DWH

Лекция №5:

Hadoop. Hive.

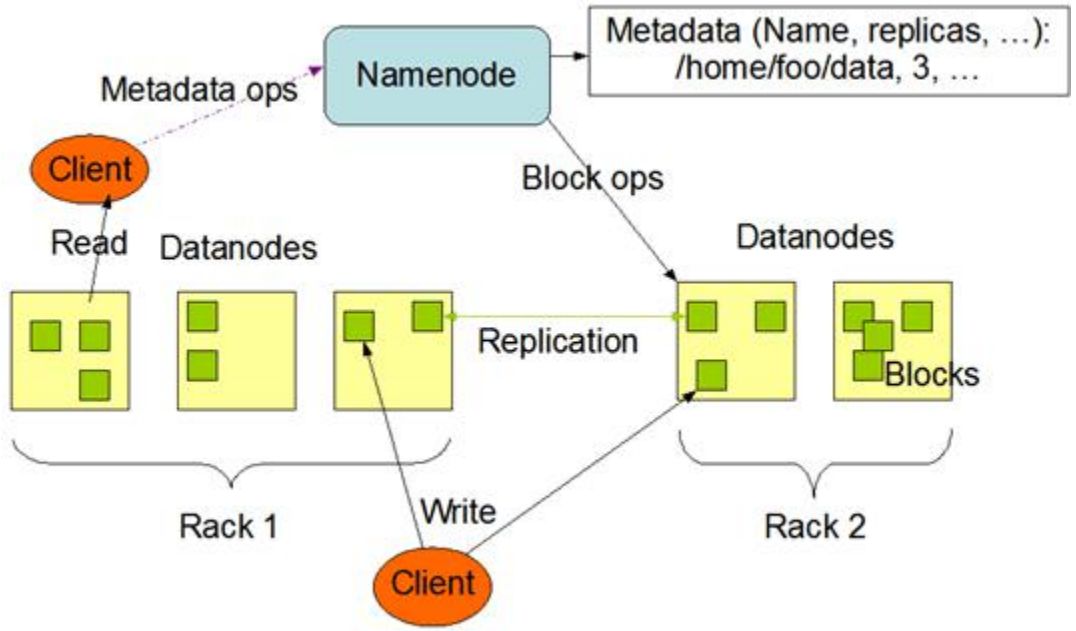
В прошлой лекции

• • •

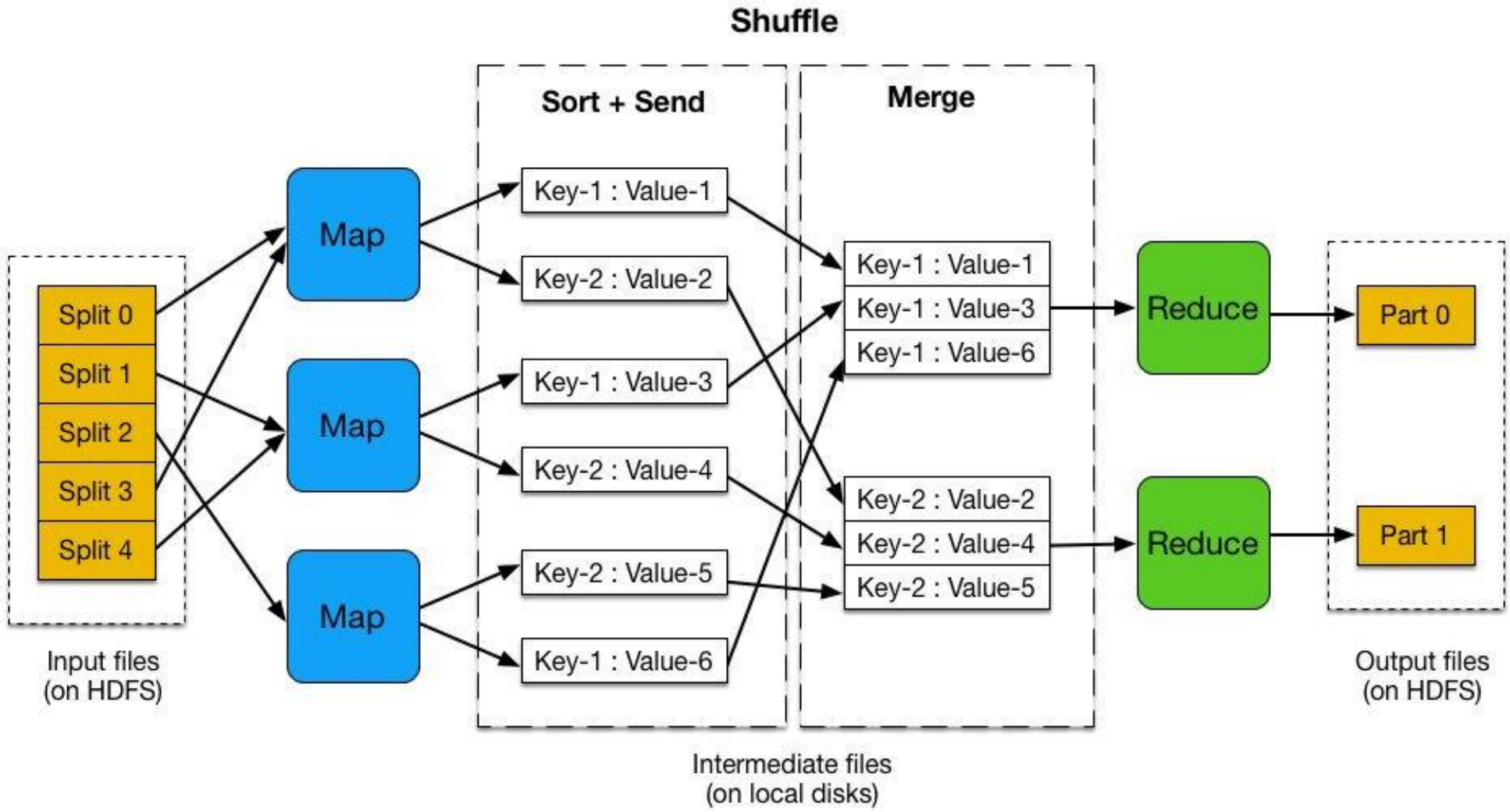
Архитектура HDFS

NameNode – сервер, управляющий пространством имен файловой системы (знает где и что лежит) и доступом клиентов к данным.

DataNodes – сервера, которые непосредственно хранят данные.



MapReduce


Shuffle
Sort + Send
Merge
Split 0
Split 1
Split 2
Split 3
Split 4
Map
Map
Map
Key-1 : Value-1
Key-2 : Value-2
Key-1 : Value-3
Key-2 : Value-4
Key-2 : Value-5
Key-1 : Value-6
Key-1 : Value-1
Key-1 : Value-3
Key-1 : Value-6
Key-2 : Value-2
Key-2 : Value-4
Key-2 : Value-5
Reduce
Reduce
Part 0
Part 1
Input files
(on HDFS)
Intermediate files
(on local disks)
Output files
(on HDFS)

Hive

Hive

Apache Hive — система управления базами данных на основе платформы Hadoop. Позволяет работать с данными, хранящимися в HDFS в привычной многим парадигме SQL.

Hive. Особенности

- SQL-подобный язык (HQL) вместо сложного MapReduce
- Интерактивная консоль (command line interface)
- Встроенные функции агрегации, математические, статистические функции и т.д. Подробнее читайте в документации.
- Поддержка пользовательских функций (UDF)
- Данные — как таблица
- Выполнение запросов через MapReduce (по умолчанию), Apache Spark или Apache Tez

Hive. Особенности

WC на Hive

```
[drop table if exists docs;]

create table docs (line string)
stored as textfile;

load data inpath '/hdfs/path/'
overwrite into table docs;

select word, count(1) as cnt
from (
    select explode(
        split(line, '\s')
    ) as word
    from docs
) temp group by word order by
word;
```

WC на mapReduce

```
import java.io.IOException;
import java.util.StringTokenizer;

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.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Hive. Основные сущности

При работе с hive можно выделить следующие объекты которыми он оперирует:

- База данных
- Таблица
- Партиция (partition)

База данных является аналогом базы данных в реляционных СУБД. Представляет собой пространство имён, содержащее таблицы.

```
create database [if not exists] <database name> [LOCATION <path>];
```

Hive. Таблицы

Таблица в hive представляет из себя аналог таблицы в классической реляционной БД. Основное отличие — что данные хранятся в виде обычных файлов на hdfs. Это могут быть обычные текстовые файлы (csv, json), бинарные sequence-файлы, более сложные колоночные (parquet, orc) и другие форматы.

Таблицы в hive бывают двух видов:

- Классическая таблица, данные в которую добавляются при помощи hive.
- Внешняя таблица, данные в которую загружаются внешними системами.

Hive. External таблицы

При создании внешней таблицы нужно указать ключевое слово EXTERNAL, указать путь расположения данных и описать формат хранения.

После этого таблицей можно пользоваться точно так же как и обычными таблицами hive. Самое удобное в этом то, что вы можете просто скопировать файл в нужную директорию в hdfs, а hive будет автоматически подхватывать новые файлы при запросах к соответствующей таблице.

Однако партиции надо создавать вручную.

```
create external table employee_external (
    id int,
     name string,
     salary double
 ) comment 'employee details'
 row format delimited
 fields terminated by '\t'
 stored as textfile
 location '/hdfs/data/external_files/';
```

Hive. Партиционирование

Каждой партиции соответствует отдельная директория на hdfs. Это означает, что данные относящиеся к разным значениям будут физически храниться в разных директориях на HDFS. Теперь, если мы будем запускать какие-либо запросы, указав в условии WHERE ограничение на значения партиций — hive возьмет входные данные только из соответствующих директорий, а не будет сканировать таблички целиком.

```
...
/user/hive/warehouse/website_log/dt=20181020/
/user/hive/warehouse/website_log/dt=20181021/
...
```

```
create table website_logs (
    ip string,
     action string
) comment ‘daily site log’
partition by (
    dt string
) stored as orc;
```

Hive. UDF

Одним из основных препятствий при работе с Hive может стать скованность рамками стандартного SQL.

В hive достаточно много встроенных функций, но если вам ничего не подходит, то эту проблему можно решить при помощи использования расширений языка — так называемых **User Defined Functions.**

Делается это на языке java, ниже представлен пример собственной UDF-функции преобразования строки в lowercase:

```
package com.example.hive.udf;

import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.io.Text;

public final class Lower extends UDF {
  public Text evaluate(final Text s) {
   if (s == null) { return null; }
   return new Text(s.toString().toLowerCase());
  }
}
```

Подробнее читайте в документации.

Hive. Выводы

Плюсы:

- SQL-подобный язык
- MapReduce под капотом. Уходит много лишней работы, связанной с написанием MR-задач: описание моделей данных, входных и выходных форматов, цепочек MR-задач.
- Интерактивность. Хорош для анализа данных в разных срезах.
- Быстрота разработки
- Отсутствие зависимостей, компиляции, сборки

Минусы:

- Не всё хорошо ложится в парадигму SQL

Форматы файлов

. . .

RCFile

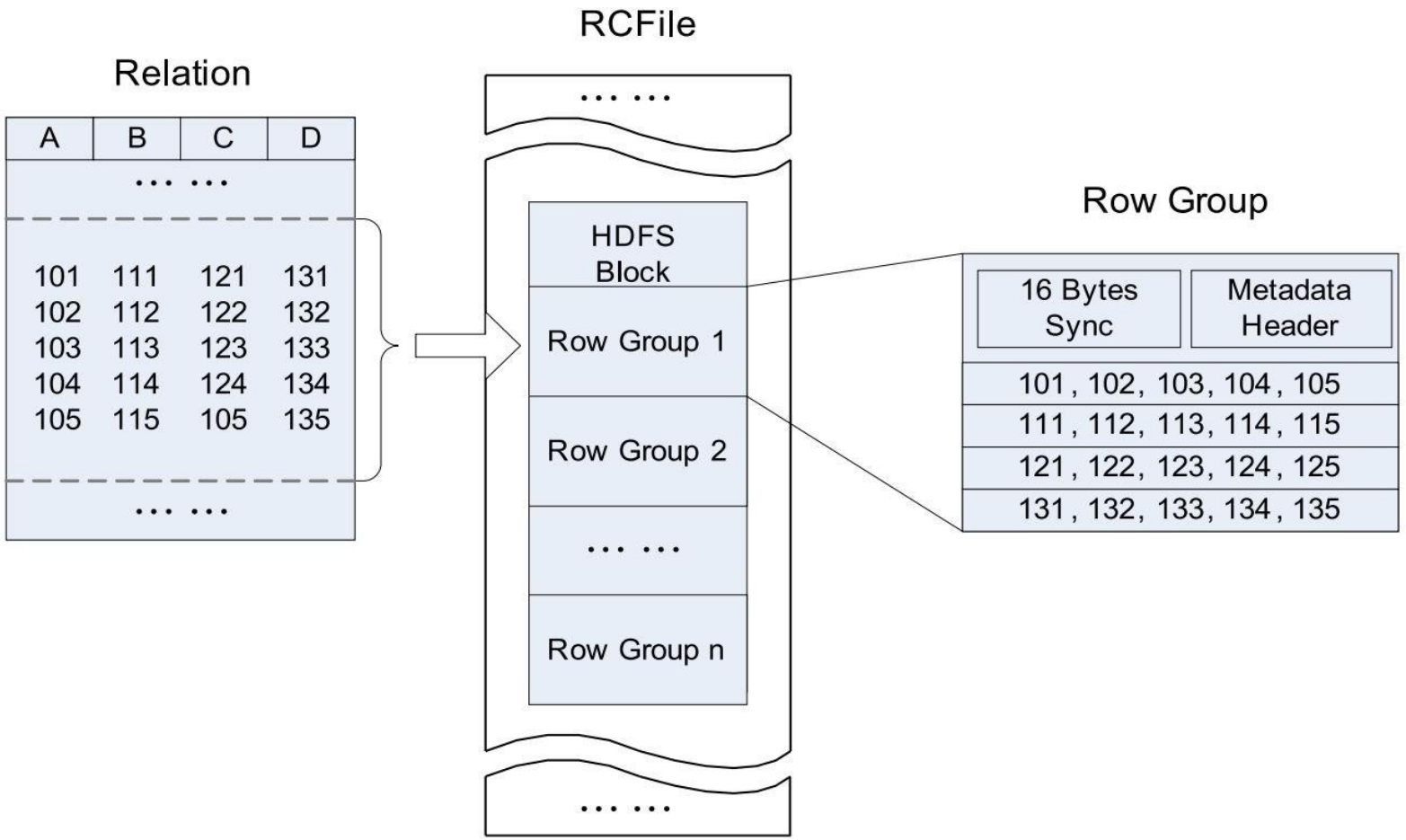
RCFile (Record Columnar file) – колончатый гибридный формат, разработанный для вычислений на базе MapReduce.

Записи разделяются “горизонтально” на группы и “вертикально” внутри групп.

Особенности:

- Гарантируется, что все поля одной строки находятся на одной ноде
- Поколончатое сжатие
- При чтении ненужные колонки могут быть пропущены

RCFile



Sync - разделяет группы записей

Metadata Header - содержит информацию о числе записей в группе, размеры (в байтах) колонок и полей

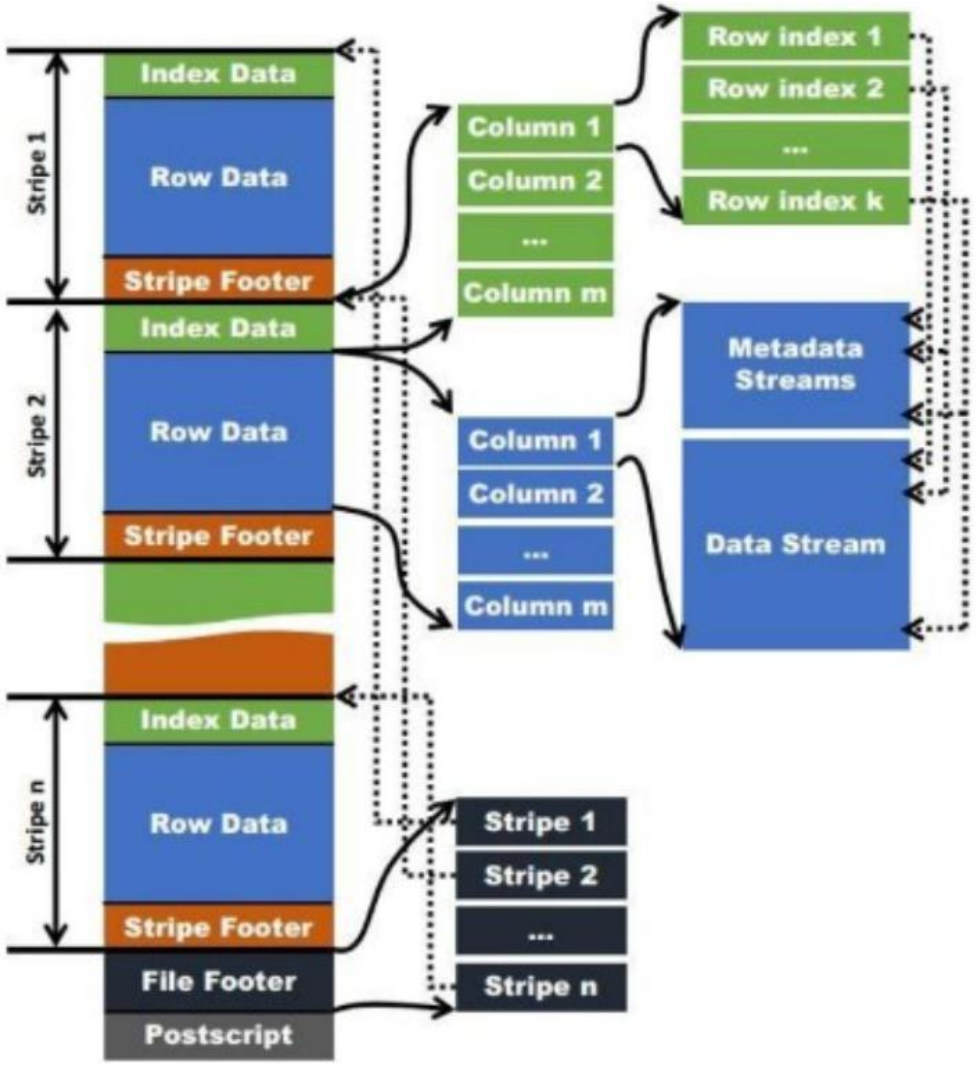
ORC

Optimized RC – Развитие RCFile, файлы хранят широкий набор метаданных (для хранения используется Protobuf).

Особенности:

- Хранит данные о типах, полная совместимость с типами в Hive
- Трехуровневая индексация (файл, страйп, запись)
- Поддержка pushdown-фильтров
- Информация о типах позволяет оптимизировать сжатие

ORC



Postscript - информация о сжатии, размер Footer'а

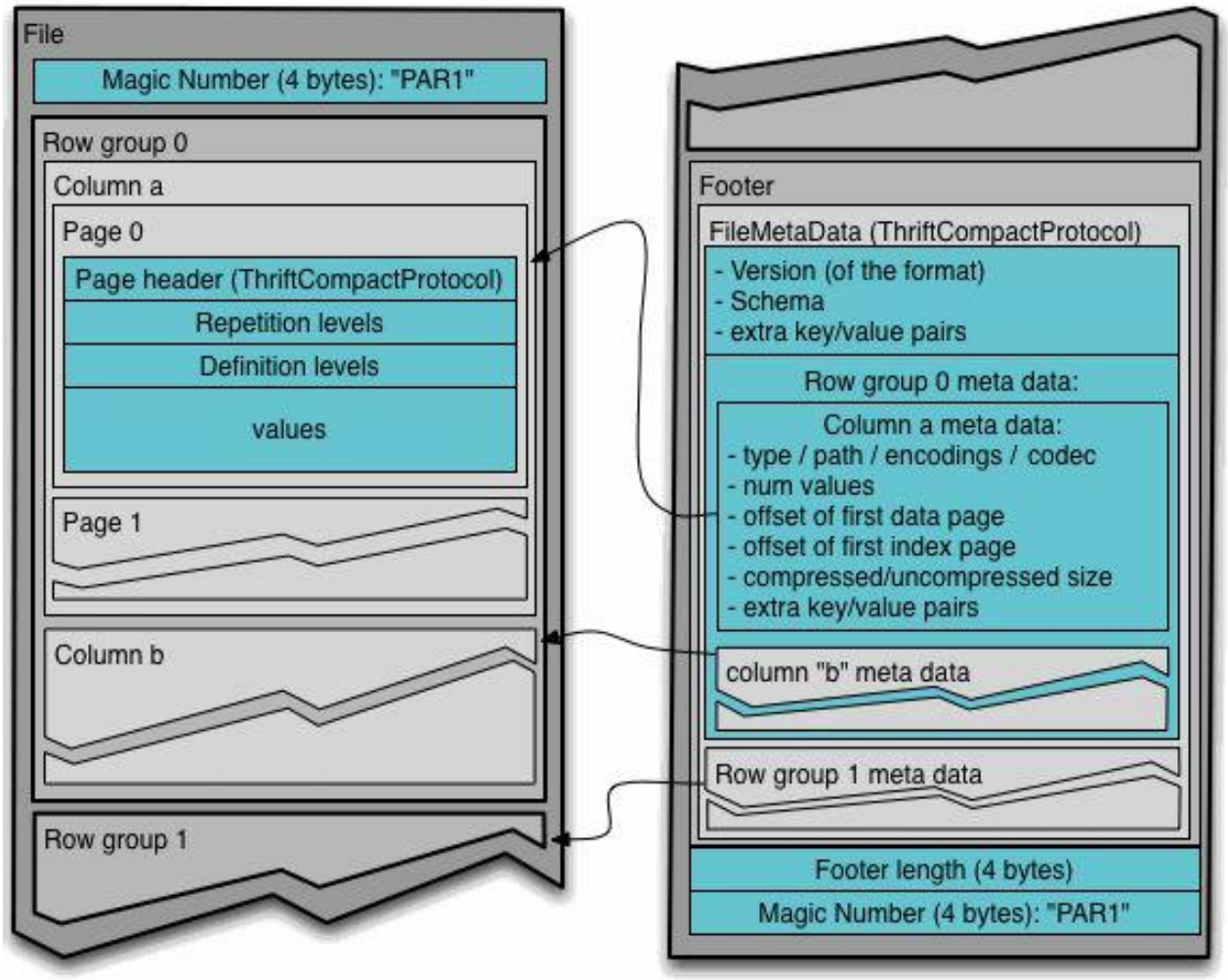
File footer - список stripe'ов, число записей в них. Поколоночные аггрегаты (среднее, макс/мин, сумма)

Stripe footer - справочник "откуда читать"

Index data - мин/макс значения в колонке, позиции строк в столбцах.

Header (не показан, 3 байта) - "ORC"

Parquet



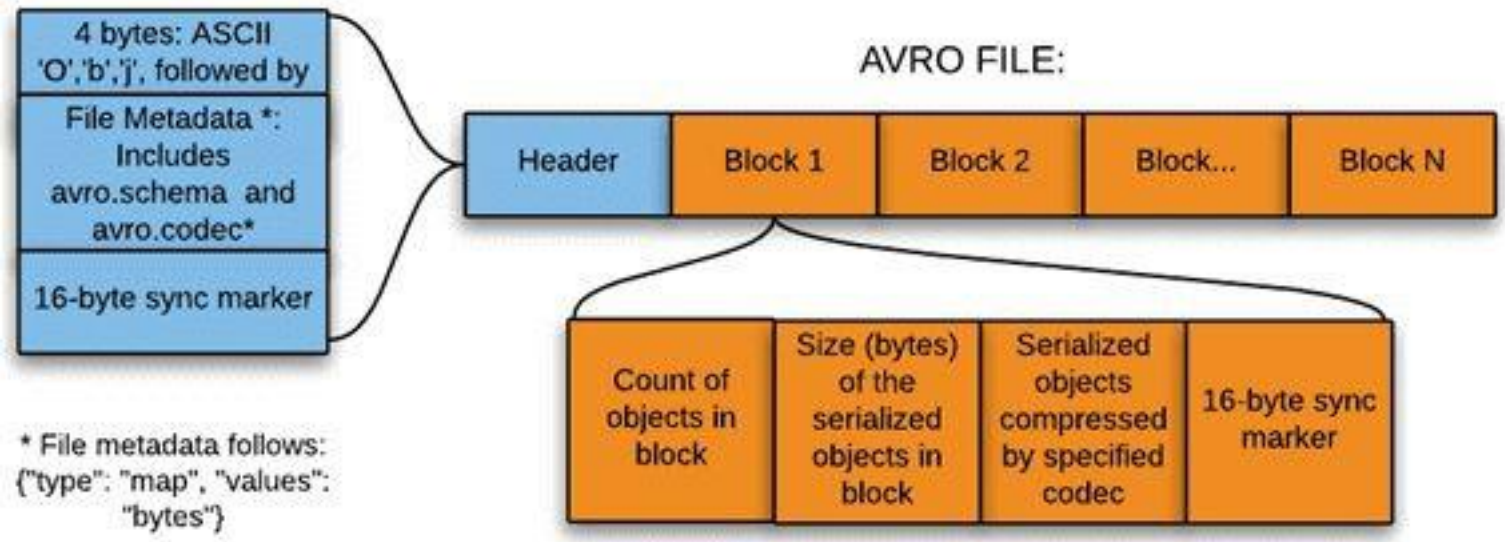
- В иерархии появляются “страницы”
- Большая часть метаданных в футере

Avro

Особенности:

- Последовательное представление записей
- Хранит схему данных в json
- Типизированные поля
- Объекты (записи) подлежат сжатию
- Есть возможность записать блоки с данными в человекочитаемом, а не бинарном формате (например для отладки)

Avro


4 bytes: ASCII 'O','b','j', followed by
File Metadata *: Includes avro.schema and avro.codec*
16-byte sync marker
* File metadata follows: {"type": "map", "values": "bytes"}
AVRO FILE:
Header
Block 1
Block 2
Block...
Block N
Count of objects in block
Size (bytes) of the serialized objects in block
Serialized objects compressed by specified codec
16-byte sync marker

Практика Hive

Пользовательская БД

Команда для запуска CLI:

```
hive
```

Создаем базу (предварительно создав директорию в hdfs):

```
create database user_%username%
location '/home/%username%/warehouse/';
```

Права у вас есть только на базу user_%username% (подставьте свой логин без знаков %):

```
use user_%username%;
```

Пример WC

```
1.  [drop table if exists docs;]

2.   create table docs (
3.       line string
4.   ) stored as textfile;

5.  load data inpath '/home/%username%/data/data1/wctest.txt'
6.  overwrite into table docs;

7.  select word, count(1) as cnt from (
8.    select explode(
9.      split(regexp_replace(lower(line), '[^a-z ]', ''), ' ')
10.   ) as word
11.   from docs
12. ) temp group by word order by word;
```

External table

Создаем внешнюю таблицу:

```
1. create external table books (
2.   line string
3. ) partitioned by (book_name string)
4. stored as textfile
5.    location '/home/%username%/data/data1/books';
```

Указываем партиции:

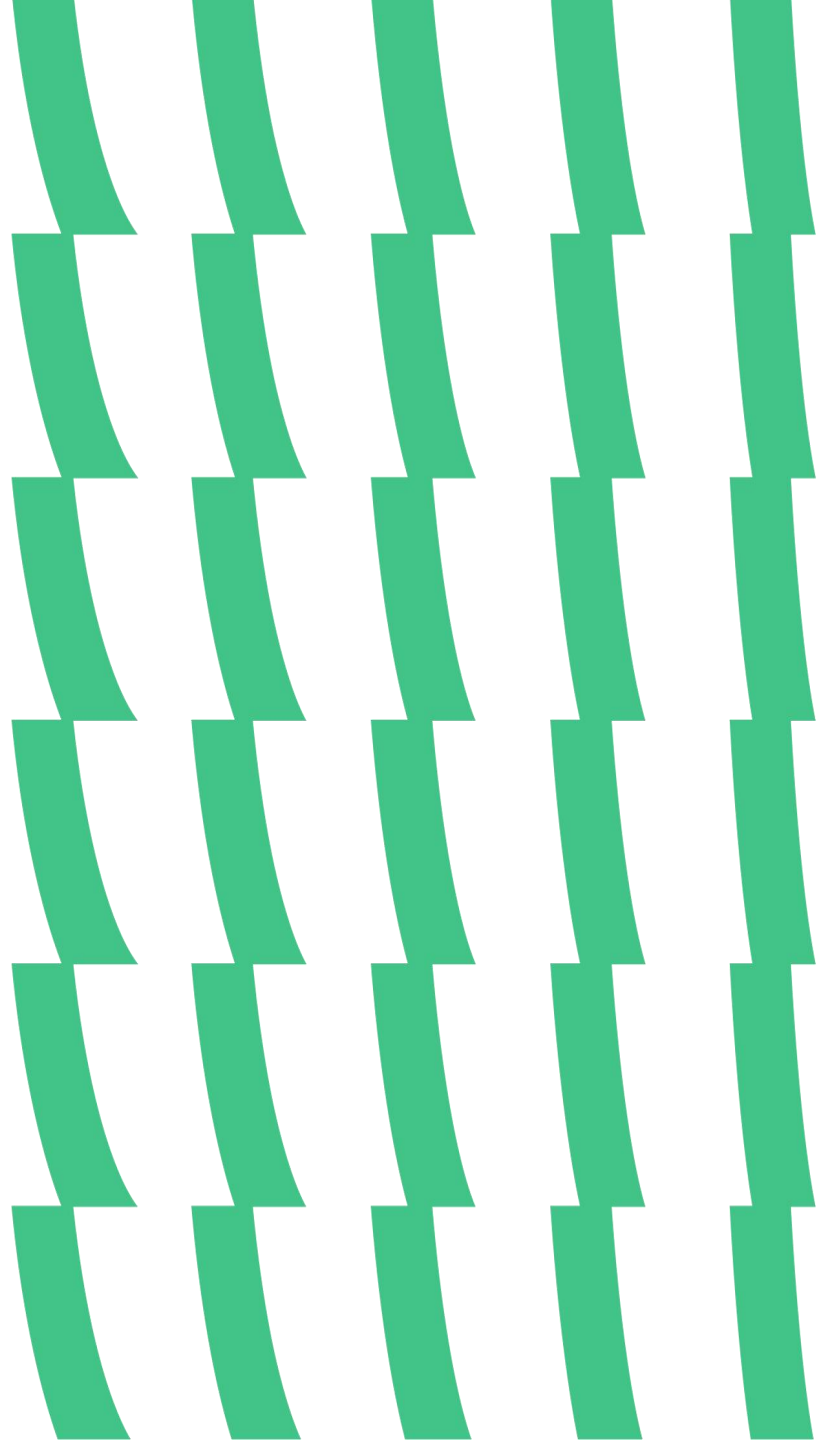
```
1. alter table books add partition (book_name='book1')
2. location '/home/%username%/data/data1/books/book1';

3. alter table books add partition (book_name='book2')
4. location '/home/%username%/data/data1/books/book2';

5. alter table books add partition (book_name='book3')
6. location '/home/%username%/data/data1/books/book3';
```

Пример Top N

```
1. select
2.   word,
3.   count(1) as cnt
4. from (
5.   select
6.     explode(
7.       split(
8.         regexp_replace(lower(line), '[^a-z ]', ''),
9.         ' '
10.      )
11.    ) as word
12.  from books
13. ) temp
14. where word != ''
15. group by word
16. order by cnt desc
17. limit 10;
```



Join

На hdfs создаем директории под таблицы и копируем в них данные.

Создаем внешнюю таблицу цен:

```
1. create external table shop_price (
2.   id int,
3.   price float
4. )
5. row format delimited
6. fields terminated by ';'
7. lines terminated by '\n'
7. location '/home/%username%/warehouse/shop_price';
```

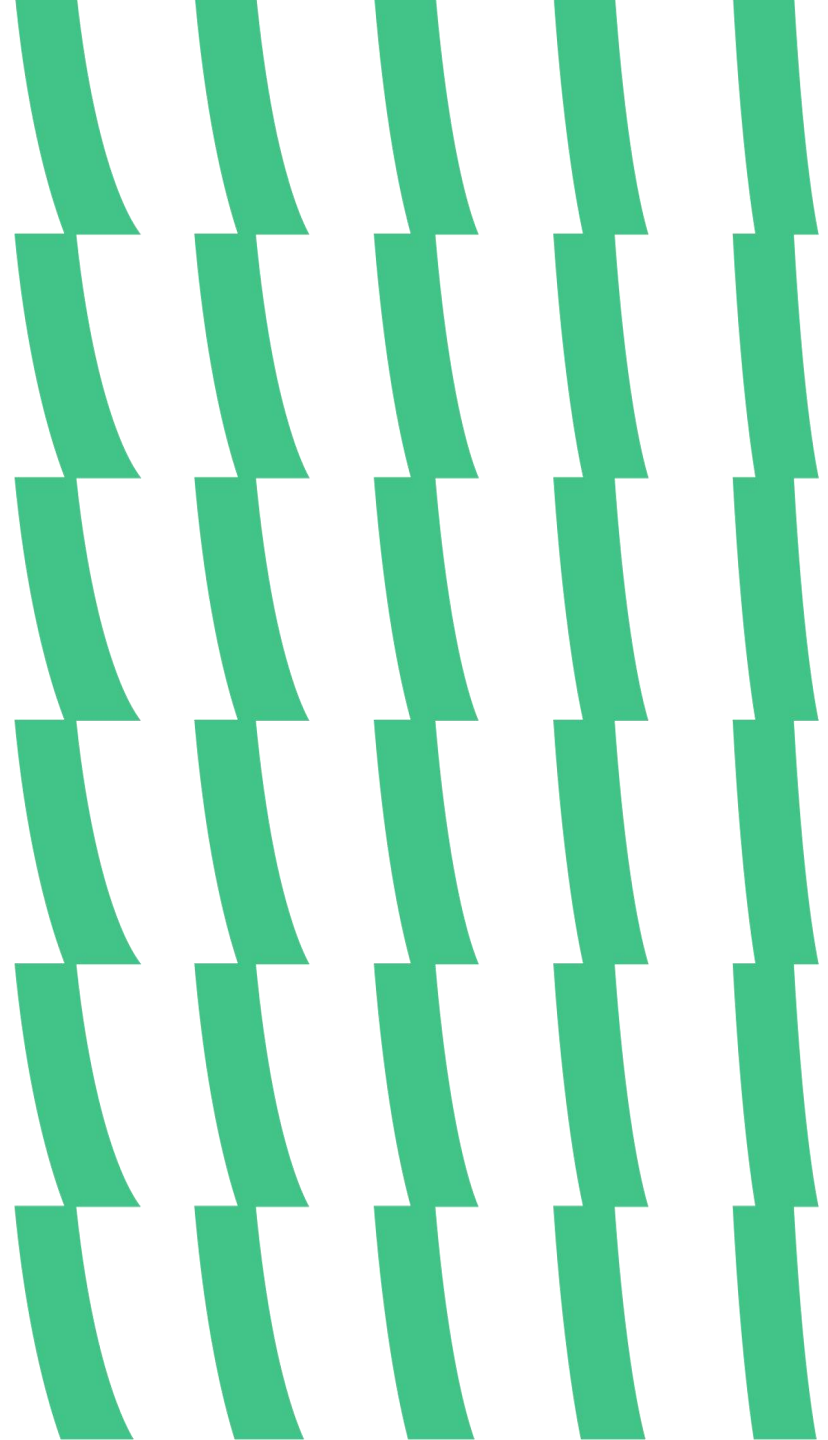
Создаем внешнюю таблицу товаров:

```
1. create external table shop_product (
2.   id int,
3.   description string
4. )
5. row format delimited
6. fields terminated by '\t'
7. lines terminated by '\n'
7. location '/home/%username%/warehouse/shop_product';
```

Join

Выполняем сам джойн:

```
select
  shop_price.id,
  description,
  price
from shop_product
join shop_price on (
  shop_product.id == shop_price.id
);
```



Домашнее задание 2

Задача: Создать внешнюю таблицу на данных coreDemography и написать запрос рассчитывающий распределение дней рождения по месяцам. В результате должна получится таблица из двух колонок (month, cnt) следующего содержания:

month	cnt
NULL	9
1	10297
2	8496
3	9242
4	8875
5	9051
6	9332
7	9534
8	9228
9	8653
10	8534
11	8061
12	8253

coreDemography содержит данные демографии пользователей ОК - архив к занятию - hadoop2.zip/data2/coreDemography, также есть info.txt с описанием полей. Решение должно содержать файлы с кодом на создание внешней таблицы (в своей базе данных) и сам запрос. В случае успешного решения присылайте решение на портале до 2024-04-02 17:59:59

Спасибо за внимание!