|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА − Российский технологический университет»**  **РТУ МИРЭА** |

**Институт информационных технологий (ИИТ)**

**Кафедра прикладной математики (ПМ)**

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ**

по дисциплине «Системы управления данными»

**Практическое занятие № 3**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент группы ИНБО-01-17 | *ИМБО-02-22, Ким Кирилл Сергеевич* | (подпись) | |
| Преподаватель | *Алексеева Екатерина Сергеевна, преподаватель* | (подпись) | |
| Отчет представлен | «\_\_\_»\_\_\_\_\_\_\_\_202\_\_г. | |  | |

Москва 2025 г.

1. Практическая работа №3
   1. Цель работы

В этой работе вы будете использовать Pig для изучения, исправления и предварительной обработки данных в файлах. Два файла данных, которые будут использоваться в лабораторной работе, хранятся в разных форматах, поэтому для их совместного анализа формат данных должен быть единым. В этой работе Pig используется для выполнения предварительной обработки.

Предварительно обработанные данные используются для анализа с помощью Hive в следующем блоке.

## Работа в оболочке Grunt

### Создайте рабочий каталог для и перейдите в него:

$ mkdir ~/works

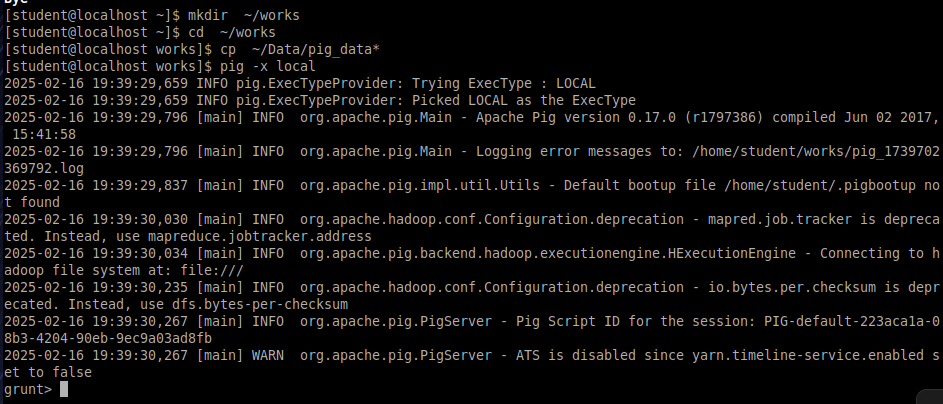
$ cd ~/works

### Скопируйте файлы pig\_data1.txt, pig\_data2.txt файлы в папке данных из домашнего каталога в рабочий каталог.

$ cp ~/Data/pig\_data\* .

### Запустите оболочку Grunt в локальном режиме.

$ pig -x local



### Загрузите данные файла data\_pig1.txt в оболочку и отобразите содержимое.

﻿grunt> tuples = LOAD 'pig\_data1.txt';

﻿grunt> dump tuples;

###### 텍스트이(가) 표시된 사진 자동 생성된 설명

Примечание: Вы можете просмотреть отдельные записи. Вы можете обнаружить записи, в которых отсутствуют какие-либо данные или есть ошибки.

﻿grunt> col\_three = load 'pig\_data1.txt' as (brand:chararray, date:chararray, model:chararray);

grunt> dump col\_three;



### Используйте команду DESCRIBE, чтобы увидеть схему col\_three;

﻿grunt> describe col\_three;

###### 

### Далее, создайте связь, которая удовлетворяет следующим условиям, используя тип данных и порядок полей по умолчанию.

#### Считайте pig\_data1.txt и найдите только записи со значением 6-го поля больше 1000 и выведите на экран результатов только значения 1-го, 2-го, 3-го и 6-го полей.

﻿grunt> data = load 'pig\_data1.txt';

﻿grunt> hi\_price = filter data by $5 > 1000;

﻿grunt> res = foreach hi\_price generate $0, $1, $2, $5;

﻿grunt> dump res;



#### Отобразите предыдущие результаты, исключив строки с пустыми значениями в первом или третьем поле.

﻿grunt> res\_notnull = filter res by $0 is not null and $2 is not null;

﻿grunt> dump res\_notnull;

grunt> quit;

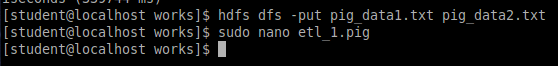
###### 텍스트이(가) 표시된 사진 자동 생성된 설명

## Обработка входных данных из сценария Pig

### Создайте файл сценария, etl\_1.pig, для проверки вашей команды в оболочке grunt.

#### Сначала, скопируйте данные pig\_data1.txt и pig\_data2.txt в папку домашнего каталога в hdfs.

$ hdfs dfs -put pig\_data1.txt pig\_data2.txt .



#### Создадим файл со сценарием и рассмотрим по шагам, что в нем написано:

$ sudo nano etl\_1.pig

data = LOAD 'pig\_data1.txt' AS (brand:chararray,

date:chararray,

model:chararray,

agent:chararray,

country:chararray,

price:int,

code:chararray);

data\_notnull = FILTER data BY brand is not null and model is not null and agent is not null;

price\_up = filter data\_notnull by price > 1000;

reordered = FOREACH price\_up GENERATE code,

UPPER(TRIM(brand)),

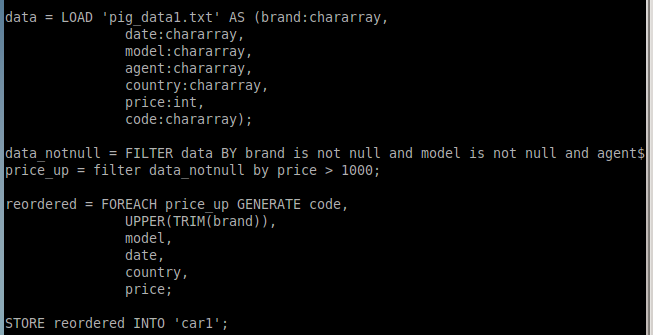
model,

date,

country,

price;

STORE reordered INTO 'car1';



#### Сначала мы загружаем данные файла pig\_data1.txt, используя приведенную ниже схему данных.

|  |  |  |
| --- | --- | --- |
| Index | Field Name | Data type |
| 0 | brand | chararray |
| 1 | date | chararray |
| 2 | model | chararray |
| 3 | agent | chararray |
| 4 | country | chararray |
| 5 | price | int |
| 6 | code | chararray |

#### Далее фильтруем все поля с нулевыми значениями для названий бренда, модели и агента.

#### После выбераем только те значения, цена которых 1000 и более и используем команду foreach для создания отношения в следующем порядке:

|  |  |  |
| --- | --- | --- |
| Index | Field Name | Data type |
| 0 | code | chararray |
| 1 | brand | chararray |
| 2 | model | chararray |
| 3 | date | chararray |
| 4 | country | chararray |
| 5 | price | int |

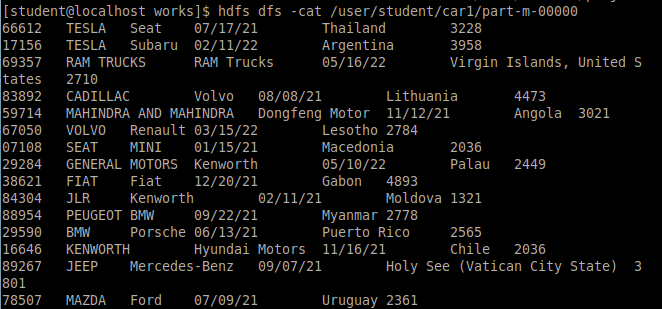
#### Вместе с тем изменияем поле brand в верхний регистр и удаляем все начальные и конечные пробелы. Это выполняется с помощью функций UPPER и TRIM

#### Наконец сохраняет результат отношения в качестве car1.

#### Запустите etl\_1.pig

$ ﻿pig etl\_1.pig

#### Посмотрим результат в файле car/part-m-00000



## Самостоятельно создайте файл сценария для предварительной обработки

### Создайте файл сценария etl\_2.pig для обработки данных в pig\_data2.txt.



#### Сначала, загрузите данные файла pig\_data2.txt используя приведенную ниже схему данных.

|  |  |  |
| --- | --- | --- |
| Index | Field Name | Data type |
| 0 | brand | chararray |
| 1 | date | chararray |
| 2 | model | chararray |
| 3 | agent | chararray |
| 4 | country | chararray |
| 5 | price | chararray |
| 6 | code | chararray |

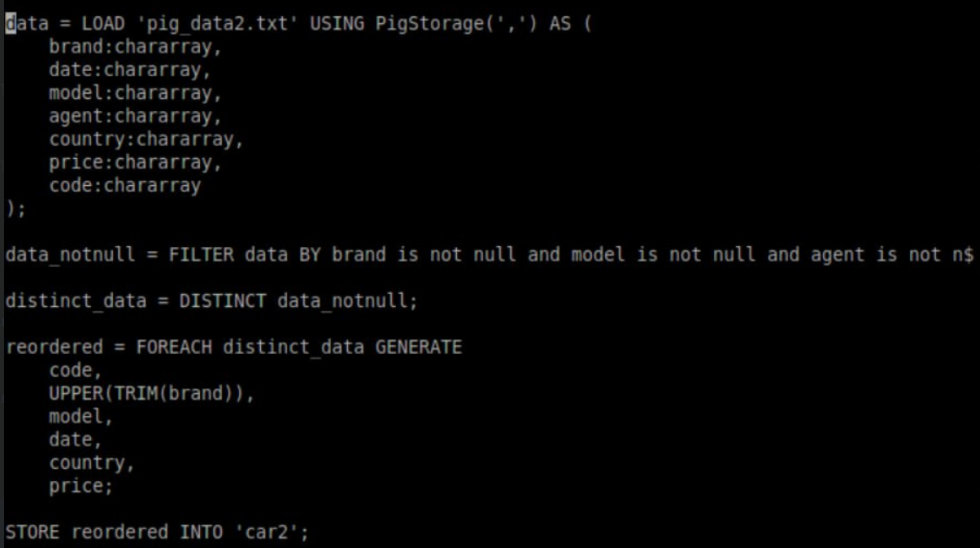
#### Отфильтруйте все поля с нулевыми значениями для названия бренда, модели и агента.

#### Повторяющиеся записи с одинаковыми значениями удалены (используйте оператор DISTINCT).

#### Измените поле brand в верхний регистр и удалите все начальные или конечные пробелы.

#### Создайте отношение со следующим порядком полей:

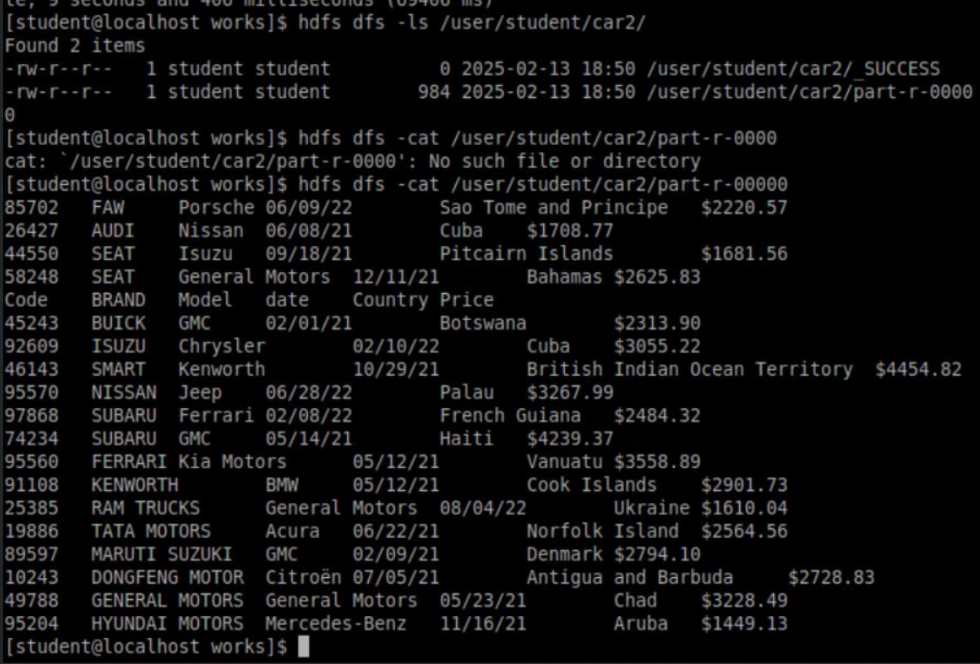
|  |  |  |
| --- | --- | --- |
| Index | Field Name | Data type |
| 0 | code | chararray |
| 1 | brand | chararray |
| 2 | model | chararray |
| 3 | date | chararray |
| 4 | country | chararray |
| 5 | price | chararray |



#### Сохраните результат в том же формате, что и etl\_1.pig в папке car2 в hdfs.

Подсказка: Поскольку в файле pig\_data2, каждое поле разделено символом ",", вы должны использовать команду USING PigStorage для его разделения при начальной загрузке:

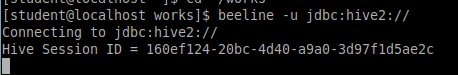
data = LOAD 'pig\_data2.txt' USING PigStorage(',') AS …



# Выполнение базовых запросов с помощью Hive QL

## В окне терминала, запустите Beeline.

﻿$ beeline –u jdbc:hive2://



### Если beeline запущен, то появится приглашение для ввода “﻿0: jdbc:hive2://> “ и отобразится экран ожидания команд. Введите команду, чтобы проверить, какие базы данных существуют:

###### 

### Далее, введите команду для отображения таблицы в базе данных по умолчанию.

###### 텍스트, 전자기기이(가) 표시된 사진 자동 생성된 설명

### Создайте тестовую базу данных и выведите список баз данных.

﻿0: jdbc:hive2://> create database test;

﻿0: jdbc:hive2://> show databases;

###### 텍스트이(가) 표시된 사진 자동 생성된 설명

Примечение: Созданная база данных сохраняется в виде папки в каталоге hdfs.

###### 

### Удалите созданную тестовую базу данных и проверьте результат.

﻿0: jdbc:hive2://> drop database test;

﻿0: jdbc:hive2://> show databases;

###### 텍스트이(가) 표시된 사진 자동 생성된 설명

### Создайте таблицу products. Будем использовать следующую схему:

|  |  |
| --- | --- |
| Column Name | Column Type |
| code | int |
| brand | string |
| l\_date | string |
| model | string |
| country | string |
| price | int |

﻿

0: jdbc:hive2://> create table if not exists products (

code int,

brand string,

l\_date string,

model string,

country string,

price string

)

row format delimited fields terminated by ‘\t’;

### Проверьте таблицу products с помощью команды desc и проверьте файловую систему hdfs (для выхода из beeline используйте команду !q).

﻿﻿0: jdbc:hive2://> show tables;

0: jdbc:hive2://> desc products;

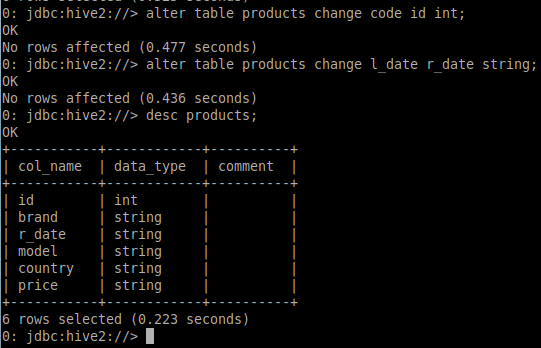
###### 텍스트이(가) 표시된 사진 자동 생성된 설명

###### 

### Измените имя поля l\_date на r\_date, а имя поля code измените на id.

﻿0: jdbc:hive2://> alter table products change code id int;

﻿0: jdbc:hive2://> alter table products change l\_date r\_date string;



### Переименуйте таблицу products на test.

﻿0: jdbc:hive2://> alter table products rename to test;

###### Примечение: При выполнении команды alter имя папки hdfs также изменяется с products на test.

###### 텍스트이(가) 표시된 사진 자동 생성된 설명

### Удалите таблицу test и проверьте каталог hdfs в /user/hive/warehouse.

﻿0: jdbc:hive2://> drop table test;

###### 텍스트, 오렌지, 설정이(가) 표시된 사진 자동 생성된 설명

Примечание: Когда управляемая таблица удаляется, папка в файловой системе hdfs также удаляется.

Создание и загрузка таблицы с помощью оболочки

### Прежде чем использовать оболочку, создайте каталог для использования в лабораторной как /mywarehouse в файловой системе hdfs и измените владельца созданной папки на student.

### 

### И данные, предварительно обработанные с помощью pig в предыдущей лабораторной, сохраните в созданной папке. Для этого выполните следующую команду:

$ hdfs dfs -mv car1 car2 /mywarehouse

### Создайте и загрузите таблицу с помощью редактора и выполните сценарий с опцией оболочки -f.

#### Для этого напишите следующий сценарий:

$sudo nano products.sql

﻿﻿create external table if not exists products (

code int,

brand string,

model string,

r\_date string,

country string,

price int

)

row format delimited

fields terminated by ‘\t’

location ‘/mywarehouse/car1’;

#### Здесь мы создаем внешнюю таблицу с помощью CREATE TABLE из файла, который мы создали в предыдущем блоке с использованием Pig. Схема созданной таблицы приведена приведена ниже:

|  |  |
| --- | --- |
| Column Name | Column Type |
| code | int |
| brand | string |
| model | string |
| r\_date | string |
| country | string |
| price | int |

#### Данные разделяются символом ‘\t’ для каждой сохраненной записи в каталоге /mywarehouse/car1 в файловой системе hdfs.

#### Запустите products.sql с помощью –f.

﻿$ beeline –u jdbc:hive2:// -f products.sql

###### 텍스트이(가) 표시된 사진 자동 생성된 설명

### Проверьте и удалите таблицу products созданную сценарием. Завершите beeline.

﻿ ﻿$ beeline –u jdbc:hive2://

0: jdbc:hive2://> desc products;

0: jdbc:hive2://> select \* from products;

﻿0: jdbc:hive2://> drop table products;

﻿0: jdbc:hive2://> !quit

(exit the beeline)

$

###### 텍스트이(가) 표시된 사진 자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

**Примечание**: Когда таблица, созданная как внешняя, удаляется, информация таблицы удаляется из метахранилища, но данные car1 остаются.

## Создайте таблицу с помощью Sqoop Hive import

### Выполните следующую команду в терминале, чтобы импортировать таблицу authors из MariaDB с помощью опции hive-import.

#### Сначала, войдите в beeline, и создайте базу данных mydb для лабораторной.

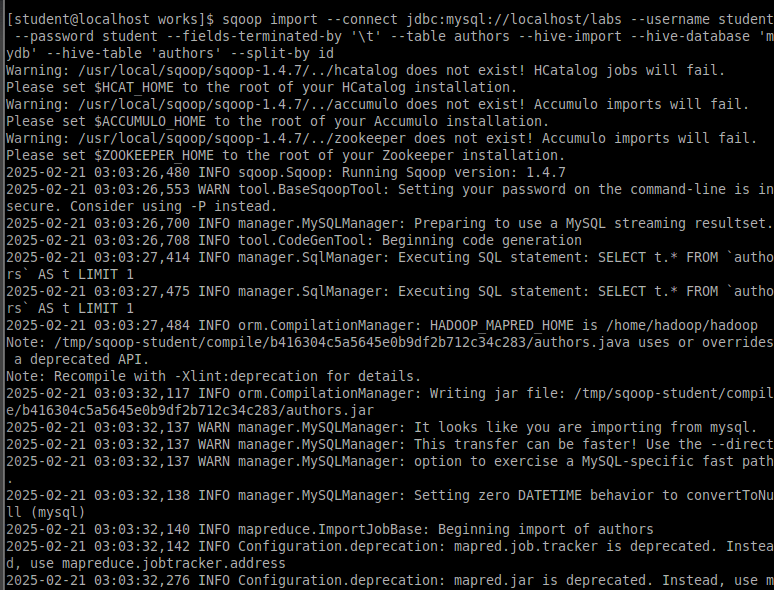
﻿0: jdbc:hive2://> create database mydb;

### **Примечпние**: Созданная база данных mydb создается в /user/hive/warehouse с названием mydb.db и отличается от общих таблиц.

#### В другом терминале, запустите команду sqoop, как показано здесь:

﻿$ hdfs dfs –rm –r authors

$ sqoop import --connect jdbc:mysql://localhost/labs --username student --password student --fields-terminated-by ‘\t’ --table authors --hive-import --hive-database ‘mydb’ --hive-table ‘authors’ --split-by id

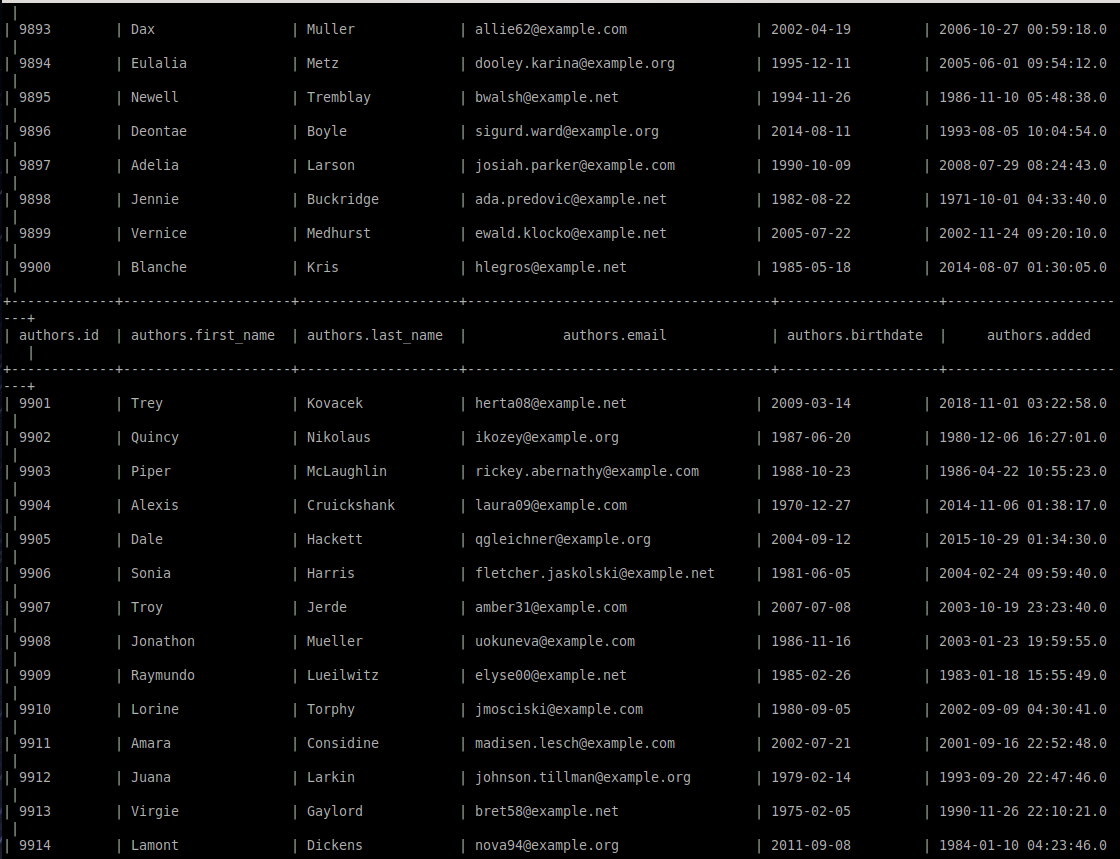


### Убедитесь из beeline, что таблица импортирована и правильно сохранена. Проверка 10000 записей.

﻿0: jdbc:hive2://> select \* from mydb.authors;

0: jdbc:hive2://> use mydb;

0: jdbc:hive2://> select count(\*) from authors;



###### 텍스트이(가) 표시된 사진 자동 생성된 설명

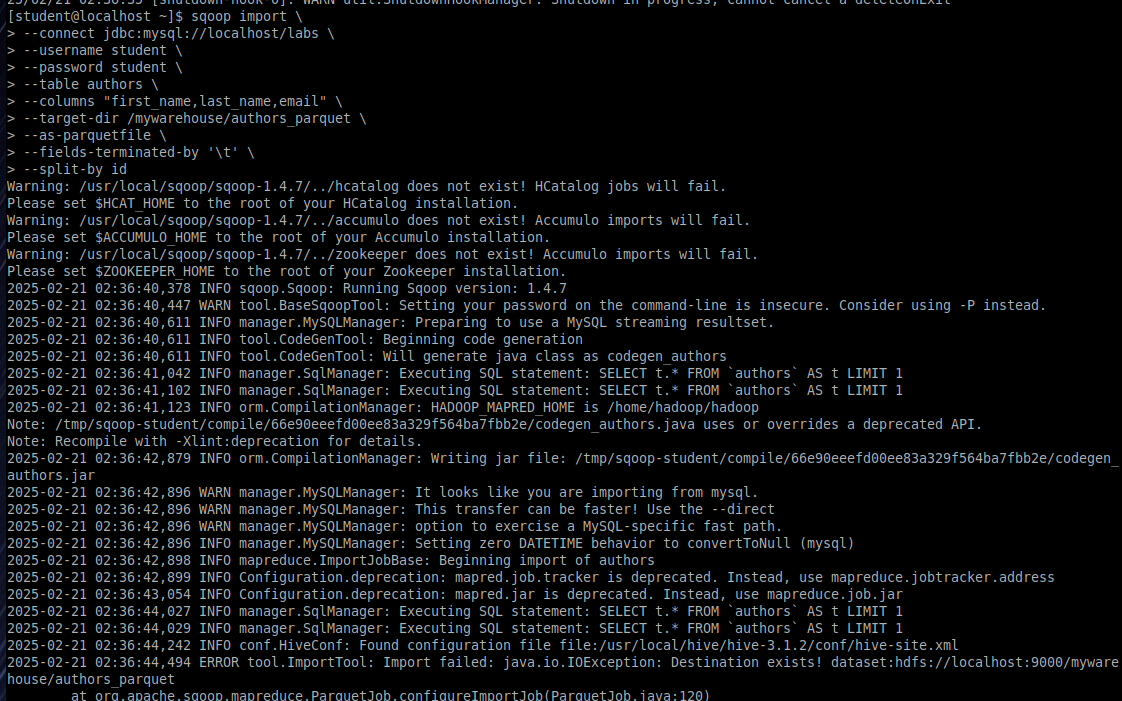
텍스트이(가) 표시된 사진

자동 생성된 설명

## Напишите команды hiveQL, которые выполняют следующие условия.

### Сохраните таблицу posts из MariaDB с помощью sqoop в базе данных mydb и найдите общее количество записей.

### Импортируйте с помощью sqoop из таблицы authors в MariaDB только first\_name, last\_name, email и сохраните результаты в папку authors\_parquet в формате parquet в /mywarehouse в hdfs. Затем создайте внешнюю таблицу authors\_parquet в Hive с помощью данной папки. В качестве типа данный для даты используйте string, для timestamp - int;



### С помощью alter table переименуйте “first\_name” столбец в “fname” в authors\_parquet. Проверьте изменения с помощью DESCRIBE.

### Переименуйте всю таблицу в authors\_parquet2.

