводится на консоль:



Не забывайте сохранять результат с помощью команды «STORE», чтобы после выхода из grunt вы ничего не потеряли и потом смогли восстановить при необходимости все нужные данные!

## Запросы Hive

Вначале поэтапно создаётся база данных в Hive (см. рисунок 12). А потом просто выполняются все запросы, указанные выше, с учетом синтаксиса языка MySQL.

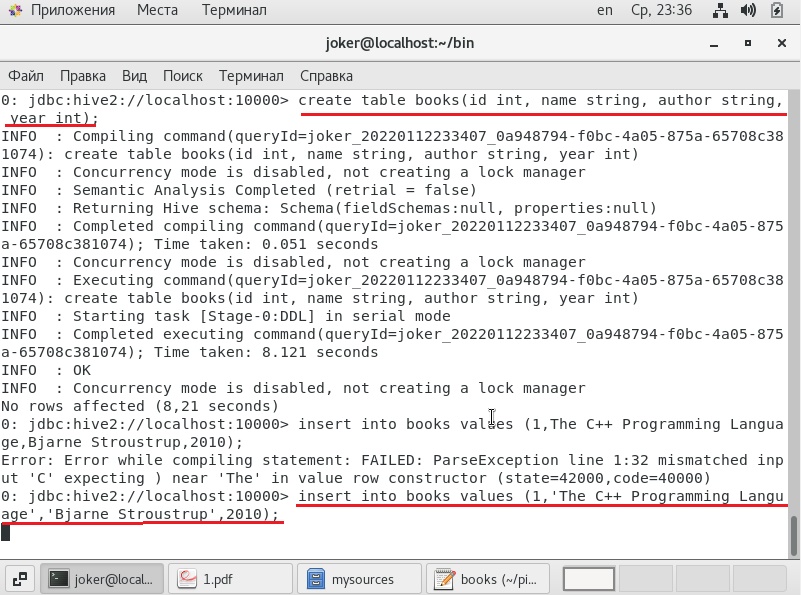


Рисунок 12 — Создание и заполнение таблицы «books» в Hive

## Собственная реализация запросов на MapReduce

Итак, программы каждого запроса будут писаться на Java в этой работе. Создаётся maven-проект «lr3» с автоматической подзагрузкой библиотек, в том числе и Java-библиотек технологии MapReduce2. Затем для выполнения первого запроса создается класс «MyMapper», в котором будет находиться логика части «map».

Нужно просто входные данные из файла вывести как таблицу. Поэтому часть «reduce» здесь не потребуется — только map. Код выборки указанного отношения «Книги» представлен ниже, также как и код главного класса Main с инициализацией данных и запуском определенного числа потоков, выполняющих поставленные задачи в рамках модели MapReduce.

package mirea.student.shayko.lr3;  
  
import mirea.student.shayko.lr3.Request1.MyMapper;  
import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Job;  
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
  
public class Main {  
 public static void main(String[] args) throws Exception {  
 Configuration conf = new Configuration();  
 Job job = new Job(conf, "request1");  
 job.setJarByClass(Request1.class);  
 job.setMapperClass(MyMapper.class);  
 job.setOutputKeyClass(IntWritable.class);  
 job.setOutputValueClass(Text.class);  
 FileInputFormat.addInputPath(job, new Path(args[0]));  
 FileOutputFormat.setOutputPath(job, new Path(args[1]));  
 System.exit(job.waitForCompletion(true) ? 0 : 1);  
 }  
}

Файл 1 — Main.java

package mirea.student.shayko.lr3;  
  
import java.io.IOException;  
import java.util.StringTokenizer;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Mapper;  
  
public class Request1$MyMapper extends Mapper<Object, Text, IntWritable, Text> {  
 private static IntWritable *counter* = new IntWritable(1);  
  
 public void map(Object key, Text value, Mapper<Object, Text, IntWritable, Text>.Context context) throws IOException, InterruptedException {  
 StringTokenizer st = new StringTokenizer(value.toString(), ",");  
 IntWritable id = new IntWritable(Integer.parseInt(st.nextToken().toString()));  
 Text name = new Text(st.nextToken().toString());  
 Text author = new Text(st.nextToken().toString());  
 IntWritable year = new IntWritable(Integer.parseInt(st.nextToken().toString()));  
 if (*counter*.get() == 1) {  
 context.write(new IntWritable(0), new Text("|+|id | name | author | year |+|"));  
 }  
  
 context.write(*counter*, new Text("|+| " + id.toString() + " | " + name.toString() + " | " + author.toString() + " | " + year.toString() + " |+|"));  
 int i = *counter*.get() + 1;  
 *counter* = new IntWritable(i);  
 }  
}

Файл 2 — Request1.java

В результате выполнения этого кода должен появиться при чтении из HDFS следующий вывод — см. рисунок 13.

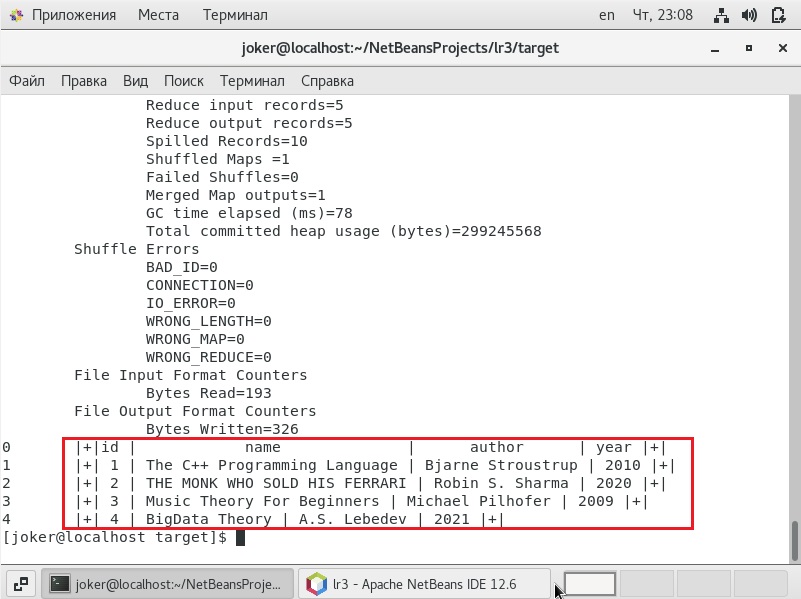


Рисунок 13 — Результат выполнения первого запроса SELECT

Теперь реализуется третий запрос с INNER JOIN. Здесь уже потребуется и класс MyReducer. В MyMapper происходит сортировка прочитанных записей всех двух таблиц: books и orders — по своим reducer’ам. Таким образом, каждый reducer примет к себе на вход все сгруппированные mapper’ом записи какой-то одной таблицы и добавит их в общий массив таблиц с именем «tables». Дальше какой-нибудь один reducer в итоге просто просмотрит все записи двух таблиц, которые требуется соединить (JOIN) в запросе, и сравнит значения «books.id» и «orders.product\_id» соответствующих записей. И если эти значения равны, то он добавит эти две записи в окончательный вывод программы (результат см. на рисунке 14).

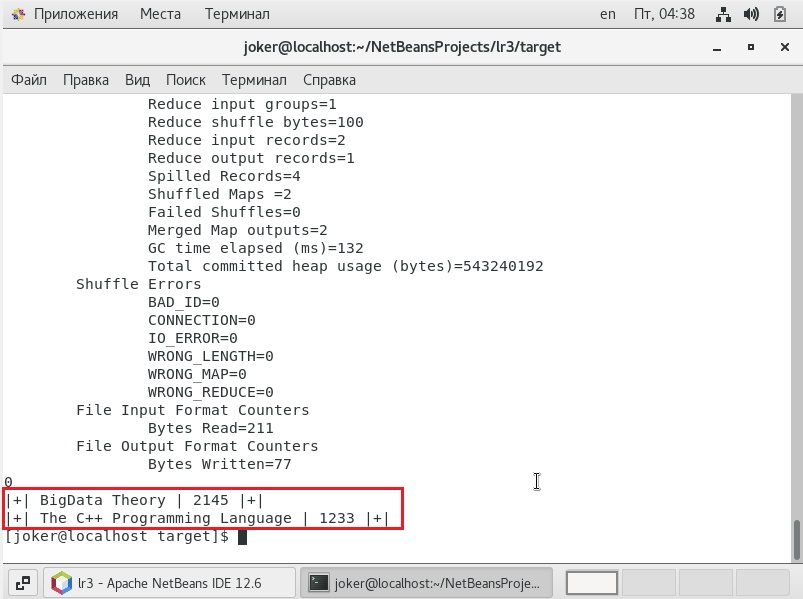


Рисунок 14 — Результат работы программы третьего запроса

Причём все исходники .java запускались с помощью bash-скрипта, образец которого выдал наш преподаватель (см. рисунок 15).

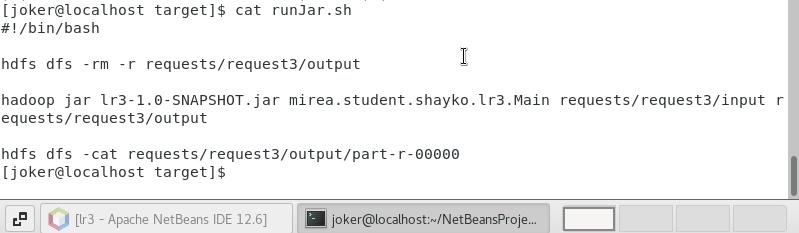


Рисунок 15 — Скрипт, который использовался для тестирования работы последней программы третьего JOIN-запроса

Два исходника последней реализованной программы по третьему запросу перечислены ниже.

Файл 3 — Main.java

*//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//*package mirea.student.shayko.lr3;  
  
import mirea.student.shayko.lr3.Request3.MyMapper;  
import mirea.student.shayko.lr3.Request3.MyReducer;  
import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Job;  
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
  
public class Main {  
 public Main() {  
 }  
  
 public static void main(String[] args) throws Exception {  
 Configuration conf = new Configuration();  
 Job job = new Job(conf, "request3");  
 job.setJarByClass(Request3.class);  
 job.setMapperClass(MyMapper.class);  
 job.setCombinerClass(MyReducer.class);  
 job.setNumReduceTasks(1);  
 job.setReducerClass(MyReducer.class);  
 job.setOutputKeyClass(IntWritable.class);  
 job.setOutputValueClass(Text.class);  
 FileInputFormat.addInputPath(job, new Path(args[0]));  
 FileOutputFormat.setOutputPath(job, new Path(args[1]));  
 System.exit(job.waitForCompletion(true) ? 0 : 1);  
 }  
}

package mirea.student.shayko.lr3;  
  
import java.io.IOException;

import java.util.ArrayList;  
import java.util.Iterator;  
import java.util.StringTokenizer;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;  
  
public class Request3$MyMapper extends Mapper<Object, Text, IntWritable, Text> {  
  
 public void map(Object key, Text value, Mapper<Object, Text, IntWritable, Text>.Context context) throws IOException, InterruptedException {  
 IntWritable id;  
 try {  
 StringTokenizer st = new StringTokenizer(value.toString(), ",");  
 IntWritable id = new IntWritable(Integer.parseInt(st.nextToken().toString()));  
 id = new IntWritable(Integer.parseInt(st.nextToken().toString()));  
 IntWritable product\_id = new IntWritable(Integer.parseInt(st.nextToken().toString()));  
 context.write(new IntWritable(2), new Text(id.toString() + "|" + id.toString() + "|" + product\_id.toString()));  
 } catch (NumberFormatException var10) {  
 StringTokenizer st = new StringTokenizer(value.toString(), ",");  
 id = new IntWritable(Integer.parseInt(st.nextToken().toString()));  
 Text name = new Text(st.nextToken().toString());  
 Text author = new Text(st.nextToken().toString());  
 IntWritable year = new IntWritable(Integer.parseInt(st.nextToken().toString()));  
 context.write(new IntWritable(1), new Text(id.toString() + "|" + name.toString() + "|" + author.toString() + "|" + year.toString()));  
 }  
  
 }  
}

public class Request3$MyReducer extends Reducer<IntWritable, Text, IntWritable, Text> {  
 private static ArrayList<Text> *tables* = new ArrayList();  
 private static int *counter* = 0;  
 private static Text *result* = new Text("");  
  
 public void reduce(IntWritable key, Iterable<Text> values, Reducer<IntWritable, Text, IntWritable, Text>.Context context) throws IOException, InterruptedException {  
 Text nextTable = new Text("");  
  
 Text val;  
 for(Iterator var5 = values.iterator(); var5.hasNext(); nextTable = new Text(nextTable.toString() + val.toString() + "===")) {  
 val = (Text)var5.next();  
 }  
  
 *tables*.add(nextTable);

Файл 4 — Request3.java

if (*tables*.size() > 1) {  
 StringTokenizer books;  
 StringTokenizer orders;  
 if (key.toString().equals("1")) {  
 books = new StringTokenizer(nextTable.toString(), "===");  
 if (nextTable.toString().equals(*tables*.get(0))) {  
 orders = new StringTokenizer(((Text)*tables*.get(1)).toString(), "===");  
 } else {  
 orders = new StringTokenizer(((Text)*tables*.get(0)).toString(), "===");  
 }  
 } else {  
 orders = new StringTokenizer(nextTable.toString(), "===");  
 if (nextTable.toString().equals(*tables*.get(0))) {  
 books = new StringTokenizer(((Text)*tables*.get(1)).toString(), "===");  
 } else {  
 books = new StringTokenizer(((Text)*tables*.get(0)).toString(), "===");  
 }  
 }  
  
 ArrayList orders2 = new ArrayList();  
  
 while(orders.hasMoreTokens()) {  
 orders2.add(new Text(orders.nextToken().toString()));  
 }  
  
 ArrayList books2 = new ArrayList();  
  
 while(books.hasMoreTokens()) {  
 books2.add(new Text(books.nextToken().toString()));  
 }  
  
 for(int i = 0; i < books2.size(); ++i) {  
 for(int j = 0; j < orders2.size(); ++j) {  
 StringTokenizer booksNextRecord = new StringTokenizer(((Text)books2.get(i)).toString(), "|");  
 StringTokenizer ordersNextRecord = new StringTokenizer(((Text)orders2.get(j)).toString(), "|");  
 String book\_id = booksNextRecord.nextToken().toString();  
 ordersNextRecord.nextToken();  
 String customer\_id = ordersNextRecord.nextToken().toString();  
 String product\_id = ordersNextRecord.nextToken().toString();  
 if (book\_id.equals(product\_id)) {  
 String book\_name = booksNextRecord.nextToken().toString();  
 *result* = new Text(*result* + "\n|+| " + book\_name + " | " + customer\_id + " |+|");  
 }  
 }  
 }  
 }  
  
 context.write(new IntWritable(0), *result*);  
 }  
}

# Список использованной литературы

1. Лебедев А.С. Методы Big Data [Электронный ресурс]: Учебно-методическое пособие / Лебедев А.С., Магомедов Ш.Г. – М.: МИРЭА – Российский технологический университет, 2021. – 1 электрон. опт. диск (CD-ROM).
2. Лебедев А.С. Big Data. Лекция 4 - Введение в PIG. - Учебный год 2021/2022.
3. Лебедев А.С. Big Data. Лекция 5 - Основы HIVE. - Учебный год 2021/2022
4. Лебедев А.С. Big Data. Лекция 3 - Алгоритмы и паттерны MapReduce. - Учебный год 2021/2022

1. Жёлтым маркером выделены первичные ключи отношений. [↑](#footnote-ref-1)