Hive

OutLine

- Hive基础
- 【实践】Hive搭建
- 【实践】Hive练习

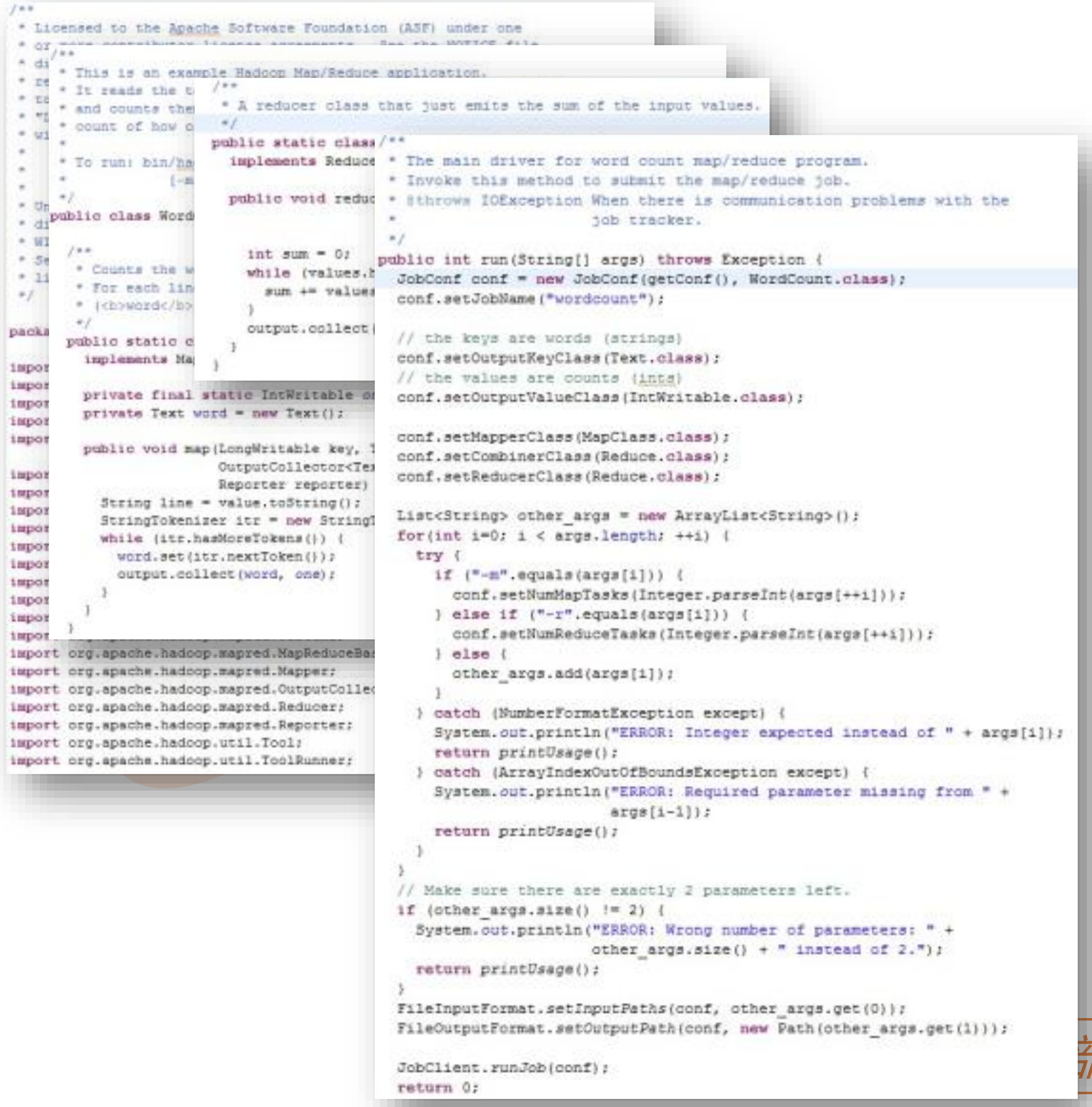
背景

- 引入原因：
  - 对存在HDFS上的文件或HBase中的表进行查询时，是要手工写一堆MapReduce代码
  - 对于统计任务，只能由动MapReduce的程序员才能搞定
  - 耗时耗力，更多精力没有有效的释放出来
- **Hive基于一个统一的查询分析层，通过SQL语句的方式对HDFS上的数据进行查询、统计和分析**

Hive是什么

- Hive是一个SQL解析引擎，将SQL语句转译成MR Job，然后再Hadoop平台上运行，达到快速开发的目的。
- Hive中的表是纯逻辑表，就只是表的定义等，即表的元数据。本质就是Hadoop的目录/文件，达到了元数据与数据存储分离的目的
- Hive本身不存储数据，它完全依赖HDFS和MapReduce。
- Hive的内容是读多写少，不支持对数据的改写和删除
- Hive中没有定义专门的数据格式，由用户指定，需要指定三个属性：
  - 列分隔符
  - 行分隔符
  - 读取文件数据的方法

为什么选择Hive



```
/**
 * Licensed to the Apache Software Foundation (ASF) under one
 * or
 * di
 * re
 * tc
 * "L
 * wi
 *
 *
 *
 * Un
 * di
 * WI
 * Se
 * li
 */

packa

impor
impor
impor
impor
impor

impor
impor
impor
impor
impor
impor
impor
impor
impor
impor
import org.apache.hadoop.mapred.MapReduceBa
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputColle
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
```

```
/**
 * This is an example Hadoop Map/Reduce application.
 * It reads the t
 * and counts the
 * count of how o
 *
 * To run: bin/ha
 *            [-m
 */
public class Word

  /**
   * Counts the w
   * For each lin
   * (<b>word</b>
   */
  public static c
    implements Ma

    private final static IntWritable o
    private Text word = new Text();

    public void map(LongWritable key, T
                    OutputCollector<Te
                    Reporter reporter)
      String line = value.toString();
      StringTokenizer itr = new StringT
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        output.collect(word, one);
      }
    }
  }
```

```
/**
 * A reducer class that just emits the sum of the input values.
 */
public static class
  implements Reduce

  public void reduc


    int sum = 0;
    while (values.h
      sum += values
    }
    output.collect(
  }
}
```

```
/**
 * The main driver for word count map/reduce program.
 * Invoke this method to submit the map/reduce job.
 * @throws IOException When there is communication problems with the
 *                     job tracker.
 */
public int run(String[] args) throws Exception {
  JobConf conf = new JobConf(getConf(), WordCount.class);
  conf.setJobName("wordcount");

  // the keys are words (strings)
  conf.setOutputKeyClass(Text.class);
  // the values are counts (ints)
  conf.setOutputValueClass(IntWritable.class);

  conf.setMapperClass(MapClass.class);
  conf.setCombinerClass(Reduce.class);
  conf.setReducerClass(Reduce.class);

  List<String> other_args = new ArrayList<String>();
  for(int i=0; i < args.length; ++i) {
    try {
      if ("-m".equals(args[i])) {
        conf.setNumMapTasks(Integer.parseInt(args[++i]));
      } else if ("-r".equals(args[i])) {
        conf.setNumReduceTasks(Integer.parseInt(args[++i]));
      } else {
        other_args.add(args[i]);
      }
    } catch (NumberFormatException except) {
      System.out.println("ERROR: Integer expected instead of " + args[i]);
      return printUsage();
    } catch (ArrayIndexOutOfBoundsException except) {
      System.out.println("ERROR: Required parameter missing from " +
                         args[i-1]);
      return printUsage();
    }
  }
  // Make sure there are exactly 2 parameters left.
  if (other_args.size() != 2) {
    System.out.println("ERROR: Wrong number of parameters: " +
                       other_args.size() + " instead of 2.");
    return printUsage();
  }
  FileInputFormat.setInputPaths(conf, other_args.get(0));
  FileOutputFormat.setOutputPath(conf, new Path(other_args.get(1)));

  JobClient.runJob(conf);
  return 0;
```

```
select word, count(*)
from (
select
explode(split(sentence, ' ')) as word
from article
) t
group by word
```

简单！！！

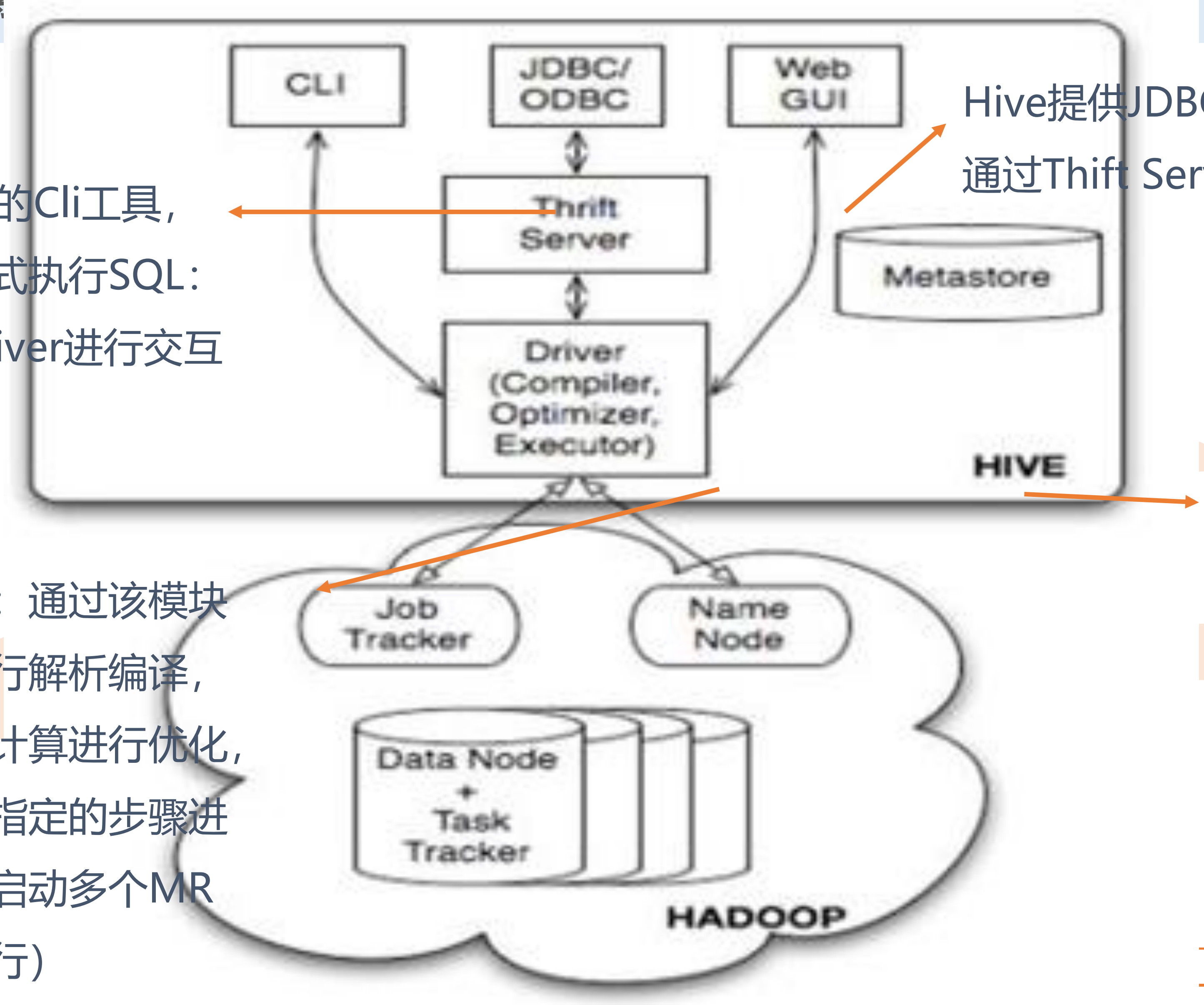
Hive中的SQL与传统SQL区别

	HQL	SQL
数据存储	HDFS、Hbase	Local FS
数据格式	用户自定义	系统决定
数据更新	不支持（把之前的数据覆盖）	支持
索引	有（0.8版之后增加）	有
执行	MapReduce	Executor
执行延迟	高	低
可扩展性	高（UDF、UDAF，UDTF）	低
数据规模	大（数据大于TB）	小
数据检查	读时模式	写时模式

与传统关系数据特点比较

- hive和关系数据库存储文件的系统不同，hive使用的是hadoop的HDFS（hadoop的分布式文件系统），关系数据库则是服务器本地的文件系统；
- hive使用的计算模型是mapreduce，而关系数据库则是自己设计的计算模型；
- 关系数据库都是为实时查询的业务进行设计的，而hive则是为海量数据做数据挖掘设计的，实时性很差
- Hive很容易扩展自己的存储能力和计算能力，这个是继承hadoop的，而关系数据库在这个方面要比数据库差很多。

Hive体系架构



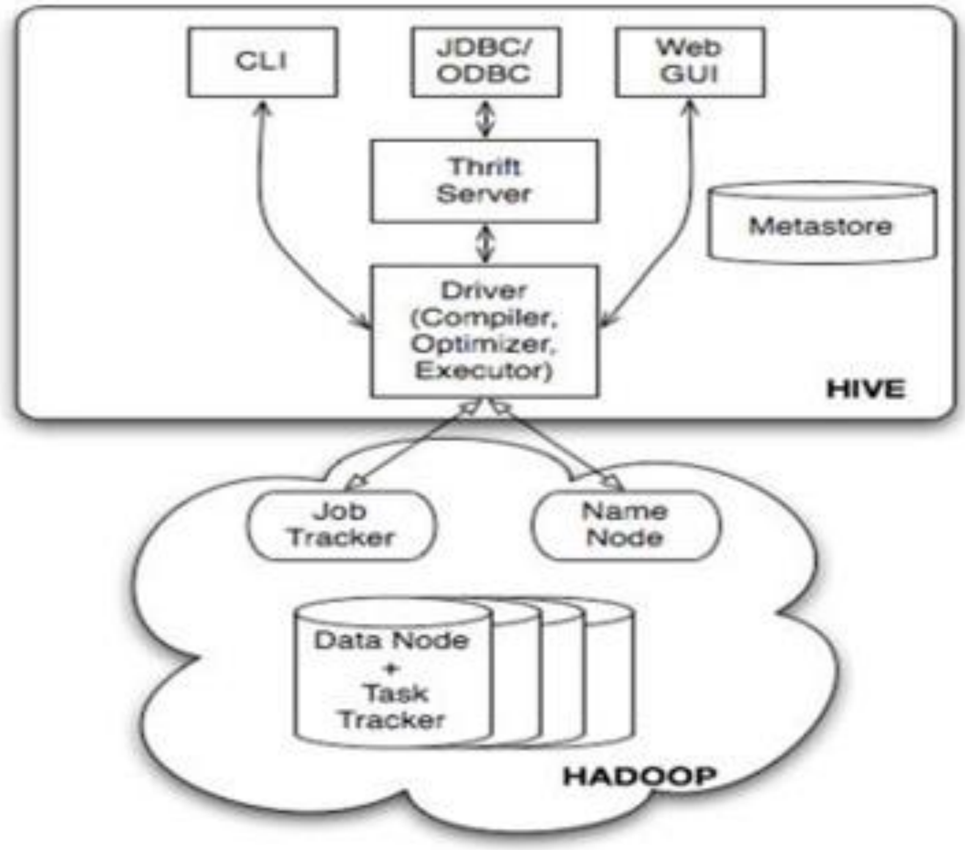
Hive提供的Cli工具，进行交互式执行SQL：直接与Driver进行交互

Hive提供JDBC驱动，作为JAVA的API：JDBC是通过Thift Server来接入，然后发送给Driver

驱动模块：通过该模块对输入进行解析编译，对需求的计算进行优化，然后按照指定的步骤进行（通常启动多个MR任务来执行）

元数据：是一个独立的关系型数据库，通常Mysql，Hive会在其中保存表模式和其他系统元数据

Hive体系架构



用户接口

- CLI：启动的时候，会同时启动一个 Hive 副本
- Client：Hive 的客户端，用户连接至 Hive Server
- GUI：通过浏览器访问 Hive

语句转换

- 解析器：生成抽象语法树
- 语法分析器：验证查询语句
- 逻辑计划生成器（包括优化器）：生成操作符树
- 查询计划生成器：转换为map-reduce任务

数据存储

- Hive数据以文件形式存储在HDFS的指定目录下
- Hive语句生成查询计划，由MapReduce调用执行

Hive数据管理

- hive的表本质就是Hadoop的目录/文件
  - hive默认表存放路径一般都是在你工作目录的hive目录里面，按表名做文件夹分开，如果你有分区表的话，分区值是子文件夹，可以直接在其它的M/R job里直接应用这部分数据

	Name	HDFS Directory
Table	mobile_user	/lbs/mobile_user
Partition	action = insight, day= 20131020	/lbs/mobile_user/action=insight/day=20131020
Bucket	clusted by user into 32 buckets	/lbs/mobile_user/action=insight/day=20131020/part-00000

Hive内部表和外部表

- Hive的create创建表的时候，选择的创建方式：
  - create table
  - create external table

- 特点：
  - 在导入数据到外部表，数据并没有移动到自己的数据仓库目录下，也就是说外部表中的数据并不是由它自己来管理的！而表则不一样；
  - 在删除表的时候，Hive将会把属于表的元数据和数据全部删掉；而删除外部表的时候，Hive仅仅删除外部表的元数据，数据是不会删除的！

Hive中的Partition

- 在 Hive 中，表中的一个 Partition 对应于表下的一个目录，所有的 Partition 的数据都存储在对应的目录中
  - 例如：pvs 表中包含 ds 和 city 两个 Partition，则
  - 对应于 ds = 20090801, ctry = US 的 HDFS 子目录为：/wh/pvs/ds=20090801/ctry=US；
  - 对应于 ds = 20090801, ctry = CA 的 HDFS 子目录为；/wh/pvs/ds=20090801/ctry=CA
- partition是辅助查询，缩小查询范围，加快数据的检索速度和对数据按照一定的规格和条件进行管理。

Hive中的Bucket

- hive中table可以拆分成partition，table和partition可以通过‘CLUSTERED BY’进一步分bucket，bucket中的数据可以通过‘SORT BY’排序。
- create table bucket_user (id int,name string)clustered by (id) into 4 buckets;
- 'set hive.enforce.bucketing = true' 可以自动控制上一轮reduce的数量从而适配bucket的个数，当然，用户也可以自主设置mapred.reduce.tasks去适配bucket个数
- Bucket主要作用:
  - 数据sampling
  - 提升某些查询操作效率，例如mapside join

Hive中的Bucket

- 查看sampling数据：
  - hive> select * from student tablesample(bucket 1 out of 2 on id);
  - tablesample是抽样语句，语法：TABLESAMPLE(BUCKET x OUT OF y)
  - y必须是table总bucket数的倍数或者因子。hive根据y的大小，决定抽样的比例。例如，table总共分了64份，当y=32时，抽取(64/32=)2个bucket的数据，当y=128时，抽取(64/128=)1/2个bucket的数据。x表示从哪个bucket开始抽取。例如，table总bucket数为32，tablesample(bucket 3 out of 16)，表示总共抽取（32/16=）2个bucket的数据，分别为第3个bucket和第（3+16=）19个bucket的数据。

Hive数据类型

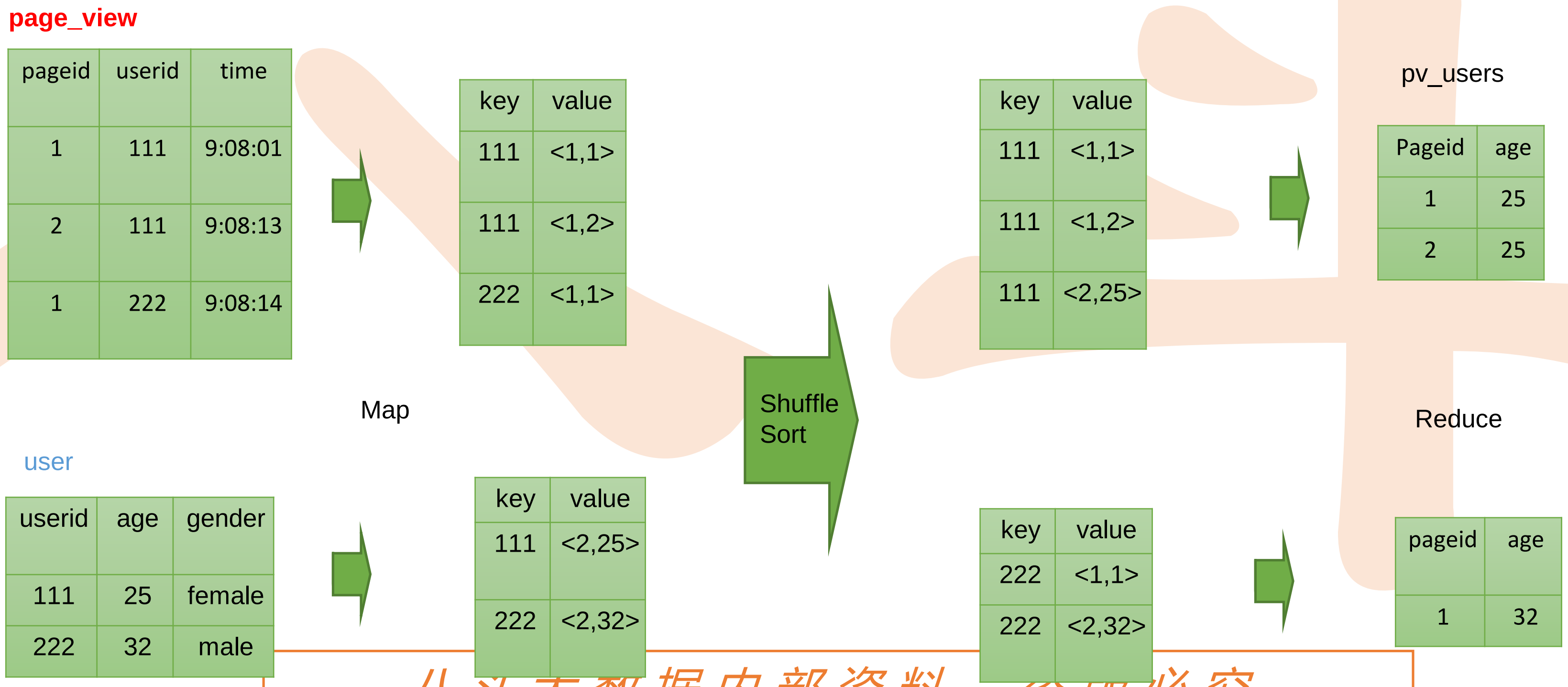
- 数据类型
  - 原生类型
    - TINYINT
    - SMALLINT
    - INT
    - BIGINT
    - BOOLEAN
    - FLOAT
    - DOUBLE
    - STRING
    - BINARY（Hive 0.8.0以上才可用）
    - TIMESTAMP（Hive 0.8.0以上才可用）

Hive数据类型

- 数据类型
  - 复合类型
    - Arrays：ARRAY<data_type>
    - Maps:MAP<primitive_type, data_type>
    - Structs:STRUCT<col_name: data_type[COMMENT col_comment],……>
    - Union:UNIONTYPE<data_type, data_type,……>

Hive SQL——Join in MR

```
INSERT OVERWRITE TABLE pv_users
SELECT pv.pageid, u.age
FROM page_view pv
   JOIN user u
   ON (pv.userid = u.userid);
```



page_view

pageid	userid	time
1	111	9:08:01
2	111	9:08:13
1	222	9:08:14

user

userid	age	gender
111	25	female
222	32	male

key	value
111	<1,1>
111	<1,2>
222	<1,1>

key	value
111	<2,25>
222	<2,32>

key	value
111	<1,1>
111	<1,2>
111	<2,25>

key	value
222	<1,1>
222	<2,32>

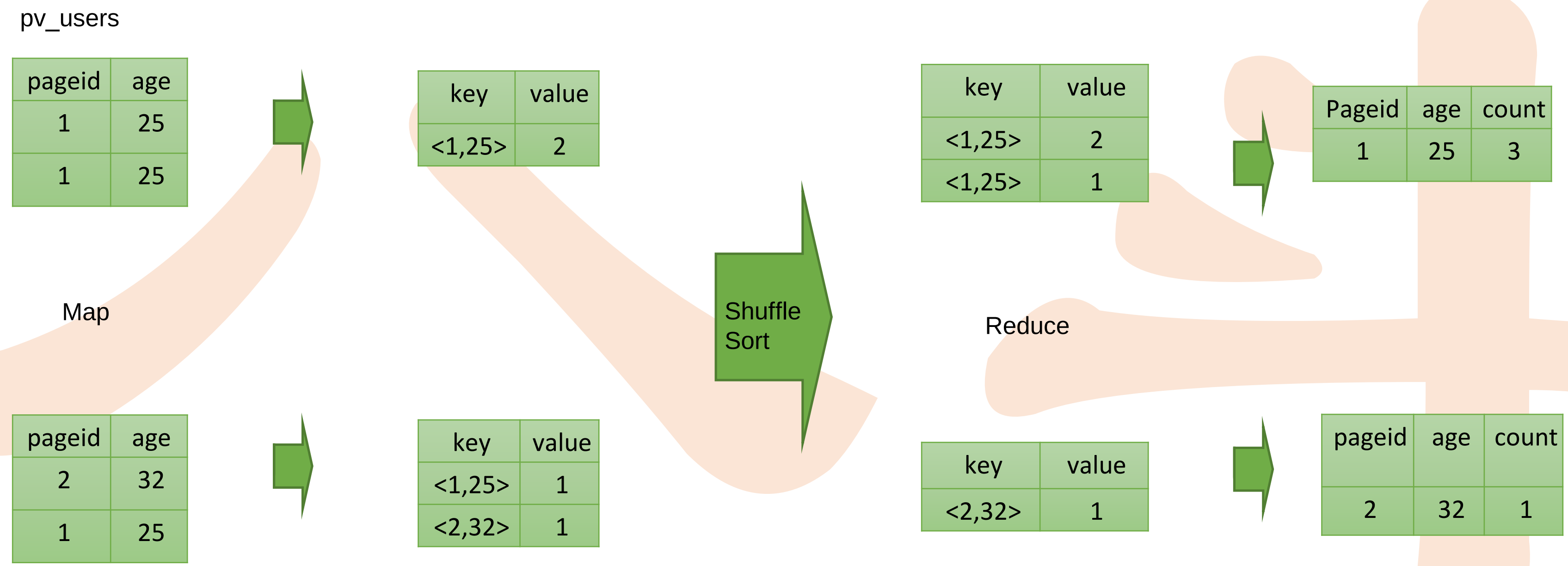
pv_users

Pageid	age
1	25
2	25

pageid	age
1	32

Hive SQL——Join in MR

SELECT pageid, age, count(1)
FROM pv_users
GROUP BY pageid, age;



pageid	age
1	25
1	25

key	value
<1,25>	2

pageid	age
2	32
1	25

key	value
<1,25>	1
<2,32>	1

key	value
<1,25>	2
<1,25>	1

Pageid	age	count
1	25	3

key	value
<2,32>	1

pageid	age	count
2	32	1

Hive的优化

- Map的优化：
  - 作业会通过input的目录产生一个或者多个map任务。set dfs.block.size(=128)
  - Map越多越好吗？是不是保证每个map处理接近文件块的大小？
  - 如何合并小文件，减少map数？

```
set mapred.max.split.size=100000000;
set mapred.min.split.size.per.node=100000000;
set mapred.min.split.size.per.rack=100000000;
set hive.input.format=org.apache.hadoop.hive.ql.io.CombineHiveInputFormat;
```

  - 如何适当的增加map数？

```
set mapred.map.tasks=10;
```

  - Map端聚合 hive.map.aggr=true。 Mr中的Combiners.

Hive的优化

- Reduce的优化：
  - hive.exec.reducers.bytes.per.reducer；reduce任务处理的数据量
  - 调整reduce的个数：
    - 设置reduce处理的数据量
    - set mapred.reduce.tasks=10

```
select pt,count(1)
 from popt_tbaccountcopy_mes
where pt = '2012-07-04' group by pt;
写成
select count(1)
       from popt_tbaccountcopy_mes
        where pt = '2012-07-04';
```

- 一个Reduce：
  - 没有group by
  - order by（可以使用distribute by和sort by）
  - 笛卡尔积

Hive的优化

- 分区裁剪（partition）
  - Where中的分区条件，会提前生效，不必特意做子查询，直接Join和GroupBy
- 笛卡尔积
  - join的时候不加on条件或者无效的on条件，Hive只能使用1个reducer来完成笛卡尔积
- Map join
  - /*+ MAPJOIN(tablelist) */，必须是小表，不要超过1G，或者50万条记录
- Union all
  - 先做union all再做join或group by等操作可以有效减少MR过程，尽管是多个Select，最终只有一个mr

Hive的优化

- Multi-insert & multi-group by
  - 从一份基础表中按照不同的维度，一次组合出不同的数据
  - FROM from_statement
  - INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1=val1)] select_statement1 group by key1
  - INSERT OVERWRITE TABLE tablename2 [PARTITION(partcol2=val2)] select_statement2 group by key2
- Automatic merge
  - 当文件大小比阈值小时，hive会启动一个mr进行合并
  - hive.merge.mapfiles = true 是否和并 Map 输出文件，默认为 True
  - hive.merge.mapredfiles = false 是否合并 Reduce 输出文件，默认为 False
  - hive.merge.size.per.task = 256*1000*1000 合并文件的大小
- Multi-Count Distinct
  - 必须设置参数：set hive.groupby.skewindata=true;
  - select dt, count(distinct uniq_id), count(distinct ip)
  - from ods_log where dt=20170301 group by dt

Hive的Join优化

- 一个MR job

```
SELECT a.val, b.val, c.val
FROM a
JOIN b ON (a.key = b.key1)
JOIN c ON (a.key = c.key1)
```

- 生成多个MR job

```
SELECT a.val, b.val, c.val
FROM a
JOIN b ON (a.key = b.key1)
JOIN c ON (c.key = b.key1)
```

Hive的Join优化——表连接顺序

- 按照JOIN顺序中的最后一个表应该尽量是大表，因为JOIN前一阶段生成的数据会存在于Reducer的buffer中，通过stream最后面的表，直接从Reducer的buffer中读取已经缓冲的中间结果数据（这个中间结果数据可能是JOIN顺序中，前面表连接的结果的Key，数据量相对较小，内存开销就小），这样，与后面的大表进行连接时，只需要从buffer中读取缓存的Key，与大表中的指定Key进行连接，速度会更快，也可能避免内存缓冲区溢出。

```
SELECT /*+ STREAMTABLE(a) */ a.val, b.val, c.val
FROM a
JOIN b ON (a.key = b.key1)
JOIN c ON (c.key = b.key1)；
```

a表被视为大表

```
SELECT /*+ MAPJOIN(b) */ a.key, a.value
FROM a
JOIN b ON a.key = b.key;
```

MAPJION会把小表全部读入内存中，在map阶段直接拿另外一个表的数据和内存中表数据做匹配，由于在map是进行了join操作，省去了reduce运行的效率也会高很多.

Hive的Join优化——表连接顺序

- 左连接时，左表中出现的JOIN字段都保留，右表没有连接上的都为空。

```
SELECT a.val, b.val
FROM a
LEFT OUTER JOIN b ON (a.key=b.key)
WHERE a.ds='2009-07-07' AND b.ds='2009-07-07'
```

```
ASELECT a.val, b.val
FROM a
LEFT OUTER JOIN b
ON (a.key=b.key AND b.ds='2009-07-07' AND a.ds='2009-07-07')
```

- 执行顺序是，首先完成2表JOIN，然后再通过WHERE条件进行过滤，这样在JOIN过程中可能会输出大量结果，再对这些结果进行过滤，比较耗时。可以进行优化，将WHERE条件放在ON后，在JOIN的过程中，就对不满足条件的记录进行了预先过滤。

Hive的优化——并行执行

- 并行实行：
  - 同步执行hive的多个阶段，hive在执行过程，将一个查询转化成一个或者多个阶段。某个特定的job可能包含众多的阶段，而这些阶段可能并非完全相互依赖的，也就是说可以并行执行的，这样可能使得整个job的执行时间缩短。hive执行开启：set hive.exec.parallel=true

Hive的优化——数据倾斜

- **操作**
  - Join
  - Group by
  - Count Distinct
- **原因**
  - key分布不均导致的
  - 人为的建表疏忽
  - 业务数据特点
- **症状**
  - 任务进度长时间维持在99%（或100%），查看任务监控页面，发现只有少量（1个或几个）reduce子任务未完成。
  - 查看未完成的子任务，可以看到本地读写数据量积累非常大，通常超过10GB可以认定为发生数据倾斜。
- **倾斜度**
  - 平均记录数超过50w且最大记录数是超过平均记录数的4倍。
  - 最长时长比平均时长超过4分钟，且最大时长超过平均时长的2倍。
- **万能方法**
  - hive.groupby.skewindata=true

Hive的优化——数据倾斜——大小表关联

- **原因**
  - Hive在进行join时，按照join的key进行分发，而在join左边的表的数据会首先读入内存，如果左边表的key相对分散，读入内存的数据会比较小，join任务执行会比较快；而如果左边的表key比较集中，而这张表的数据量很大，那么数据倾斜就会比较严重，而如果这张表是小表，则还是应该把这张表放在join左边。
- **思路**
  - 将key相对分散，并且数据量小的表放在join的左边，这样可以有效减少内存溢出错误发生的几率
  - 使用map join让小的维度表先进内存。
- **方法**
  - Small_table join big_table

Hive的优化——数据倾斜——大大表关联

- **原因**
  - 日志中有一部分的userid是空或者是0的情况，导致在用user_id进行hash分桶的时候，会将日志中userid为0或者空的数据分到一起，导致了过大的斜率。

- **思路**
  - 把空值的key变成一个字符串加上随机数，把倾斜的数据分到不同的reduce上，由于null值关联不上，处理后并不影响最终结果。

- **方法**
  - on case when (x.uid = '-' or x.uid = '0 ‘ or x.uid is null) then concat('dp_hive_search',rand()) else x.uid end = f.user_id;

Hive的优化——数据倾斜——大大表关联（业务削减）

- **案例**
  - Select * from dw_log t join dw_user t1 on t.user_id=t1.user_id
  - 现象：两个表都上千万，跑起来很悬
- **思路**
  - 当天登陆的用户其实很少
- **方法**
  - Select/*+MAPJOIN(t12)*/ *
  - from dw_log t11
  - join (
  -    select/*+MAPJOIN(t)*/ t1.*
  -    from (
  -        select user_id from dw_log group by user_id
  -    ) t
  -    join dw_user t1
  -    on t.user_id=t1.user_id
  -) t12
  - on t11.user_id=t12.user_id

Hive的优化——数据倾斜——聚合时存在大量特殊值

- **原因**
  - 做count distinct时，该字段存在大量值为NULL或空的记录。

- **思路**
  - count distinct时，将值为空的情况单独处理，如果是计算count distinct，可以不用处理，直接过滤，在最后结果中加1。
  - 如果还有其他计算，需要进行group by，可以先将值为空的记录单独处理，再和其他计算结果进行union

- **方法**

```
select cast(count(distinct(user_id))+1 as bigint) as user_cnt
        from tab_a
        where user_id is not null and user_id <> ''
```

Hive的优化——数据倾斜——空间换时间

- **案例**
  - Select day,count(distinct session_id),count(distinct user_id) from log a group by day
- **问题**
  - 同一个reduce上进行distinct操作时压力很大
- **方法**

```
select day,
       count(case when type='session' then 1 else null end) as session_cnt,
       count(case when type='user' then 1 else null end) as user_cnt
from (
     select day,session_id,type
     from (
          select day,session_id,'session' as type
          from log
          union all
          select day user_id,'user' as type
          from log
     )
     group by day,session_id,type
) t1
group by day
```

OutLine

Hive基础

【实践】Hive搭建

【实践】Hive练习

Mysql配置

- 默认情况下，Hive的元数据信息存储在内置的Derby数据中。
- Hive支持将元数据存储在MySQL中

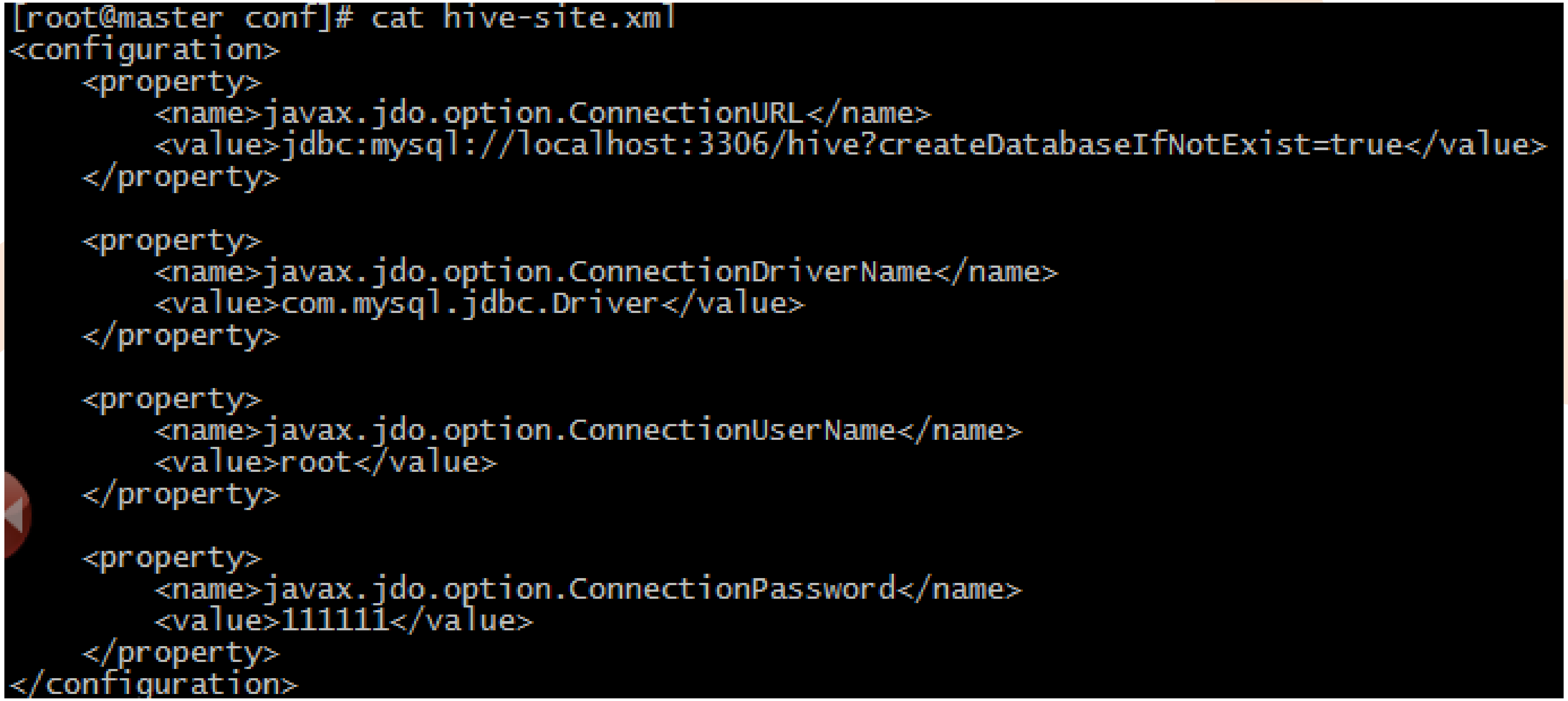
- 元数据存储配置：
  - 【本地配置1】：默认
  - 【本地配置2】：本地搭建mysql，通过localhost:Port方式访问
  - 【远程配置】：远程搭建mysql，通过IP:Port方式访问

Mysql配置

- **第一步：**安装MySQL服务器端和MySQL客户端,并启动MySQL服务
- 安装：
  - yum install mysql
  - yum install mysql-server
- 启动：
  - /etc/init.d/mysqld start
- 设置用户名和密mysqladmin -u root password码：
  - '111111 ’
- 测试登录是否成功：
  - mysql -uroot -p111111

Mysql配置

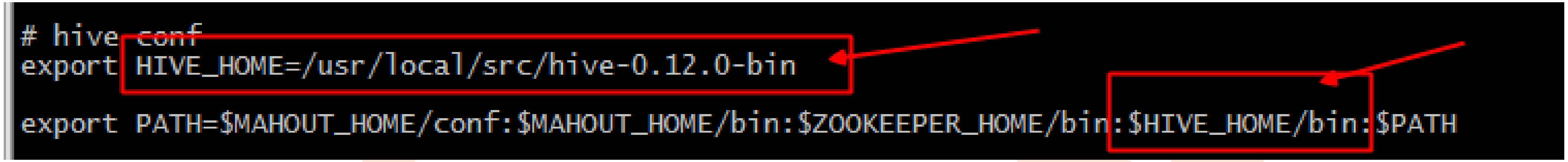
- **第二步：**安装Hive
- 下载apache-hive-0.13.0-bin.tgz，并解压：
- 在conf目录下，创建hive-site.xml配置文件：



```
[root@master conf]# cat hive-site.xml
<configuration>
    <property>
        <name>javax.jdo.option.ConnectionURL</name>
        <value>jdbc:mysql://localhost:3306/hive?createDatabaseIfNotExist=true</value>
    </property>

    <property>
        <name>javax.jdo.option.ConnectionDriverName</name>
        <value>com.mysql.jdbc.Driver</value>
    </property>

    <property>
        <name>javax.jdo.option.ConnectionUserName</name>
        <value>root</value>
    </property>

    <property>
        <name>javax.jdo.option.ConnectionPassword</name>
        <value>111111</value>
    </property>
</configuration>
```

Mysql配置

- **第二步：** 安装Hive
- 修改bashrc，配置环境变量：

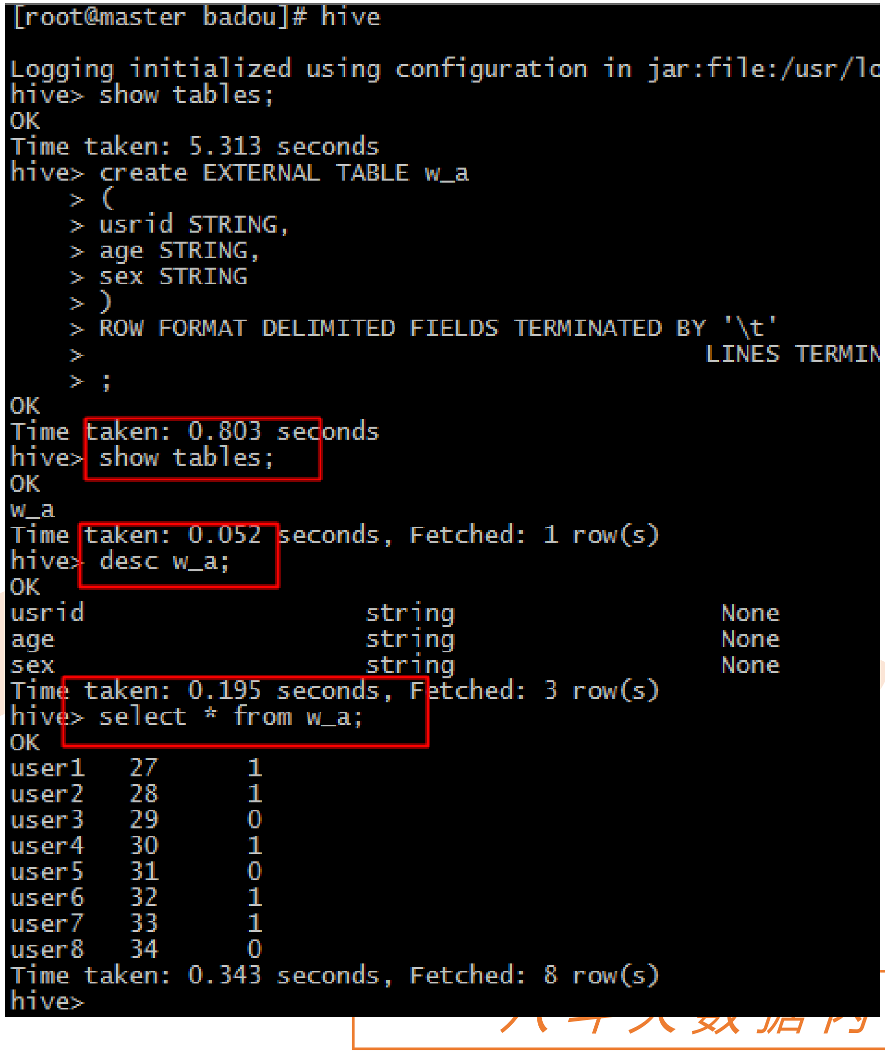


```
# hive conf
export HIVE_HOME=/usr/local/src/hive-0.12.0-bin

export PATH=$MAHOUT_HOME/conf:$MAHOUT_HOME/bin:$ZOOKEEPER_HOME/bin:$HIVE_HOME/bin:$PATH
```

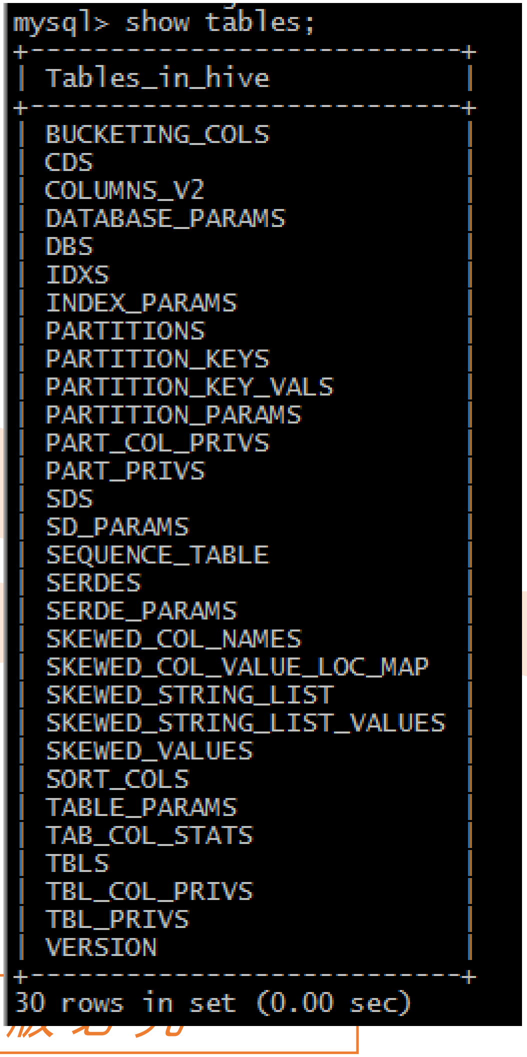
- 将mysql-connector-java-5.1.41-bin.jar拷贝到hive home的lib目录下，以支持hive对mysql的操作

Mysql配置

- **第三步：**测试Hive



```
[root@master badou]# hive

Logging initialized using configuration in jar:file:/usr/lo
hive> show tables;
OK
Time taken: 5.313 seconds
hive> create EXTERNAL TABLE w_a
    > (
    > usrid STRING,
    > age STRING,
    > sex STRING
    > )
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
    >                                         LINES TERMIN
    > ;
OK
Time taken: 0.803 seconds
hive> show tables;
OK
w_a
Time taken: 0.052 seconds, Fetched: 1 row(s)
hive> desc w_a;
OK
usrid                   string                  None
age                     string                  None
sex                     string                  None
Time taken: 0.195 seconds, Fetched: 3 row(s)
hive> select * from w_a;
OK
user1   27      1
user2   28      1
user3   29      0
user4   30      1
user5   31      0
user6   32      1
user7   33      1
user8   34      0
Time taken: 0.343 seconds, Fetched: 8 row(s)
hive>
```



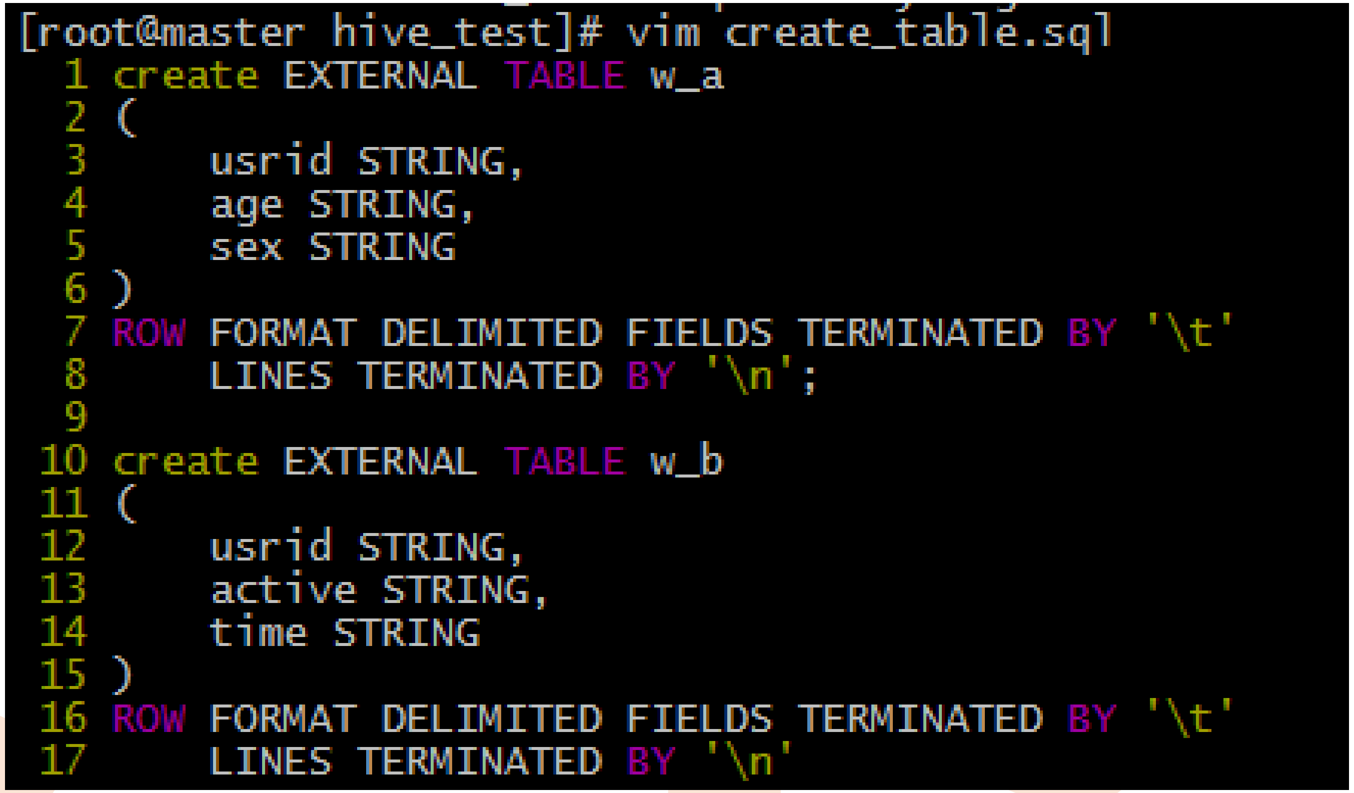
```
mysql> show tables;
+---------------------------+
| Tables_in_hive            |
+---------------------------+
| BUCKETING_COLS            |
| CDS                       |
| COLUMNS_V2                |
| DATABASE_PARAMS           |
| DBS                       |
| IDXS                      |
| INDEX_PARAMS              |
| PARTITIONS                |
| PARTITION_KEYS            |
| PARTITION_KEY_VALS        |
| PARTITION_PARAMS          |
| PART_COL_PRIVS            |
| PART_PRIVS                |
| SDS                       |
| SD_PARAMS                 |
| SEQUENCE_TABLE            |
| SERDES                    |
| SERDE_PARAMS              |
| SKEWED_COL_NAMES          |
| SKEWED_COL_VALUE_LOC_MAP  |
| SKEWED_STRING_LIST        |
| SKEWED_STRING_LIST_VALUES |
| SKEWED_VALUES             |
| SORT_COLS                 |
| TABLE_PARAMS              |
| TAB_COL_STATS             |
| TBLS                      |
| TBL_COL_PRIVS             |
| TBL_PRIVS                 |
| VERSION                   |
+---------------------------+
30 rows in set (0.00 sec)
```

OutLine

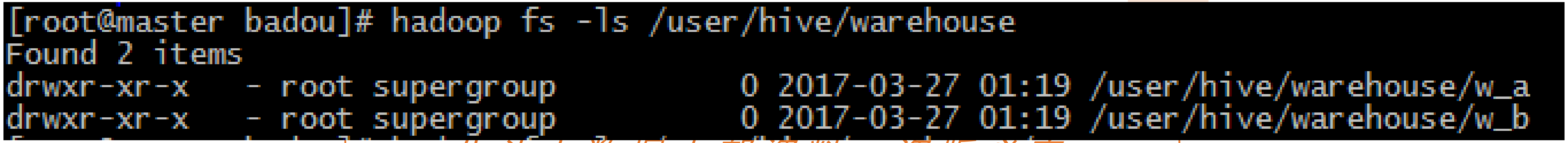
- Hive基础
- 【实践】Hive搭建
- 【实践】Hive练习

案例一：导入本地Local的数据，并进行简单统计

- 准备数据
- 设计schema，建库，建表



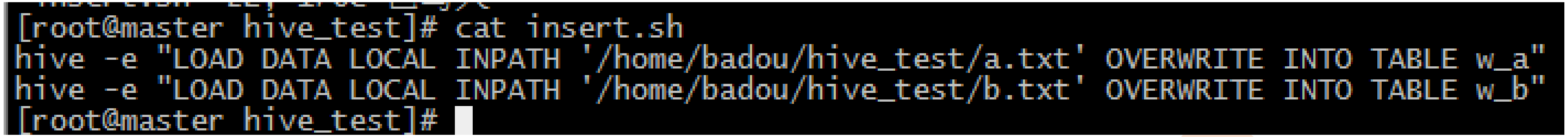
```
[root@master hive_test]# vim create_table.sql
create EXTERNAL TABLE w_a
(
    usrid STRING,
    age STRING,
    sex STRING
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
    LINES TERMINATED BY '\n';

create EXTERNAL TABLE w_b
(
    usrid STRING,
    active STRING,
    time STRING
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
    LINES TERMINATED BY '\n'
```



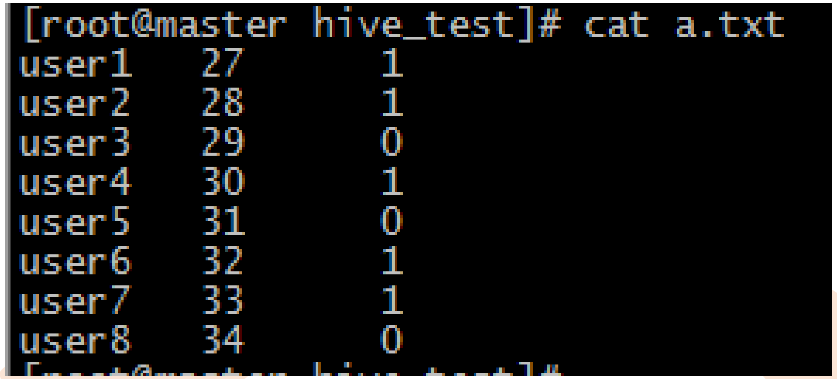
```
[root@master badou]# hadoop fs -ls /user/hive/warehouse
Found 2 items
drwxr-xr-x   - root supergroup          0 2017-03-27 01:19 /user/hive/warehouse/w_a
drwxr-xr-x   - root supergroup          0 2017-03-27 01:19 /user/hive/warehouse/w_b
```

案例一：导入本地Local的数据，并进行简单统计

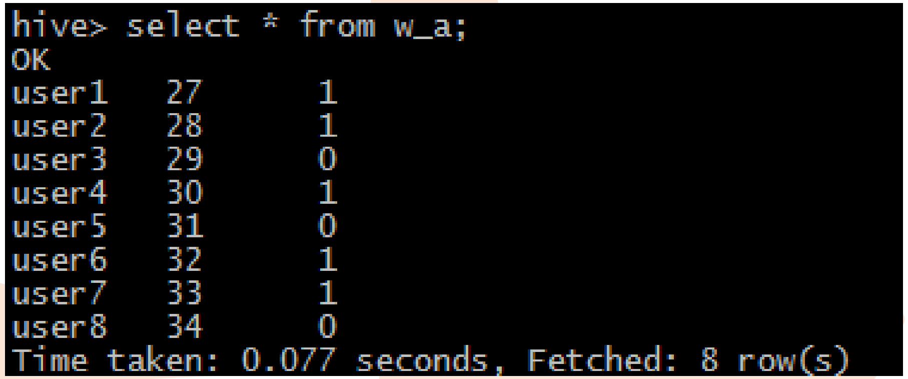
- 导入



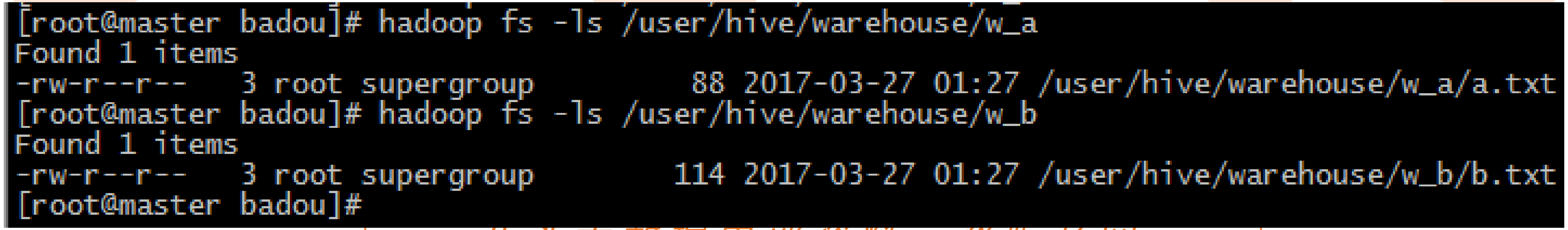
```
[root@master hive_test]# cat insert.sh
hive -e "LOAD DATA LOCAL INPATH '/home/badou/hive_test/a.txt' OVERWRITE INTO TABLE w_a"
hive -e "LOAD DATA LOCAL INPATH '/home/badou/hive_test/b.txt' OVERWRITE INTO TABLE w_b"
[root@master hive_test]#
```



```
[root@master hive_test]# cat a.txt
user1   27      1
user2   28      1
user3   29      0
user4   30      1
user5   31      0
user6   32      1
user7   33      1
user8   34      0
```



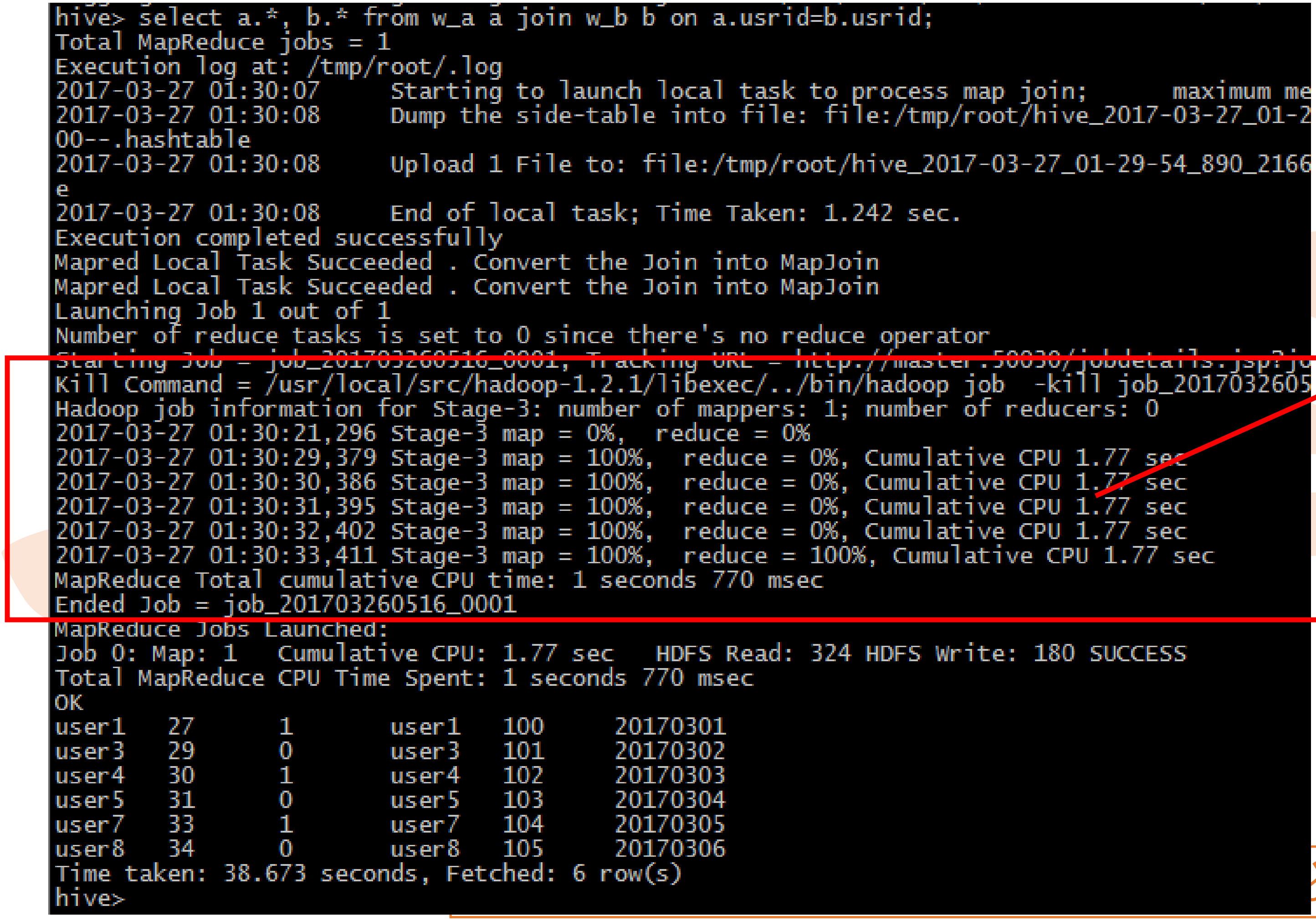
```
hive> select * from w_a;
OK
user1   27      1
user2   28      1
user3   29      0
user4   30      1
user5   31      0
user6   32      1
user7   33      1
user8   34      0
Time taken: 0.077 seconds, Fetched: 8 row(s)
```



```
[root@master badou]# hadoop fs -ls /user/hive/warehouse/w_a
Found 1 items
-rw-r--r--   3 root supergroup         88 2017-03-27 01:27 /user/hive/warehouse/w_a/a.txt
[root@master badou]# hadoop fs -ls /user/hive/warehouse/w_b
Found 1 items
-rw-r--r--   3 root supergroup        114 2017-03-27 01:27 /user/hive/warehouse/w_b/b.txt
[root@master badou]#
```

案例二：两表的Join

- 执行命令：select a.*, b.* from w_a a join w_b b on a.usrid=b.usrid;



```
hive> select a.*, b.* from w_a a join w_b b on a.usrid=b.usrid;
Total MapReduce jobs = 1
Execution log at: /tmp/root/.log
2017-03-27 01:30:07     Starting to launch local task to process map join;      maximum me
2017-03-27 01:30:08     Dump the side-table into file: file:/tmp/root/hive_2017-03-27_01-2
00--.hashtable
2017-03-27 01:30:08     Upload 1 File to: file:/tmp/root/hive_2017-03-27_01-29-54_890_2166
e
2017-03-27 01:30:08     End of local task; Time Taken: 1.242 sec.
Execution completed successfully
Mapred Local Task Succeeded . Convert the Join into MapJoin
Mapred Local Task Succeeded . Convert the Join into MapJoin
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201703260516_0001, Tracking URL = http://master:50030/jobdetails.jsp?jo
Kill Command = /usr/local/src/hadoop-1.2.1/libexec/../bin/hadoop job  -kill job_2017032605
Hadoop job information for Stage-3: number of mappers: 1; number of reducers: 0
2017-03-27 01:30:21,296 Stage-3 map = 0%,  reduce = 0%
2017-03-27 01:30:29,379 Stage-3 map = 100%,  reduce = 0%, Cumulative CPU 1.77 sec
2017-03-27 01:30:30,386 Stage-3 map = 100%,  reduce = 0%, Cumulative CPU 1.77 sec
2017-03-27 01:30:31,395 Stage-3 map = 100%,  reduce = 0%, Cumulative CPU 1.77 sec
2017-03-27 01:30:32,402 Stage-3 map = 100%,  reduce = 0%, Cumulative CPU 1.77 sec
2017-03-27 01:30:33,411 Stage-3 map = 100%,  reduce = 100%, Cumulative CPU 1.77 sec
MapReduce Total cumulative CPU time: 1 seconds 770 msec
Ended Job = job_201703260516_0001
MapReduce Jobs Launched:
Job 0: Map: 1   Cumulative CPU: 1.77 sec   HDFS Read: 324 HDFS Write: 180 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 770 msec
OK
user1   27      1       user1   100     20170301
user3   29      0       user3   101     20170302
user4   30      1       user4   102     20170303
user5   31      0       user5   103     20170304
user7   33      1       user7   104     20170305
user8   34      0       user8   105     20170306
Time taken: 38.673 seconds, Fetched: 6 row(s)
hive>
```

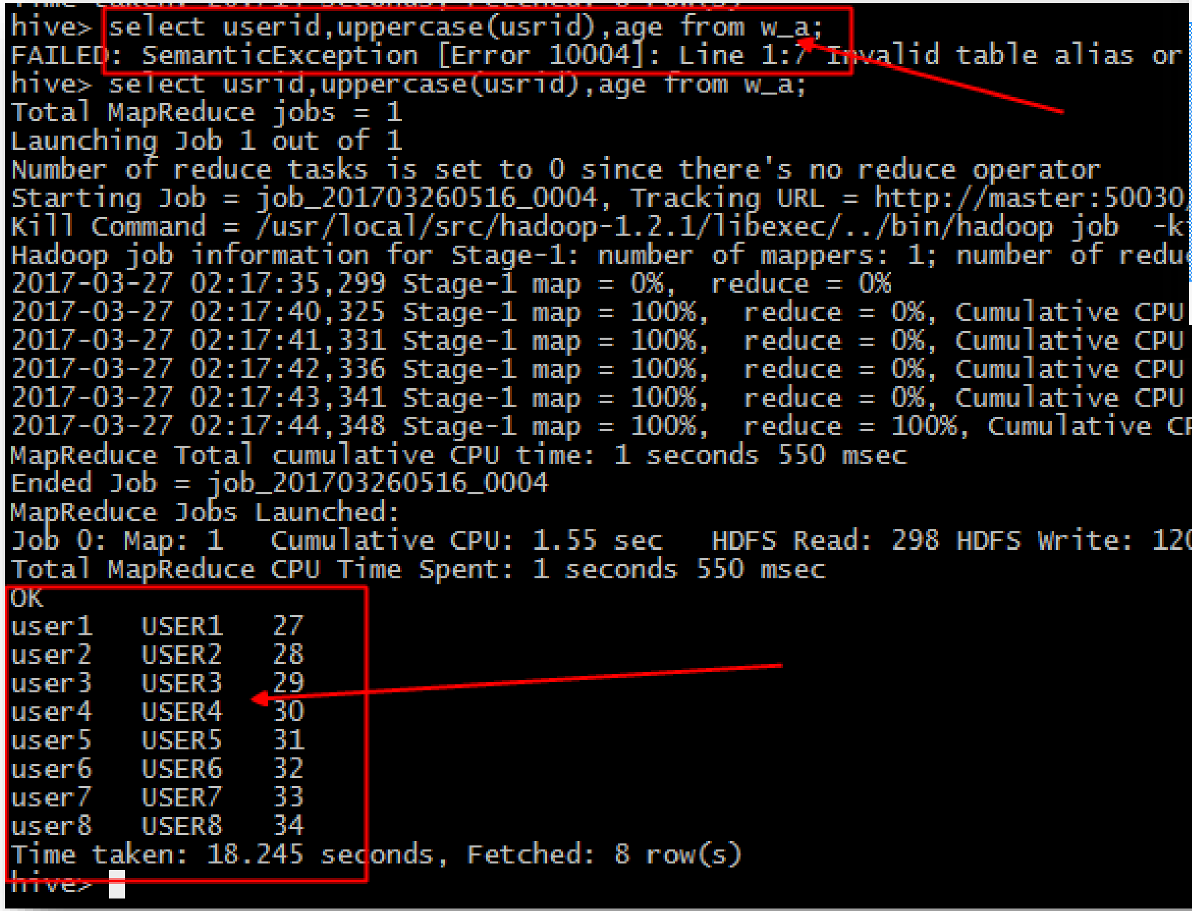
MapReduce！！！

案例三：UDF

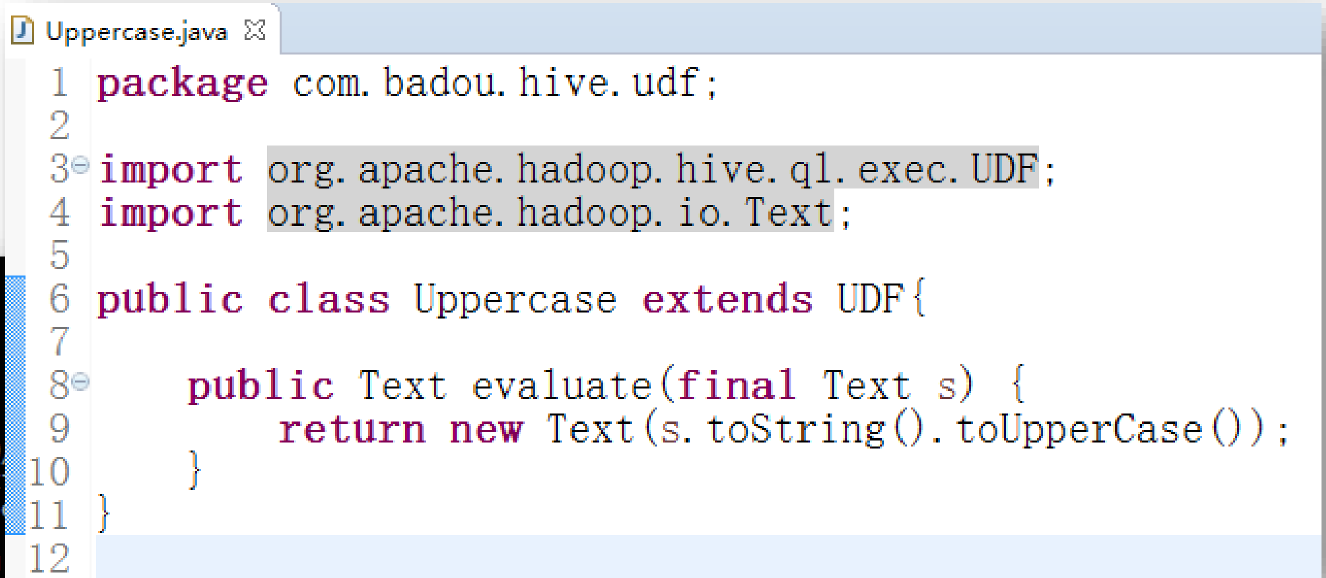
- UDF：User-Defined-Function 用户自定义函数
- UDF函数可以直接应用于select语句，对查询结构做格式化处理后，再输出内容。
- 编写UDF函数的时候需要注意一下几点：
  - 自定义UDF需要继承org.apache.hadoop.hive.ql.UDF。
  - 需要实现evaluate函。
  - evaluate函数支持重载。

案例三：UDF

- 借助UDF，实现一个大写函数
- 效果验证：



```
hive> select userid,uppercase(usrid),age from w_a;
FAILED: SemanticException [Error 10004]: Line 1:7 Invalid table alias or
hive> select usrid,uppercase(usrid),age from w_a;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201703260516_0004, Tracking URL = http://master:50030
Kill Command = /usr/local/src/hadoop-1.2.1/libexec/../bin/hadoop job  -k
Hadoop job information for Stage-1: number of mappers: 1; number of redu
2017-03-27 02:17:35,299 Stage-1 map = 0%,  reduce = 0%
2017-03-27 02:17:40,325 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU
2017-03-27 02:17:41,331 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.55 sec
2017-03-27 02:17:42,336 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.55 sec
2017-03-27 02:17:43,341 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.55 sec
2017-03-27 02:17:44,348 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 1.55 sec
MapReduce Total cumulative CPU time: 1 seconds 550 msec
Ended Job = job_201703260516_0004
MapReduce Jobs Launched:
Job 0: Map: 1   Cumulative CPU: 1.55 sec   HDFS Read: 298 HDFS Write: 120 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 550 msec
OK
user1   USER1   27
user2   USER2   28
user3   USER3   29
user4   USER4   30
user5   USER5   31
user6   USER6   32
user7   USER7   33
user8   USER8   34
Time taken: 18.245 seconds, Fetched: 8 row(s)
hive>
```



```java
package com.badou.hive.udf;

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

public class Uppercase extends UDF{

    public Text evaluate(final Text s) {
        return new Text(s.toString().toUpperCase());
    }
}
```

案例四：从HDFS中导入

- 执行命令：
  - LOAD DATA INPATH '/user_name.data.utf.txt' OVERWRITE INTO TABLE u_info
- overwrite表示加载的数据会覆盖原来的内容
- 对比本地的方式：LOAD DATA LOCAL INPATH

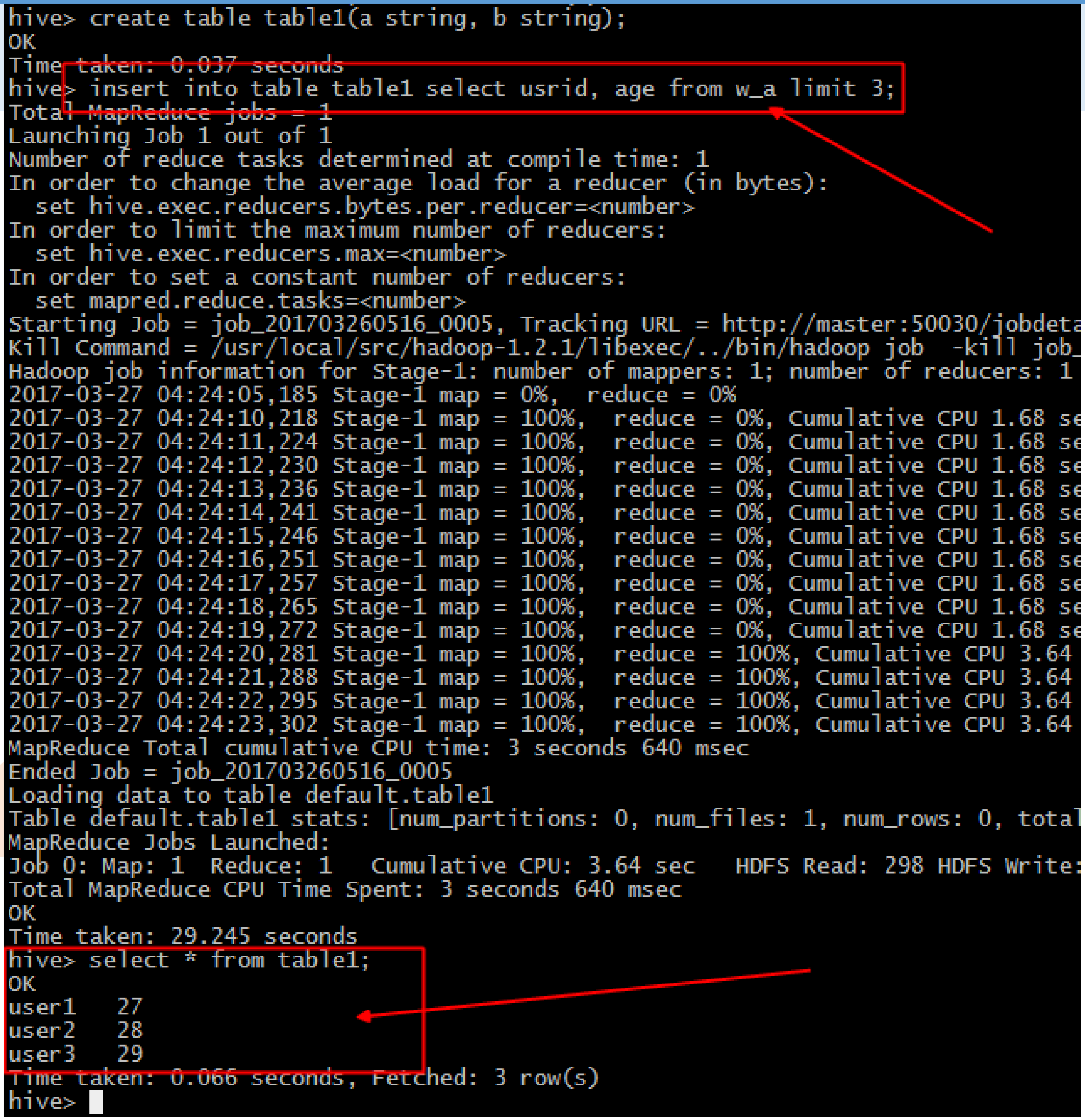
案例五：利用Insert命令导入数据

- 执行命令（例子）：

insert into table table1 select usrid, age from w_a limit 3;

- 也可以支持动态分区插入：

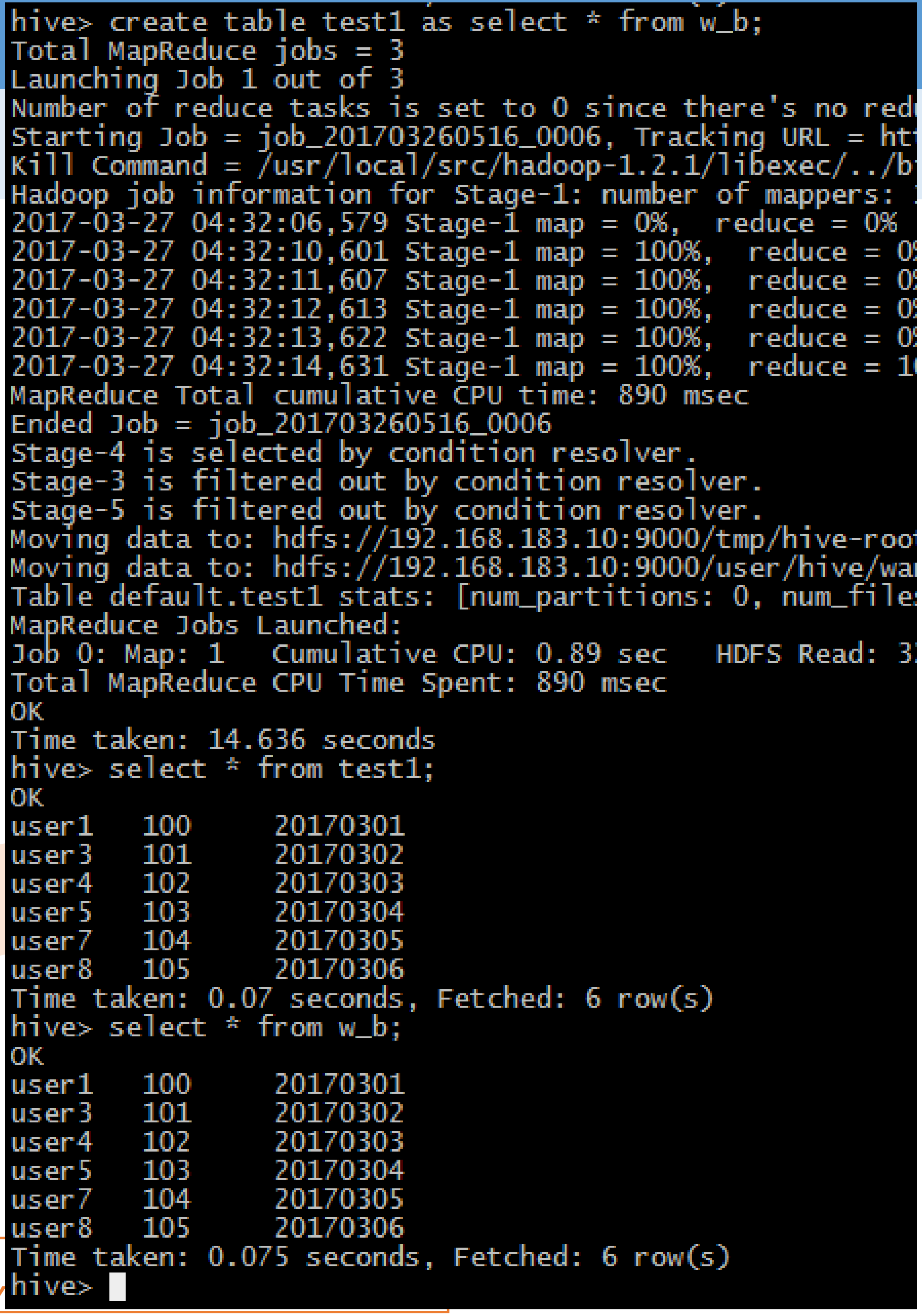
insert into table test1 partition(c) select * from test2;



```
hive> create table table1(a string, b string);
OK
Time taken: 0.037 seconds
hive> insert into table table1 select usrid, age from w_a limit 3;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201703260516_0005, Tracking URL = http://master:50030/jobdeta
Kill Command = /usr/local/src/hadoop-1.2.1/libexec/../bin/hadoop job  -kill job_
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2017-03-27 04:24:05,185 Stage-1 map = 0%,  reduce = 0%
2017-03-27 04:24:10,218 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.68 se
2017-03-27 04:24:11,224 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.68 se
2017-03-27 04:24:12,230 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.68 se
2017-03-27 04:24:13,236 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.68 se
2017-03-27 04:24:14,241 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.68 se
2017-03-27 04:24:15,246 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.68 se
2017-03-27 04:24:16,251 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.68 se
2017-03-27 04:24:17,257 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.68 se
2017-03-27 04:24:18,265 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.68 se
2017-03-27 04:24:19,272 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.68 se
2017-03-27 04:24:20,281 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.64
2017-03-27 04:24:21,288 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.64
2017-03-27 04:24:22,295 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.64
2017-03-27 04:24:23,302 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.64
MapReduce Total cumulative CPU time: 3 seconds 640 msec
Ended Job = job_201703260516_0005
Loading data to table default.table1
Table default.table1 stats: [num_partitions: 0, num_files: 1, num_rows: 0, total
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1   Cumulative CPU: 3.64 sec   HDFS Read: 298 HDFS Write:
Total MapReduce CPU Time Spent: 3 seconds 640 msec
OK
Time taken: 29.245 seconds
hive> select * from table1;
OK
user1   27
user2   28
user3   29
Time taken: 0.066 seconds, Fetched: 3 row(s)
hive> 
```

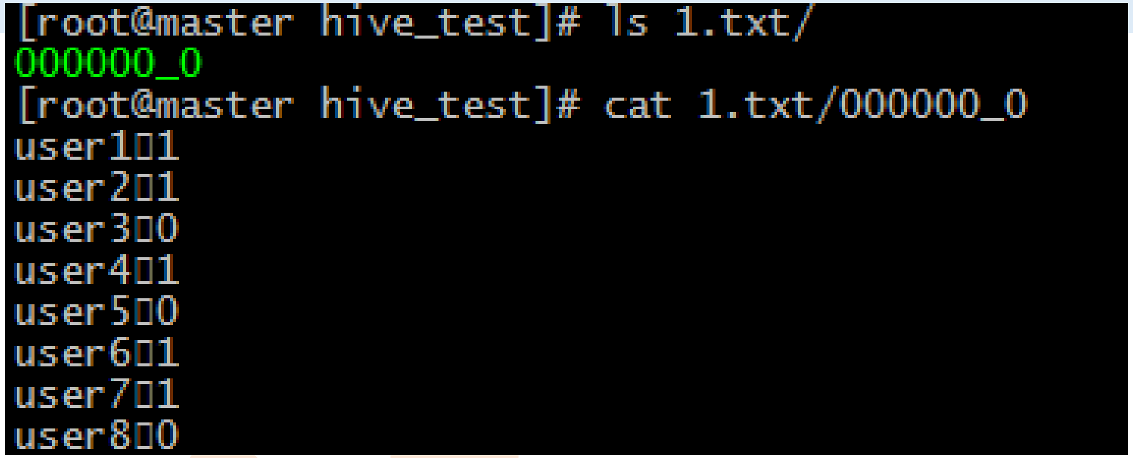
案例六：直接通过查询插入

- 执行命令（例子）：
- create table test2 as select * from test1;



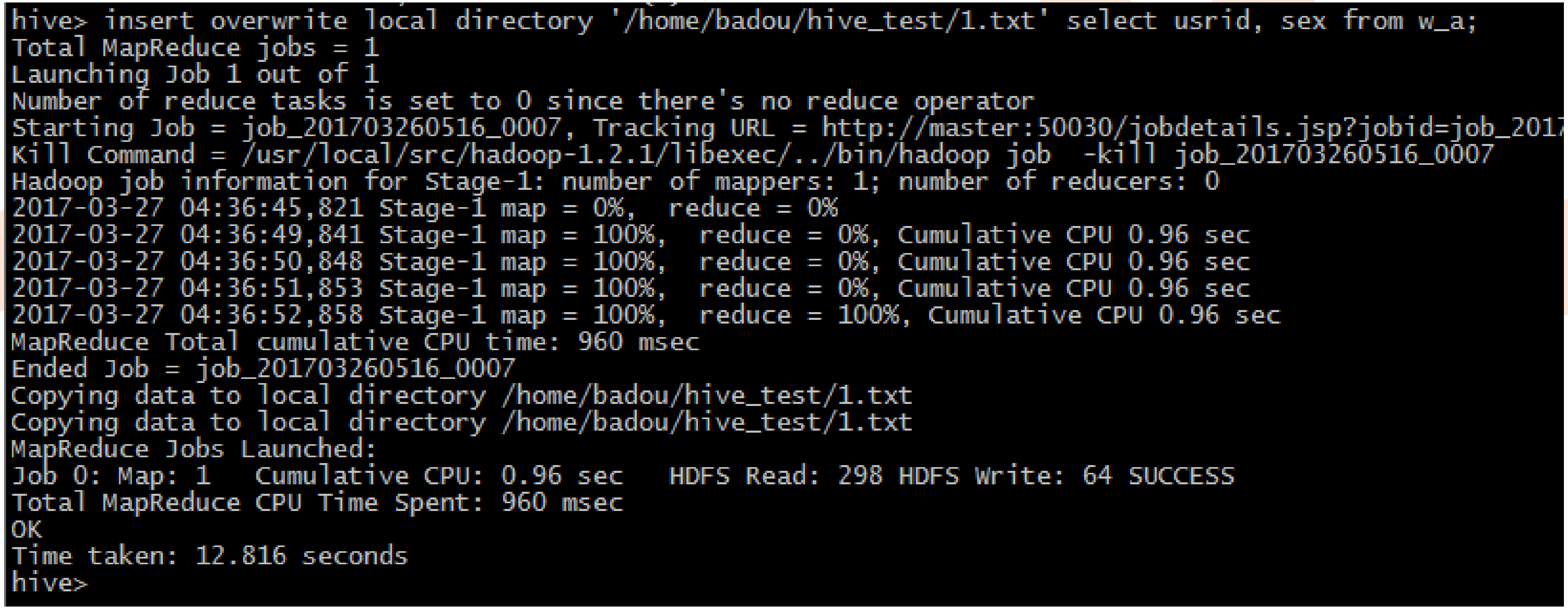
```
hive> create table test1 as select * from w_b;
Total MapReduce jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no red
Starting Job = job_201703260516_0006, Tracking URL = ht
Kill Command = /usr/local/src/hadoop-1.2.1/libexec/../b
Hadoop job information for Stage-1: number of mappers: 
2017-03-27 04:32:06,579 Stage-1 map = 0%,  reduce = 0%
2017-03-27 04:32:10,601 Stage-1 map = 100%,  reduce = 0
2017-03-27 04:32:11,607 Stage-1 map = 100%,  reduce = 0
2017-03-27 04:32:12,613 Stage-1 map = 100%,  reduce = 0
2017-03-27 04:32:13,622 Stage-1 map = 100%,  reduce = 0
2017-03-27 04:32:14,631 Stage-1 map = 100%,  reduce = 1
MapReduce Total cumulative CPU time: 890 msec
Ended Job = job_201703260516_0006
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to: hdfs://192.168.183.10:9000/tmp/hive-roo
Moving data to: hdfs://192.168.183.10:9000/user/hive/wa
Table default.test1 stats: [num_partitions: 0, num_file
MapReduce Jobs Launched:
Job 0: Map: 1   Cumulative CPU: 0.89 sec   HDFS Read: 3
Total MapReduce CPU Time Spent: 890 msec
OK
Time taken: 14.636 seconds
hive> select * from test1;
OK
user1   100     20170301
user3   101     20170302
user4   102     20170303
user5   103     20170304
user7   104     20170305
user8   105     20170306
Time taken: 0.07 seconds, Fetched: 6 row(s)
hive> select * from w_b;
OK
user1   100     20170301
user3   101     20170302
user4   102     20170303
user5   103     20170304
user7   104     20170305
user8   105     20170306
Time taken: 0.075 seconds, Fetched: 6 row(s)
hive> 
```

案例六：数据导出（导出为本地文件）



```
[root@master hive_test]# ls 1.txt/
000000_0
[root@master hive_test]# cat 1.txt/000000_0
user1□1
user2□1
user3□0
user4□1
user5□0
user6□1
user7□1
user8□0
```

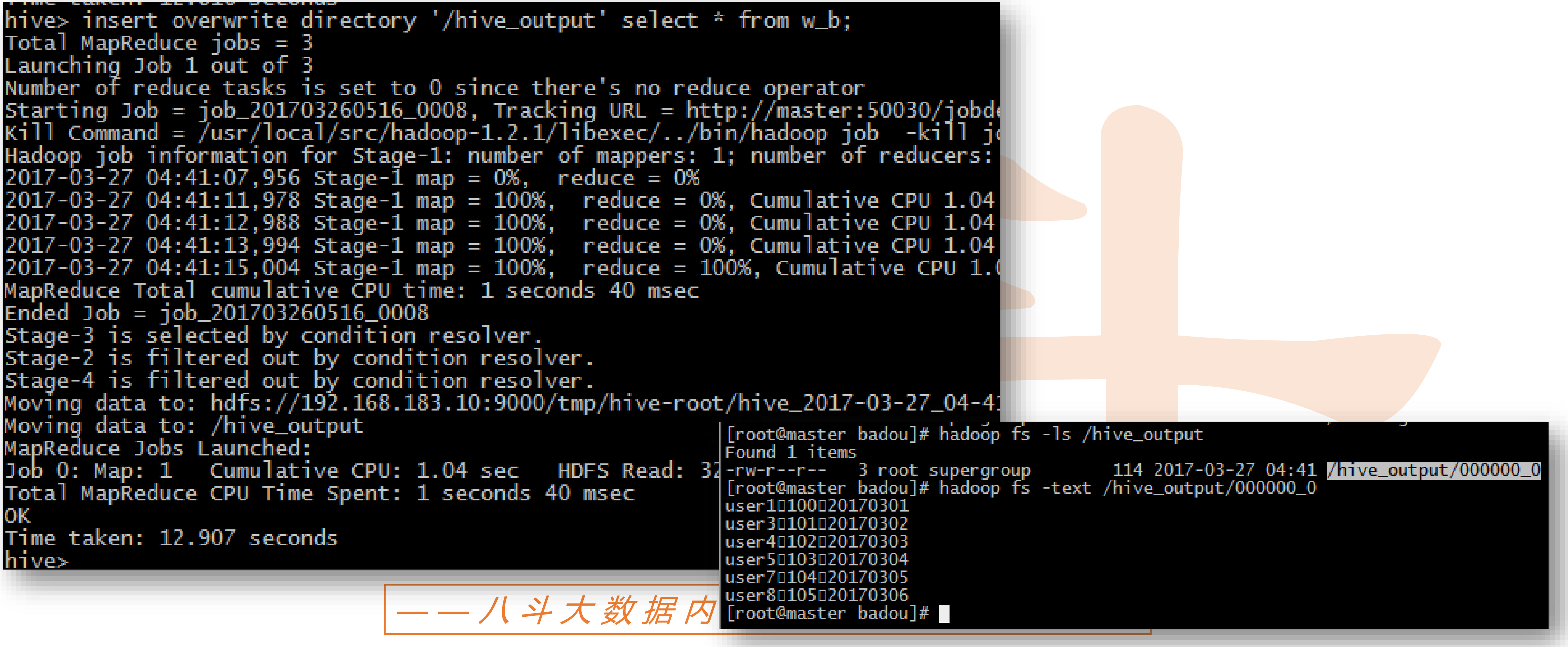
- 执行命令（例子）：
- insert overwrite local directory '/home/badou/hive_test/1.txt' select usrid, sex from w_a;



```
hive> insert overwrite local directory '/home/badou/hive_test/1.txt' select usrid, sex from w_a;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201703260516_0007, Tracking URL = http://master:50030/jobdetails.jsp?jobid=job_2017
Kill Command = /usr/local/src/hadoop-1.2.1/libexec/../bin/hadoop job  -kill job_201703260516_0007
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2017-03-27 04:36:45,821 Stage-1 map = 0%,  reduce = 0%
2017-03-27 04:36:49,841 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.96 sec
2017-03-27 04:36:50,848 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.96 sec
2017-03-27 04:36:51,853 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.96 sec
2017-03-27 04:36:52,858 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 0.96 sec
MapReduce Total cumulative CPU time: 960 msec
Ended Job = job_201703260516_0007
Copying data to local directory /home/badou/hive_test/1.txt
Copying data to local directory /home/badou/hive_test/1.txt
MapReduce Jobs Launched:
Job 0: Map: 1   Cumulative CPU: 0.96 sec   HDFS Read: 298 HDFS Write: 64 SUCCESS
Total MapReduce CPU Time Spent: 960 msec
OK
Time taken: 12.816 seconds
hive>
```

案例七：数据导出（导出为HDFS文件）

- 执行命令（例子）：insert overwrite directory '/hive_output' select * from w_b;



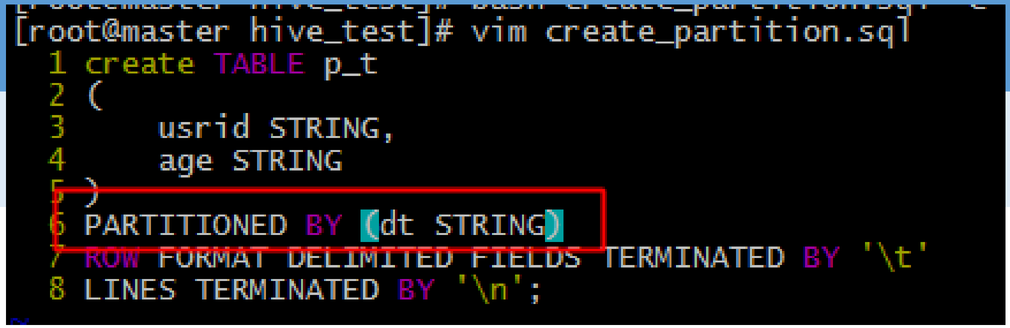
```
hive> insert overwrite directory '/hive_output' select * from w_b;
Total MapReduce jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201703260516_0008, Tracking URL = http://master:50030/jobde
Kill Command = /usr/local/src/hadoop-1.2.1/libexec/../bin/hadoop job  -kill jo
Hadoop job information for Stage-1: number of mappers: 1; number of reducers:
2017-03-27 04:41:07,956 Stage-1 map = 0%,  reduce = 0%
2017-03-27 04:41:11,978 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.04
2017-03-27 04:41:12,988 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.04
2017-03-27 04:41:13,994 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.04
2017-03-27 04:41:15,004 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 1.0
MapReduce Total cumulative CPU time: 1 seconds 40 msec
Ended Job = job_201703260516_0008
Stage-3 is selected by condition resolver.
Stage-2 is filtered out by condition resolver.
Stage-4 is filtered out by condition resolver.
Moving data to: hdfs://192.168.183.10:9000/tmp/hive-root/hive_2017-03-27_04-41
Moving data to: /hive_output
MapReduce Jobs Launched:
Job 0: Map: 1   Cumulative CPU: 1.04 sec   HDFS Read: 32
Total MapReduce CPU Time Spent: 1 seconds 40 msec
OK
Time taken: 12.907 seconds
hive>
```

```
[root@master badou]# hadoop fs -ls /hive_output
Found 1 items
-rw-r--r--   3 root supergroup        114 2017-03-27 04:41 /hive_output/000000_0
[root@master badou]# hadoop fs -text /hive_output/000000_0
user1□100□20170301
user3□101□20170302
user4□102□20170303
user5□103□20170304
user7□104□20170305
user8□105□20170306
[root@master badou]# 
```

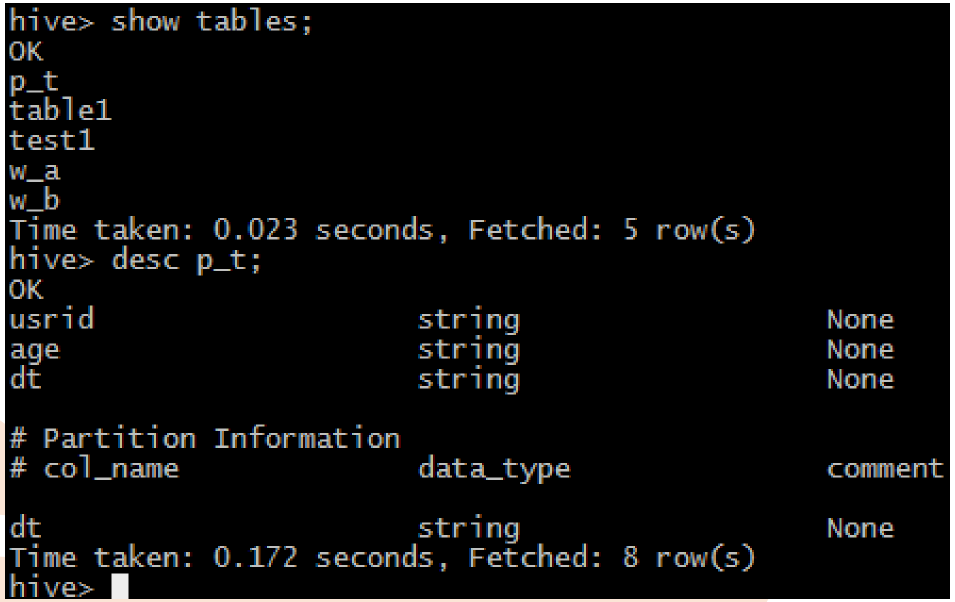
——八斗大数据内

案例八：Partition

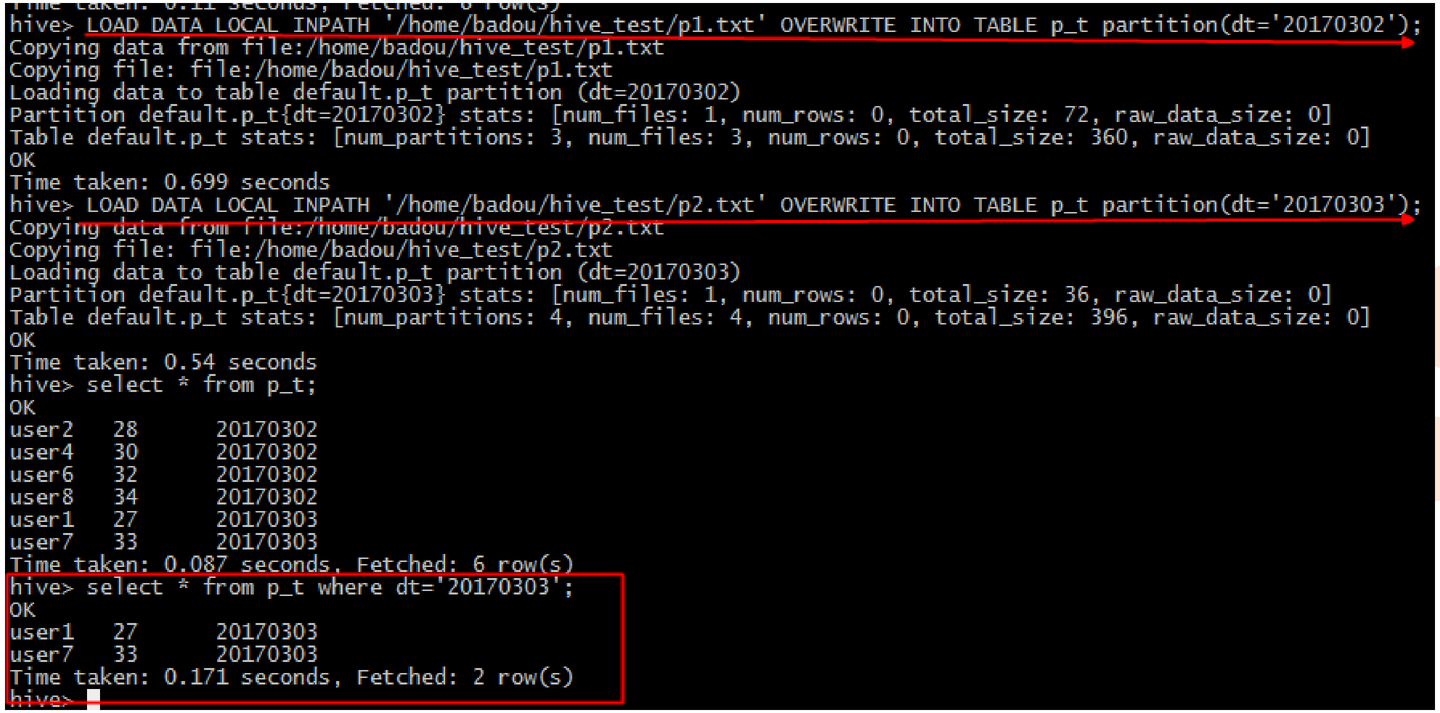
- partition是Hive提供的一种机制：用户通过指定一个或多个partition key，决定数据存放方式，进而优化数据的查询,一个表可以指定多个partition key，每个partition在hive中以文件夹的形式存在。



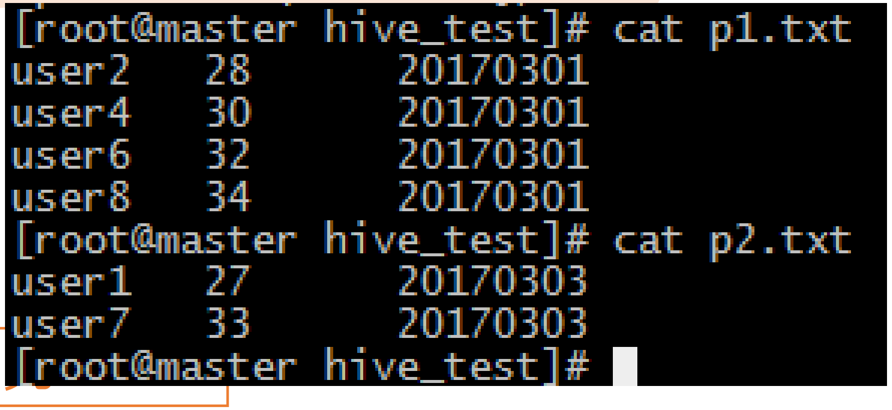
```
[root@master hive_test]# vim create_partition.sql
create TABLE p_t
(
    usrid STRING,
    age STRING
)
PARTITIONED BY (dt STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n';
```



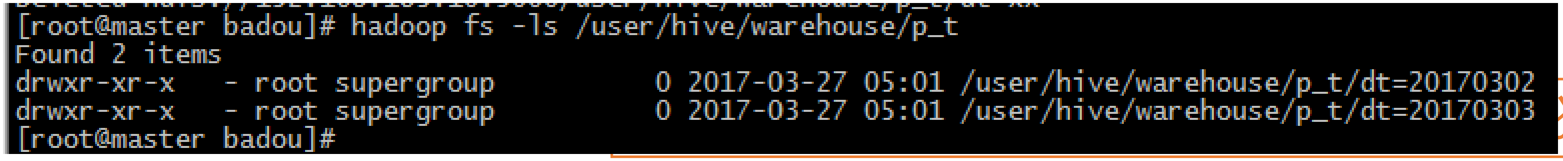
```
hive> show tables;
OK
p_t
table1
test1
w_a
w_b
Time taken: 0.023 seconds, Fetched: 5 row(s)
hive> desc p_t;
OK
usrid                   string                  None
age                     string                  None
dt                      string                  None

# Partition Information
# col_name              data_type               comment

dt                      string                  None
Time taken: 0.172 seconds, Fetched: 8 row(s)
hive>
```



```
Time taken: 0.11 seconds, Fetched: 6 row(s)
hive> LOAD DATA LOCAL INPATH '/home/badou/hive_test/p1.txt' OVERWRITE INTO TABLE p_t partition(dt='20170302');
Copying data from file:/home/badou/hive_test/p1.txt
Copying file: file:/home/badou/hive_test/p1.txt
Loading data to table default.p_t partition (dt=20170302)
Partition default.p_t{dt=20170302} stats: [num_files: 1, num_rows: 0, total_size: 72, raw_data_size: 0]
Table default.p_t stats: [num_partitions: 3, num_files: 3, num_rows: 0, total_size: 360, raw_data_size: 0]
OK
Time taken: 0.699 seconds
hive> LOAD DATA LOCAL INPATH '/home/badou/hive_test/p2.txt' OVERWRITE INTO TABLE p_t partition(dt='20170303');
Copying data from file:/home/badou/hive_test/p2.txt
Copying file: file:/home/badou/hive_test/p2.txt
Loading data to table default.p_t partition (dt=20170303)
Partition default.p_t{dt=20170303} stats: [num_files: 1, num_rows: 0, total_size: 36, raw_data_size: 0]
Table default.p_t stats: [num_partitions: 4, num_files: 4, num_rows: 0, total_size: 396, raw_data_size: 0]
OK
Time taken: 0.54 seconds
hive> select * from p_t;
OK
user2   28      20170302
user4   30      20170302
user6   32      20170302
user8   34      20170302
user1   27      20170303
user7   33      20170303
Time taken: 0.087 seconds, Fetched: 6 row(s)
hive> select * from p_t where dt='20170303';
OK
user1   27      20170303
user7   33      20170303
Time taken: 0.171 seconds, Fetched: 2 row(s)
hive>
```



```
[root@master hive_test]# cat p1.txt
user2   28      20170301
user4   30      20170301
user6   32      20170301
user8   34      20170301
[root@master hive_test]# cat p2.txt
user1   27      20170303
user7   33      20170303
[root@master hive_test]#
```

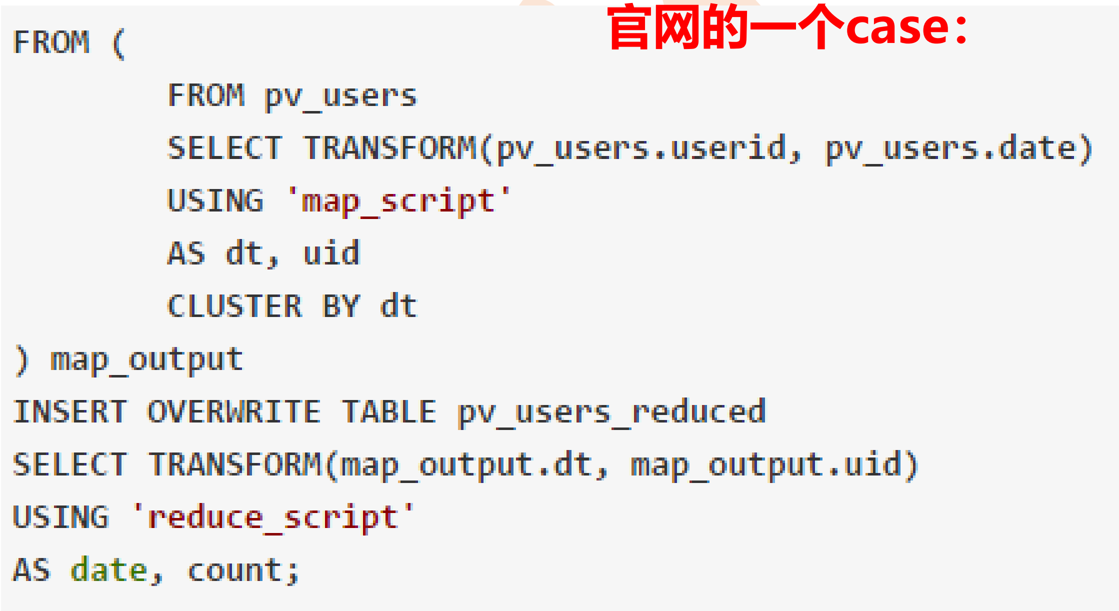


```
[root@master badou]# hadoop fs -ls /user/hive/warehouse/p_t
Found 2 items
drwxr-xr-x   - root supergroup          0 2017-03-27 05:01 /user/hive/warehouse/p_t/dt=20170302
drwxr-xr-x   - root supergroup          0 2017-03-27 05:01 /user/hive/warehouse/p_t/dt=20170303
[root@master badou]#
```

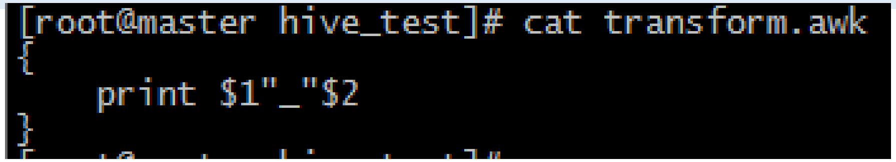
案例九：Transform

- transform功能部分可以用UDF替代，但是如果拼接的字段是根据上一次查询的结果时，UDF就不能用，UDF只能用在本行操作
- transform功能缺点是效率底了点

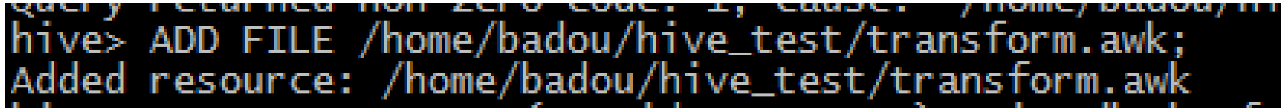
官网的一个case：



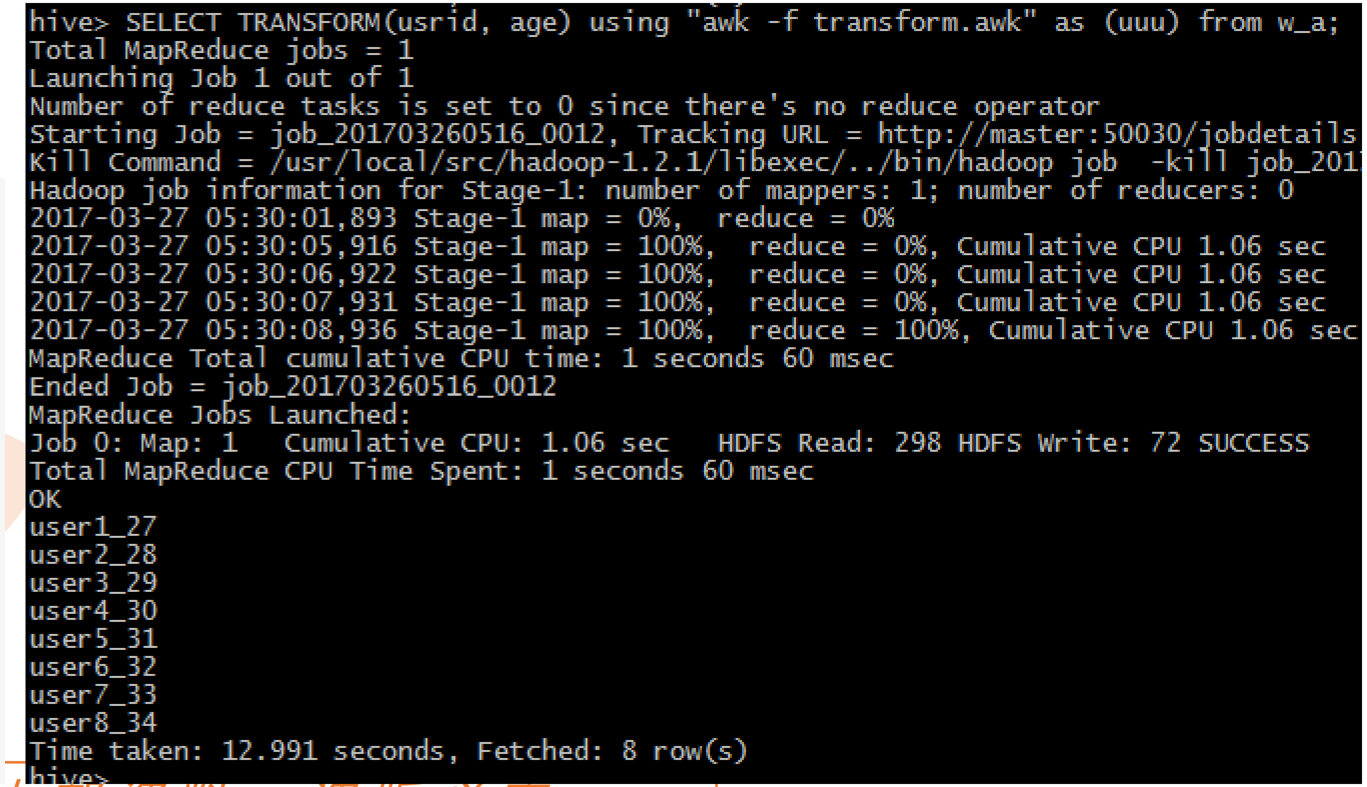
```
FROM (
        FROM pv_users
        SELECT TRANSFORM(pv_users.userid, pv_users.date)
        USING 'map_script'
        AS dt, uid
        CLUSTER BY dt
) map_output
INSERT OVERWRITE TABLE pv_users_reduced
SELECT TRANSFORM(map_output.dt, map_output.uid)
USING 'reduce_script'
AS date, count;
```



```
[root@master hive_test]# cat transform.awk
{
    print $1"_"$2
}
```

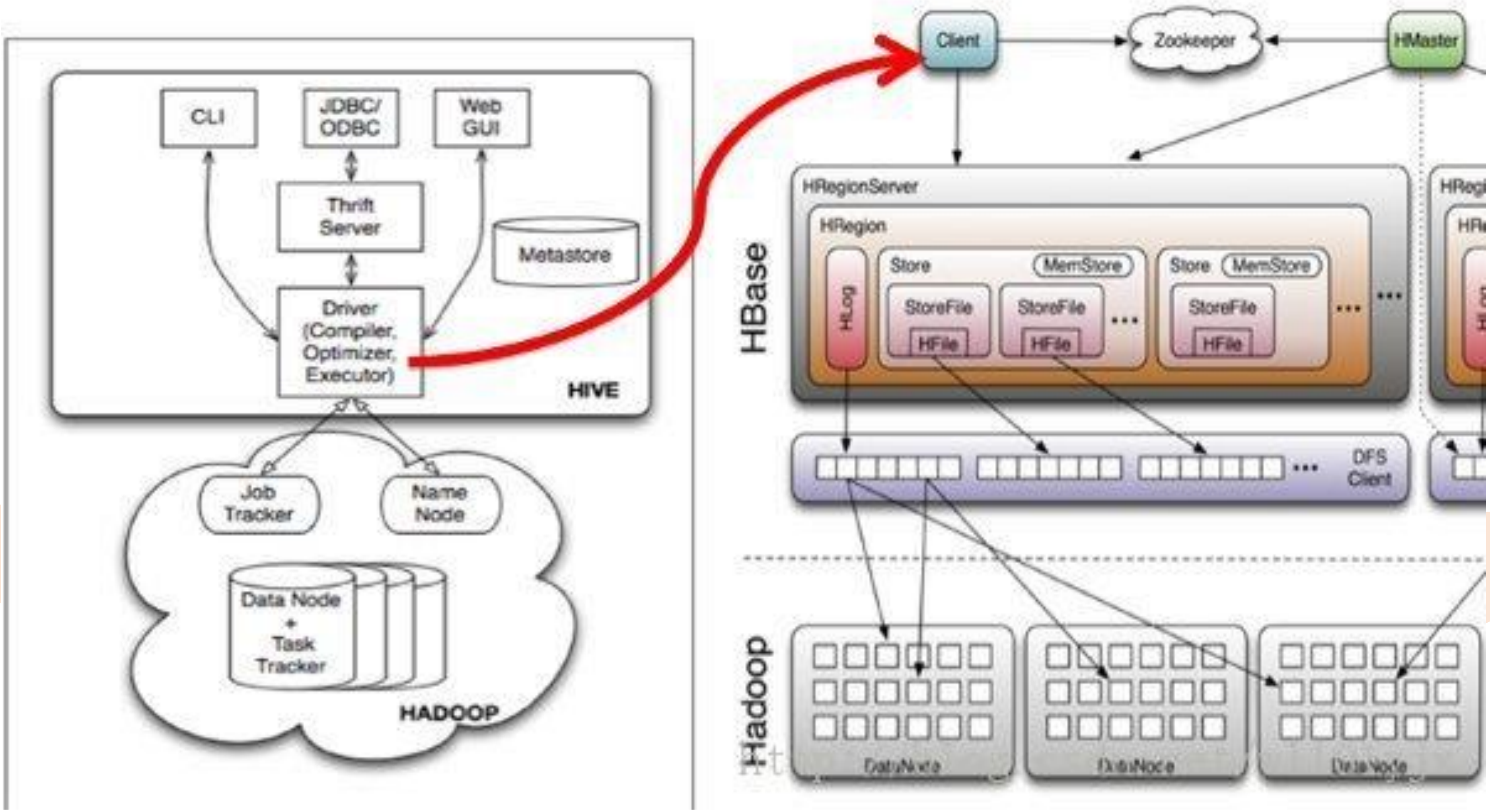


```
Query returned non-zero code: 1, cause: /home/badou/hi
hive> ADD FILE /home/badou/hive_test/transform.awk;
Added resource: /home/badou/hive_test/transform.awk
```



```
hive> SELECT TRANSFORM(usrid, age) using "awk -f transform.awk" as (uuu) from w_a;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201703260516_0012, Tracking URL = http://master:50030/jobdetails
Kill Command = /usr/local/src/hadoop-1.2.1/libexec/../bin/hadoop job  -kill job_201
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2017-03-27 05:30:01,893 Stage-1 map = 0%,  reduce = 0%
2017-03-27 05:30:05,916 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.06 sec
2017-03-27 05:30:06,922 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.06 sec
2017-03-27 05:30:07,931 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.06 sec
2017-03-27 05:30:08,936 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 1.06 sec
MapReduce Total cumulative CPU time: 1 seconds 60 msec
Ended Job = job_201703260516_0012
MapReduce Jobs Launched:
Job 0: Map: 1   Cumulative CPU: 1.06 sec   HDFS Read: 298 HDFS Write: 72 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 60 msec
OK
user1_27
user2_28
user3_29
user4_30
user5_31
user6_32
user7_33
user8_34
Time taken: 12.991 seconds, Fetched: 8 row(s)
hive>
```

案例十：Hive整合Hbase



——八斗大数据内部资料，盗版必究——

案例十：Hive整合Hbase

- 创建Hbase表：
  - create 'classes','user'
- 加入数据：
  - put 'classes','001','user:name','jack'
  - put 'classes','001','user:age','20'
  - put 'classes','002','user:name','liza'
  - put 'classes','002','user:age','18'

案例十： Hive整合Hbase

- 创建Hive表并验证：
  - create external table classes(id int, name string, age int)
  - STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
  - WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,user:name,user:age")
  - TBLPROPERTIES("hbase.table.name" = "classes");
- 再添加数据到Hbase：
  - put 'classes','003','user:age','1820183291839132'

Q&A

@八斗学院