模块八 Hive优化



《Hive数据仓库应用教程》

目 录 / CONTENTS

1 任务8-1 Hive参数优化询

2 任务8-2 HQL优化

3 任务8-3 本章小结

4 任务8-4 本章习题

任务 8-1 Hive参数优化

8.1.1 任务描述

在大数据技术生态系统中，Hive参数调优在实际运行中，时时遇到。企业级的数据平台，随着数据规模的不断增长，如果持续更高效率地运行下去，需要根据实际情况进行优化。Hive拥有较多在特定情况下优化的特性，如何利用好相关特性，是Hive参数调优的关键。







8.1.2 任务准备知识

为了避免与前面几个章节数据库中的数据发生冲突，在本章节中，重新创建一个新的数据库，名称为unit8db。本章所有的案例操作，均基于该数据库，此处请读者注意。同时，在本章中所需的资源，本地文件资源均存储在/usr/datas8目录中，HDFS文件系统中的资源均存储于/tmp目录中，此处请读者注意。

为了更好的说明本小节的内容，需要说明该数据库unit8db的创建过程。但是具体的创建过程在第五章的例题1中，已经详细说明，故此在此不再赘述。

例题 1：创建本小节所需要的数据库 unit8db。

操作过程：

step01：创建数据库 unit8db。hive> CREATE DATABASE IF NOT EXISTS unit8db;

step02：查看数据库unit8db。hive> SHOW DATABASES;

8.1.2 任务准备知识

step03：使用数据库unit8db。hive> USE unit8db;创建数据库unit8db、使用数据库unit8db的操作过程，以上步骤的操作的具体执行和操作结果如图8-1所示，由操作结果可以看出，创建数据库unit8db成功、使用数据库unit8db成功。

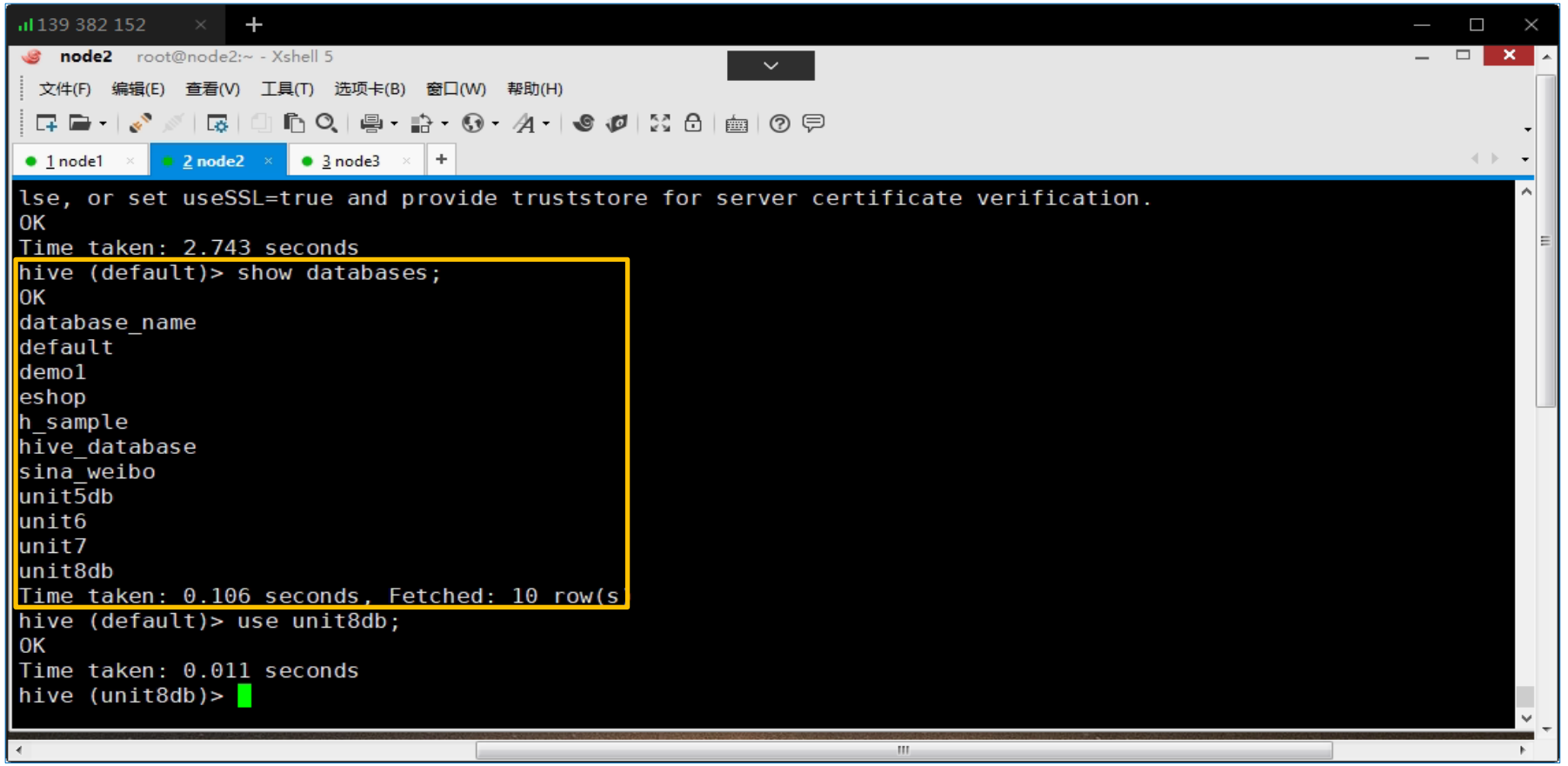

图 8-1 创建数据库 unit8db、使用数据库 unit8db

8.1.3 任务实施

Hive 参数优化的方式主要有本地模式、配置 MapReduce 压缩、配置 Map 个数、配置 Reduce 个数、合并小文件等五种。

一、本地模式

大多数的Hadoop Job 是需要Hadoop 提供的完整的可扩展性来处理大数据集的。不过，有时 Hive 的输入数据量是非常小的。在这种情况下，为查询触发执行任务消耗的时间可能会比实际 Job 的执行时间要多的多。对于大多数这种情况，Hive 可以通过本地模式在单台机器上处理所有的任务。对于小数据集，执行时间可以明显被缩短。

8.1.3 任务实施

例题 1：使用本地模式和默认模式分别查询表 bigtable。

操作过程：

step01：　　创建数据库 unit8db 中表 bigtable。创建数据库 unit8db 中的表 bigtable，具体操作命令如下：

```
hive>　CREATETABLE　bigtable
　( id　bigint,　t　bigint,　uid　string,　keyword　　string, url_rank int,
　　click_num int,　　click_url string)
　ROW　　FORMAT　　DELIMITED　　FIELDS TERMINATED　by ',';
```

查询表 bigtable 是否创建成功，成功后，向表中加载数据，如图 8-2 和图 8-3所示，表 bigtable 中共有 320 条数据，此处只显示前 15 条数据。具体操作命令如下：

```
hive>　LOAD　DATA　　LOCAL inpath '/usr/datas8/bigtable001.csv' INTO
　　TABLE　bigtable;
```

8.1.3 任务实施

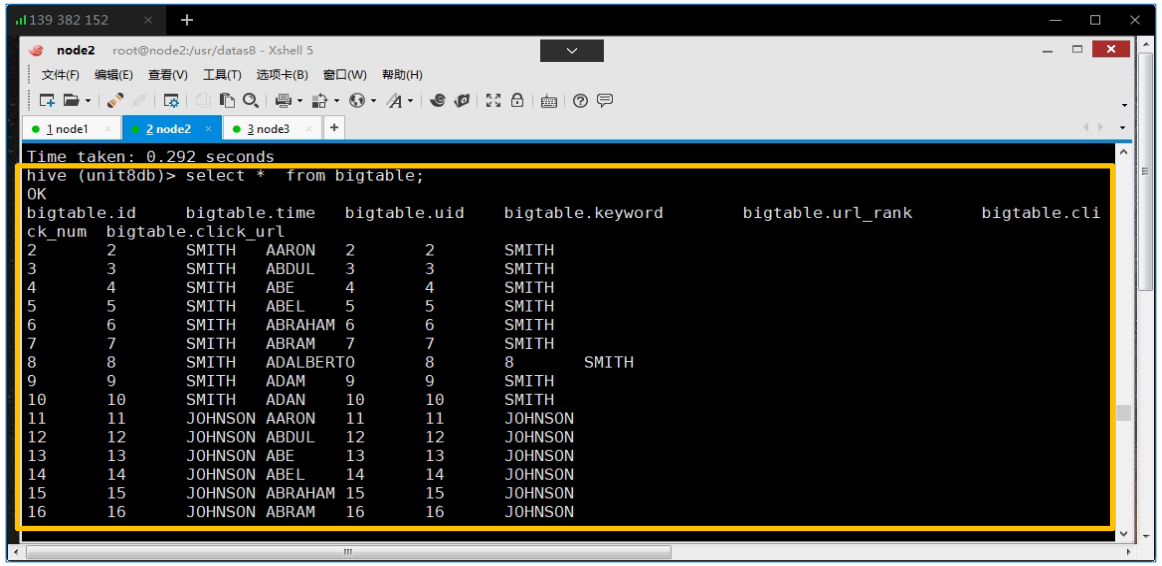

图 8-2

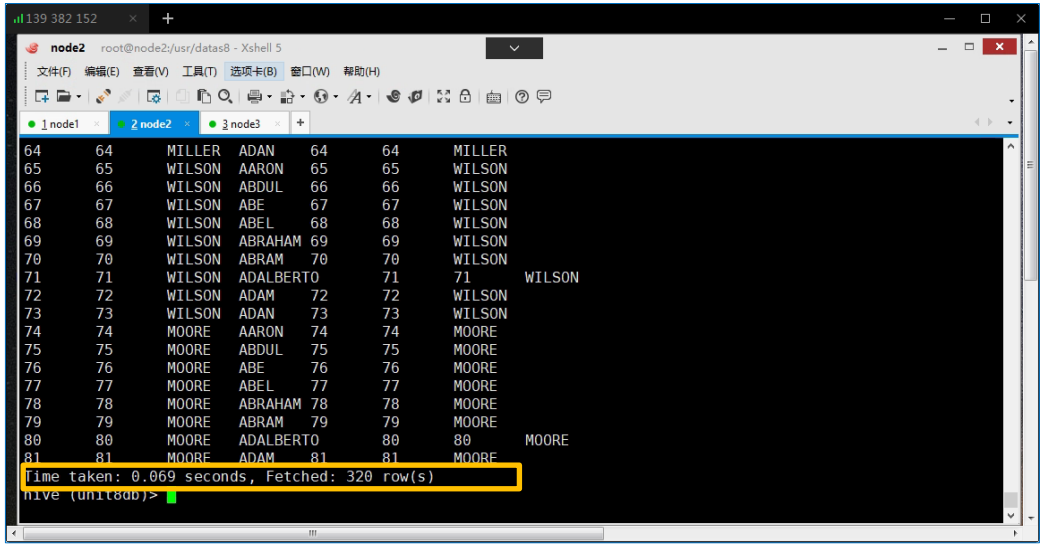

图 8-3

step02：　关闭本地模式，使用默认模式，查询表 bigtable 中的数据关闭本地模式，使用默认模式，具体操作命令如下：hive>　SET　hive.exec.mode.local.auto;

8.1.3 任务实施

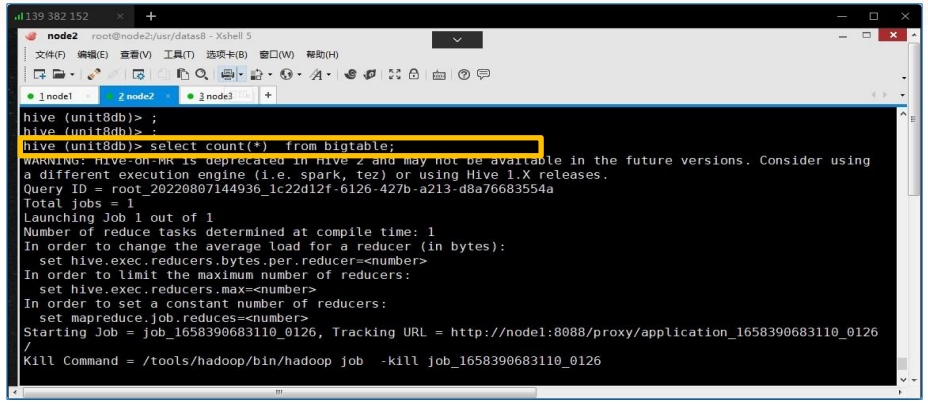

图 8-4

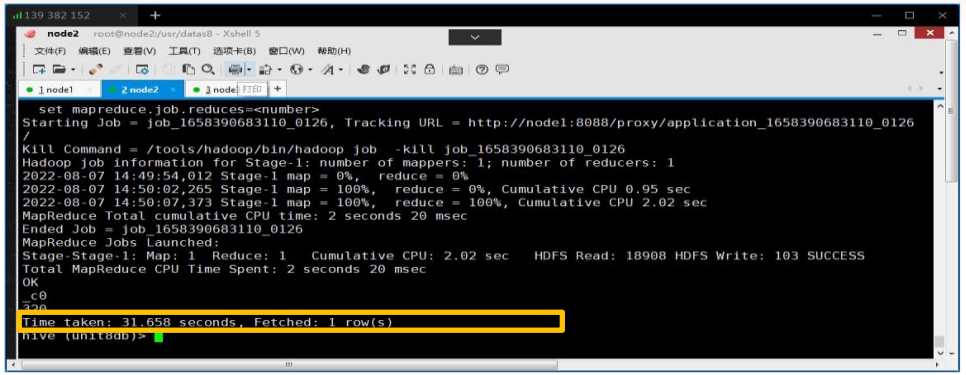

图 8-5

然后，查询表 bigtable 中的数据，如图 8-4 和图 8-5 所示，由结果可知，时间为 31.658 seconds，具体操作命令如下：hive> SELECT count(*) from bigtable;

step03： 关闭默认模式，使用本地模式，查询表 bigtable 中的数据关闭默认模式，使用本地模式，具体操作命令如下：hive> SET hive.exec.mode.local.auto=true;

然后，查询表 bigtable 中的数据，如图 8-6 和图 8-7 所示，由结果可知，时间为 13.028 seconds，具体操作命令如下：hive> SELECT count(*) from bigtable;

8.1.3 任务实施

然后，查询表 bigtable 中的数据，如图 8-6 和图 8-7 所示，由结果可知，时间为 13.028 seconds，具体操作命令如下：hive>　SELECT count(*)　from　bigtable;

8.1.3 任务实施

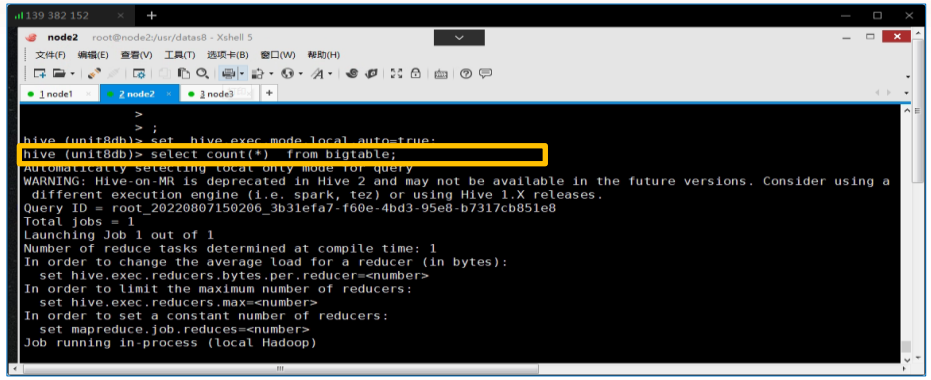

图 8-6

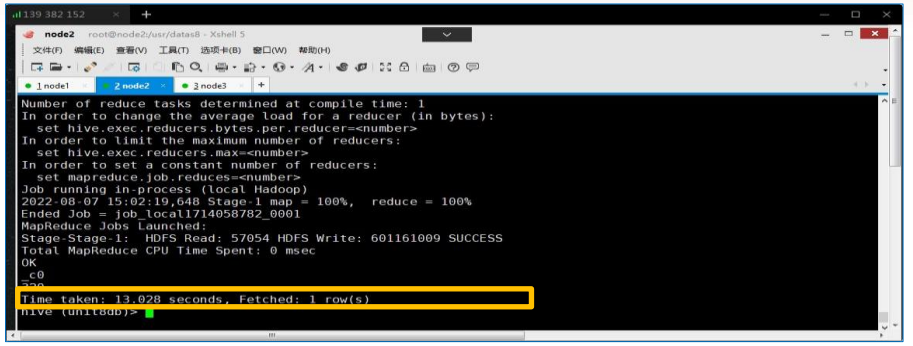

图 8-7

当一个 MapReduce 任务的数据量和计算任务很小的时候，在 MapReduce 框架中 Map 任务和 Reduce 任务的启动过程占用了任务执行的大部分时间，真正的逻辑处理其实占用时间很少，但是给用户的感受就是：很小的任务，同样执行较长的时间。

那么在 0.7 版本之后，Hive 引入了本地模式，那么对于小的任务的执行， Hive 客户端不再需要到 Yarn 上申请 Map 任务和 Reduce 任务，只需要在本地进行 Map 和 Reduce 的执行，大大的加快了小的任务的执行时间，通常可以把分钟级别任务的执行时间降低秒级。

8.1.3 任务实施

二、配置 MapReduce 压缩

MapReduce 的主要压缩过程有三个阶段：map 之前、map 之后、reduce 之后。

其中，

1. 压缩的原则

1) 运算密集型的 Job，少用压缩

2) IO 密集型的 Job，多用压缩

2. 压缩算法和性能对比分别如表 8-2 和表 8-3 所示

压缩格式	Hadoop 自带？	算法	文件扩展名	是否可切片	换成压缩格式后，原来的程序是否需要修改
DEFLATE	是，直接使用	DEFLATE	.DEFLATE	否	和文本处理一样，不需要修改
Gzip	是，直接使用	DEFLATE	.gz	否	和文本处理一样，不需要修改
bzip2	是，直接使用	bzip2	.bz2	是	和文本处理一样，不需要修改
LZO	否，需要安装	LZO	.lzo	是	需要建索引，还需要指定输入格式
Snappy	是，直接使用	Snappy	.snappy	否	和文本处理一样，不需要修改

8.1.3 任务实施

表8-3 压缩性能的对比

压缩算法	原始文件大小	压缩文件大小	压缩速度	解压速度
Gzip	8.3GB	1.8GB	17.5MB/s	58MB/s
bzip2	8.3GB	1.8GB	2.4MB/s	9.5MB/s
LZO	8.3GB	1.8GB	49.3MB/s	74.6MB/s

8.1.3 任务实施

3. 压缩算法选择

需要考虑压缩/解压缩速度、压缩率（压缩后存储大小）、压缩后是否可以支持切片

4. 压缩参数配置，包含 Hadoop 编码/解码器和参数配置，分别如表 8-4 和表 8-5所示

表 8-4 Hadoop 编码/解码器

压缩格式	对应的编码/解码器
DEFLATE	org.apache.hadoop.io.compress.DefaultCodec
Gzip	org.apache.hadoop.io.compress.GzipCodec
bzip2	org.apache.hadoop.io.compress.BZip2Codec
LZO	com.hadoop.compression.lzo.LzopCodec
Snappy	org.apache.hadoop.io.compress.SnappyCodec

8.1.3 任务实施

表 8-5 参数配置

参数	默认值	阶段	建议
io.compression.codecs（在 core-site.xml 中配置）	无，这个需要在命令行输入 hadoop checknative 查看	输出压缩	Hadoop 使用文件扩展名判断是否支持某种编解码器
mapreduce.map.output.compress（在 mapred-site.xml 中配置）	false	map 输出	这个参数设为 true 启用压缩
mapreduce.map.output.compress.codec（在 mapred-site.xml 中配置）	org.apache.hadoop.io.compress.DefaultCodec	map 输出	企业多使用 LZO 或 Snappy 编解码器在此阶段压缩数据
mapreduce.output.fileoutputformat.compress（在 mapred-site.xml 中配置）	false	reducer 输出	这个参数设为 true 启用压缩
mapreduce.output.fileoutputformat.compress.codec（在 mapred-site.xml 中配置）	org.apache.hadoop.io.compress.DefaultCodec	reducer 输出	使用标准工具或者编解码器，如 gzip 和 bzip2

8.1.3 任务实施

因此，根据 MapReduce 的工作原理，如图 8-8 所示，可知，压缩共分为三个阶段。



图 8-8 MapReduce 的工作原理

1. Map 之前：要考虑数据量的大小，数据量小的，就不需要考虑过多的切片，主要追求的是快速，可以选择压缩方式 snappy 或者 LZO。数据量大的话，考虑到切片，可以选择 LZO 或者 bzip2
2. Map 之后：map 之后的数据要经历 reduce 聚合，主要考虑速度，所以选择 snappy 或者 LZO
3. Reduce 之后：需要看具体的需求，类似于持久保存，那就需要看数据量的大小，数据量小的，就不需要考虑过多的切片，主要追求的是快速，可以选择压缩方式 Snappy 或者 LZO。数据量大的话，考虑到切片，可以选择 LZO 或者 bzip2

三、配置 Map 和 Reduce 个数

通常情况下，作业会通过 input 的目录产生一个或者多个 Map 任务。主要的决定因素有：input 的文件总个数，input 的文件大小，集群设置的文件块大小。并不是 Map 数越多越好。如果一个任务有很多小文件（远远小于块大小 128m），则每个小文件被视为一个块，用一个 Map 任务来完成，而一个 Map 任务启动和初始化的时间远远大于逻辑处理的时间，就会造成很大的资源浪费。而且，同时可执行的 Map 数是受限的。

并不是保证每个 Map 处理接近 128M 的文件块，性能就一定高，就可以高枕无忧了。比如有一个 127M 的文件，正常使用一个 Map 去完成，但这个文件只有一个或者两个小字段，却有几千万的记录，如果 Map 处理的逻辑比较复杂，用一个 Map 任务去做，肯定也比较耗时。

8.1.3 任务实施

针对上面的问题，我们需要采取两种方式来解决：即减少 Map 数和增加 Map数；

1. 复杂文件增加 Map 数

Map 的数量通常是由 Hadoop 集群的 DFS 块大小确定的，也就是输入文件的总块数，正常的 Map 数量的并行规模大致是每一个Node 是 10~100 个，对于 CPU消耗较小的作业可以设置 Map 数量为 300 个左右，但是由于 Hadoop 的没一个任务在初始化时需要一定的时间，因此比较合理的情况是每个 Map 执行的时间至少超过 1 分钟。

具体的数据分片是这样的，InputFormat 在默认情况下会根据 Hadoop 集群的 DFS 块大小进行分片，每一个分片会由一个 Map 任务来进行处理，当然用户还是可以通过参数 mapred.min.split.size 参数在作业提交客户端进行自定义设置。

还有一个重要参数就是 mapred.map.tasks，这个参数设置的 map 数量仅仅是一个提示，只有当 InputFormat 决定了 map 任务的个数比 mapred.map.tasks 值小时才起作用。

8.1.3 任务实施

同样，Map 任务的个数也能通过使用 JobConf 的 conf.setNumMapTasks(int num)方法来手动地设置。这个方法能够用来增加 Map 任务的个数，但是不能设定任务的个数小于 Hadoop 系统通过分割输入数据得到的值。

当然为了提高集群的并发效率，可以设置一个默认的 Map 数量，当用户的Map 数量较小或者比本身自动分割的值还小时可以使用一个相对交大的默认值，

从而提高整体 Hadoop 集群的效率。

当 input 的文件都很大，任务逻辑复杂，Map 执行非常慢的时候，可以考虑增加 Map 数，来使得每个 map 处理的数据量减少，从而提高任务的执行效率。

增加 Map 的方法为：根据 computeSliteSize(Math.max(minSize,Math.min(maxSize,blocksize)))=blocksize=128 M 公式，调整 MaxSize 最大值。让 MaxSize 最大值低于 BlockSize 就可以增加Map的个数。

8.1.3 任务实施

例题2：使用增加Map数的方式给表emp调优。

操作过程：

创建数据库unit8db中表emp

创建数据库unit8db中的表emp，具体操作命令如下：

```
hive> CREATE TABLE IF NOT EXISTS emp(
empno int, ename string, job string, mgr int,
hiredate string, sal double, comm double, deptno int)
ROW FORMAT DELIMITED FIELDS TERMINATED by '\t';
```

查询表emp是否创建成功，成功后，向表中加载数据，如图8-9所示。具体操作命令如下：

```
hive> LOAD DATA LOCAL inpath '/usr/datas8/emp.txt'
INTO TABLE emp;
```

8.1.3 任务实施

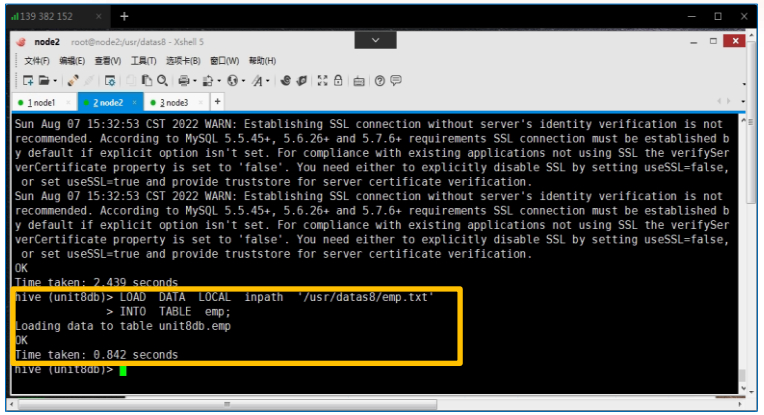

图 8-9

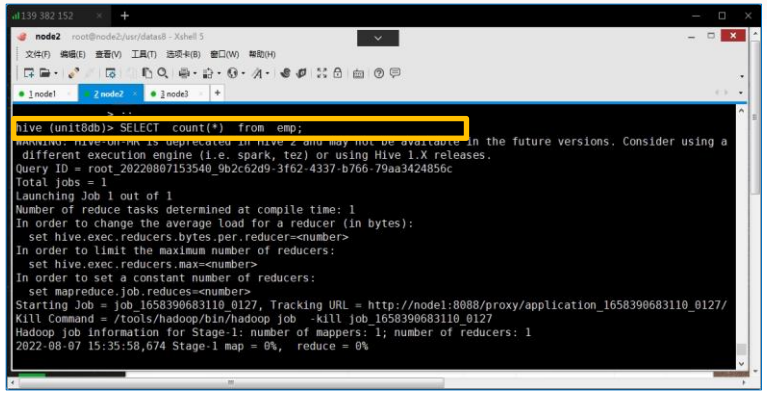

图 8-10

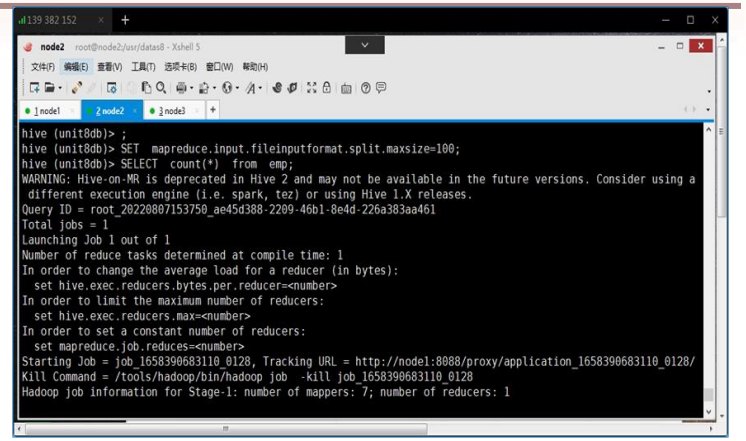

图8-11

step02： 对表 emp 执行查询操作

查询数据库 unit8db 中的表 emp，如图 8-10 所示，具体操作命令如下：

hive> SELECT count(*) from emp;

step03： 设置最大切片值为 100 字节

对开发环境进行设置，设置最大切片值为 100 字节，具体操作命令如下：

hive> SET mapreduce.input.fileinputformat.split.maxsize=100;

然后，再次查询数据库 unit8db 中的表 emp，如图 8-11 所示，具体操作命令如下：

hive> SELECT count(*) from emp;

2. 小文件进行合并

例题 3：小文件进行合并。**操作过程：**

1) 在 map 执行前合并小文件，减少 map 数：CombineHiveInputFormat 具有对小文件进行合并的功能（系统默认的格式）。HiveInputFormat 没有对小文件合并功能。具体操作命令如下：

```
hive>SET hive.input.format=org.apache.hadoop.hive.ql.io.CombineHiveInputFormat;
```

2) 在 Map-Reduce 的任务结束时合并小文件的设置：

- 在 map-only 任务结束时合并小文件，默认为 true，具体操作命令如下：
  ```
  hive> SET hive.merge.mapfiles=true;
  ```
- 在 map-reduce 任务结束时合并小文件，默认为 false，具体操作命令如下：
  ```
  hive> SET hive.merge.mapredfiles=true;
  ```
- 合并文件的大小，默认为 256M，具体操作命令如下：
  ```
  hive>SET hive.merge.size.per.task=268435456;
  ```

8.1.3 任务实施

当输出文件的平均大小小于该值时，启动一个独立的 map-reduce 任务进行文件merge，具体操作命令如下：hive> SET hive.merge.smallfiles.avgsize=16777216;

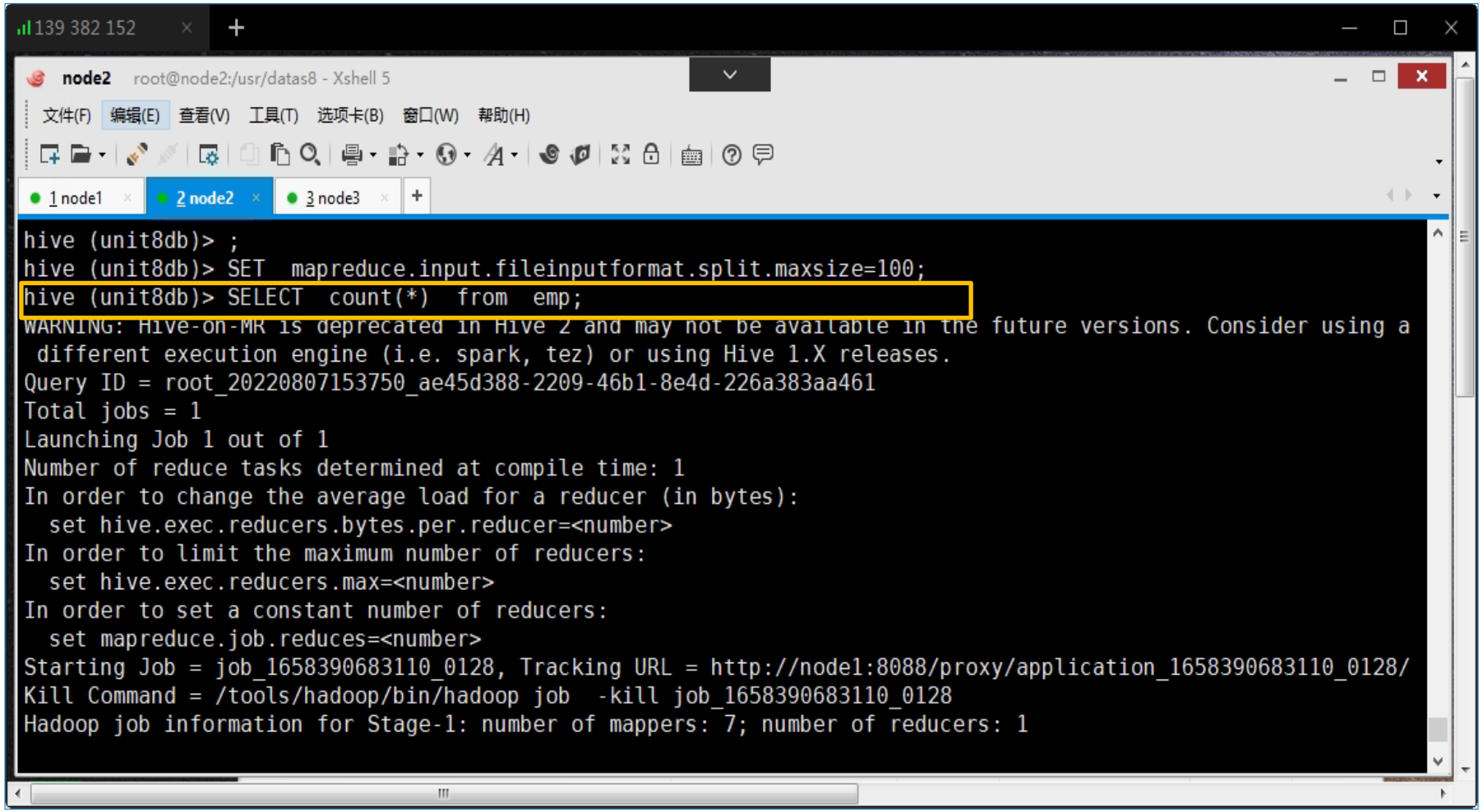

图 8-11 再次查询数据库的表 emp 中数据

8.1.3 任务实施

2. 小文件进行合并

例题 3：小文件进行合并。**操作过程：**

1) 在 map 执行前合并小文件，减少 map 数：CombineHiveInputFormat 具有对小文件进行合并的功能（系统默认的格式）。HiveInputFormat 没有对小文件合并功能。具体操作命令如下：

```
hive> SET hive.input.format=
    org.apache.hadoop.hive.ql.io.CombineHiveInputFormat;
```

2) 在 Map-Reduce 的任务结束时合并小文件的设置：

- 在 map-only 任务结束时合并小文件，默认为 true，具体操作命令如下：

```
hive> SET hive.merge.mapfiles=true;
```

- 在 map-reduce 任务结束时合并小文件，默认为 false，具体操作命令如下：

```
hive> SET hive.merge.mapredfiles=true;
```

- 合并文件的大小，默认为 256M，具体操作命令如下：

```
hive> SET hive.merge.size.per.task=268435456;
```

当输出文件的平均大小小于该值时，启动一个独立的 map-reduce 任务进行文件merge，具体操作命令如

3. 合理设置 Reduce 数

reduce 在运行时往往需要从相关 map 端复制数据到 reduce 节点来处理，因此相比于 map 任务。reduce 节点资源是相对比较缺少的，同时相对运行较慢，正确的 reduce 任务的个数应该是 0.95 或者 1.75*（节点数 × mapred.tasktracker.tasks.maximum 参数值）。

如果任务数是节点个数的 0.95 倍，那么所有的 reduce 任务能够在 map 任务的输出传输结束后同时开始运行。如果任务数是节点个数的 1.75 倍，那么高速的节点会在完成他们第一批 reduce 任务计算之后开始计算第二批 reduce 任务，这样的情况更有利于负载均衡。

同时需要注意增加 reduce 的数量虽然会增加系统的资源开销，但是可以改善负载匀衡，降低任务失败带来的负面影响。

同样，Reduce 任务也能够与 map 任务一样，通过设定 JobConf 的 conf.setNumReduceTasks(int num)方法来增加任务个数。

8.1.3 任务实施

例题 4：合理设置 Reduce 数。**操作过程：**

1) 调整 reduce 个数方法一

- 每个 Reduce 处理的数据量默认是 256MB，具体操作命令如下：

  hive.exec.reducers.bytes.per.reducer=256000000

- 每个任务最大的 reduce 数，默认为 1009，具体操作命令如下：hive.exec.reducers.max=1009
- 计算 reducer 数的公式，具体操作命令如下：N=min(参数 2，总输入数据量/参数 1)

2) 调整 reduce 个数方法二

- 在 hadoop 的 mapred-default.xml 文件中修改，设置每个 job 的 Reduce 个数，具体操作命令如下：

  Hive> set mapreduce.job.reduces = 15;

3) Reduce 个数并不是越多越好过多的启动和初始化 reduce 也会消耗时间和资源；

另外，有多少个reduce，就会有多少个输出文件，如果生成了很多个小文件，那么如果这些小文件作为下一个任务的输入，则也会出现小文件过多的问题；

在设置reduce个数的时候也需要考虑这两个原则：处理大数据量利用合适的reduce数；使单个reduce任务处理数据量大小要合适；

8.1.3 任务实施

四、合并小文件

执行 MapReduce 任务时，通过会作一些优化，比如增加 Reduce 并行度来提升计算效率。而 Reduce 的增多意味着输出结果文件的增多，所以会产生小文件问题。

NameNode 存储元数据信息，包括 HDFS 的抽象目录树结构，存储数据和 block 分块对应关系、位置、大小等，启动时会加载到内存。每个存储对象大约 150 字节，所以当文件达到千万级时，内存中的元数据达到 3G 以上，NameNode的性能就会受影响。

HDFS文件读写时，会先向NameNode获取元数据信息，再通过元数据找到block对应的 DataNode 进行文件读写，大量的小文件会导致 HDFS 的读写受到影响。

执行 Hive 任务时，每个文件执行一个 MapTask，小文件数据过多，会导致大量 MapTask 的不断启动和初始化，这样时间远远大于逻辑处理时间，导致 Hive任务时间太长，造成资源浪费，甚至 oom。执行 Hive 任务时，如果发现数据量不大但是任务数很多时，应该注意在 Map 的输入进行小文件合并。

8.1.3 任务实施

Hive 优化方式

1) 输入合并，也就是在 Hive 读取文件时进行小文件合并，减少 Map 数量，。

2) 输出合并，在 MapReduce 输出结果时进行小文件合并。

3) JVM 重用，一个 Container 中运行多个 MapTask，减少 Container 开启和注销时间。

8.1.3 任务实施

例题 5：合并小文件。**操作过程：**

1) 输入合并

- 设置 hive 输入，执行 map 前进行小文件合并，设置后，DataNode 上多个小文件会进行合并，合并文件数由 mapred.max.split.size 决定，具体操作如下： Hive> SET hive.input.format=org.apache.hadoop.hive.a1.io.CombineHiveInputFormat;
- 每个 MapTask 最大文件大小，具体操作如下：
  Hive> SET mapred.max.split.size=256000000;
- 节点上最小分片大小，如果输入文件在一个节点上多个 block 的总和，大于这个值则创建一个 split，每个节点最多开启一个 split，小于则将块用于机架设置值进行计算，具体操作如下：
  Hive> SET mapred.min.split.size.per.node=100000000;
- 机架上最小分片大小。决定不同机架的 DataNode 上文件是都进行合并，对于同一机架上多个 DataNode 的小于节点设置值的块总和进行计算，大于这个值则创建 split，小于则进行多个机架计算，具体操作如下：
  Hive> SET mapred.min.split.size.per.rack=100000000;

2) 输出合并

- Map 结束时合并小文件，默认开启，具体操作如下：

  Hive> SET hive.merge.mapredfiles=true;

- 输出结果时合并，默认关闭

  Hive> SET hive.merge.mapfiles=true;

- 小于这个值会开启一个独立的 mapreduce 任务进行小文件合并，默认 16m Hive> SET hive.merge.smallfiles.avgsize= 16000000;

- 合并后的文件大小，默认 256m，推荐 128m，一个 hdfs 分块的大小

  Hive> SET hive.merge.size.per.task=256000000;

3) JVM 重用

- 设置 mapred. job. jvm. num. tasks 的值，具体操作如下：

  Hive> SET mapred.job.jvm.num.tasks

任务 8-2

HQL优化

8.2.1 任务描述

Hive用户面临的一个最大的挑战是运行即席查询的最终用户要面对较慢的响应时间。与传统关系数据库查询所达到的性能相比，Hive的响应速度通常慢的令人无法接受，那么如何才能达到最终用户所习惯的那种性能，Hive SQL的性能优化是重中之重。







8.2.2 任务准备知识

本小节将会分成几种不同的优化方式来介绍Hive SQL的性能优化。一些调优涉及到调整配置参数的值，而其他一些调优过程则包括启用或者禁用某些特定的特性。

8.2.3 任务实施

Hive SQL 的性能优化的调优方式多样，本小节主要从 Group By 的优化、Join相关的优化、Count（Distinct）优化、行/列过滤优化等四种方式进行说明。

一、Group By 优化

Hive 的优化分为“Join 相关的优化”和“Join 无关的优化”。从项目实际来说，Join 相关的优化其实占据了 Hive 优化的大部分内容，而 join 相关的优化又分为 mapjoin 可以解决的 Join 优化和 mapjoin 无法解决的 Join 优化。其实在实际 Hive SQL 开发的过程中，Hive SQL 性能的问题上实际只有小部分和数据倾相关。很多时候，Hive SQL 运行得慢是由开发人员对于使用的数据了解不够以及一些不良的使用习惯引起的。

Hive SQL 性能问题基本上大部分都和 Join 相关，对于和 Join 无关的问题主要有 Group By 相关的倾斜和 Count Distinct 相关的优化。Group By 引起的倾斜主要是输入数据行按照“Group By 列分布不均匀”引起的。默认情况下，map 阶段同一 key 数据分发给同一 reduce，如果单一 key 过大就很容易造成数据倾斜。（100 条数据分组后一组 90 条，一组 10 条，这就会造成严重的数据倾斜）。这样的话，我们就可以预先在 map 端进行一些聚合操作，减轻 reduce 端的压力。

8.2.3 任务实施

例题6：Group By优化案例。

操作过程：

假设，表test（此表在数据库中并未存在，此处只是为了说明使用）中共有10条数据，通过表中数据，说明Group By的优化过程，表中的10条数据分别为：

张三,10　张三,20　张三,30　张三,10　张三,20
张三,30　张三,20　张三,30　李四,30　李四,40

常用参数

是否在Map端进行聚合，默认为True，具体操作如下：

hive.map.aggr=true

在Map端进行聚合操作的条目数目，具体操作如下：

hive.groupby.mapaggr.checkinterval=100000

有数据倾斜的时候进行负载均衡（默认是false），具体操作如下：

hive.groupby.skewindata=true

8.2.3 任务实施

2) hive 分组的负载均衡策略是

- 生成两个 MR Job
- 第一个 MR Job 中的 Map 端会随机将数据分发给 Reduce 端，从而降低数据倾斜的发生
- 第二个MR Job 在根据预处理的数据结果按照Group By Key 再次分发数据完成分组

这样，相当于进行了一次彻彻底底的 shuffle 来消除数据倾斜。

3) 案例分析过程

如果，没有开启负载均衡，那么有2 个Reduce 端，随机分发数据到2 个Reduce端，如图 8-12 所示，数据发生明显的数据倾斜，具体操作如下：

8.2.3 任务实施

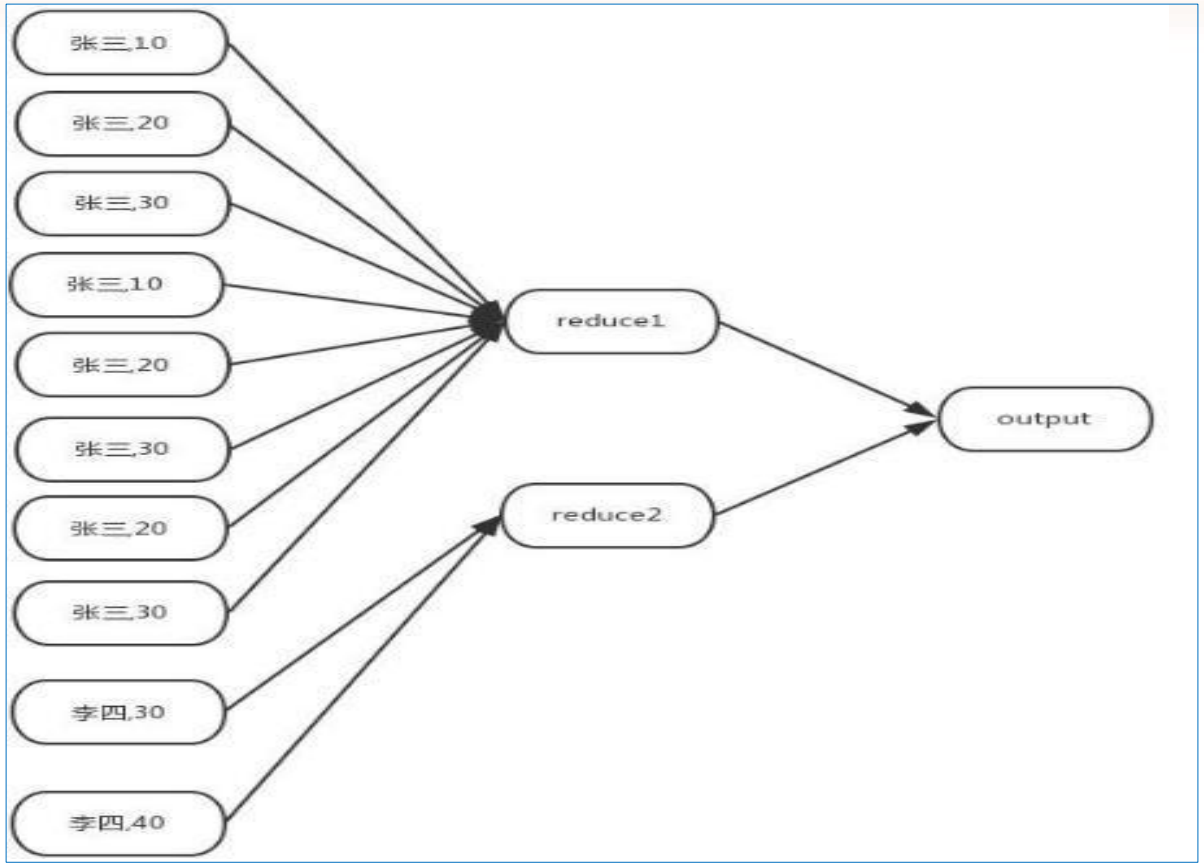

图 8-12 未开启负载均衡

如果，已经开启负载均衡，那么有2 个Reduce 端，随机分发数据到2 个Reduce端之后，形成结果，在预定结果之后，再次分发数据，不同的是，分发数据之前，使用 Group By Key 对数据进行干预，可以有效的降低数据倾斜的概率，如图 8-13所示，具体操作如下：

8.2.3 任务实施

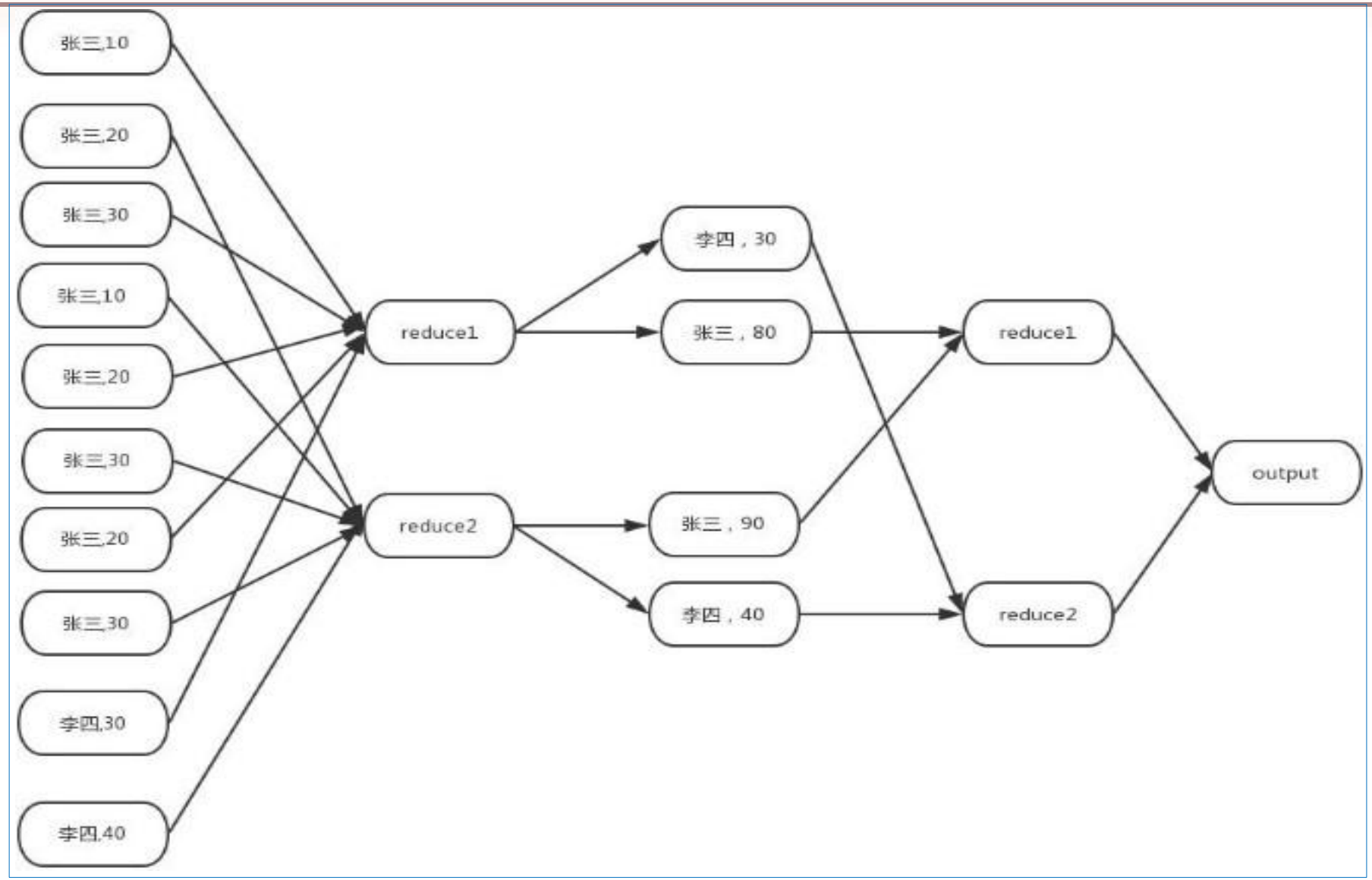

图 8-13 已开启负载均衡

二、 Join 优化

join 相关的优化主要分为 mapjoin 可以解决的优化(即大表 join 小表)和 mapjoin 无法解决的优化(即大表 join 大表)。大表 join 小表相对容易解决，大表 join 大表相对复杂和难以解决，但也不是不可解决的，只是相对比较麻烦而已。本小节中，重点说明 mapjoin 可以解决的优化(即大表 join 小表)。

例题 7：Join 优化案例。**操作过程：**

step01： 设置参数

设置自动选择 MapJoin 的参数，默认为 true，具体操作命令如下： hive> SET hive.auto.convert.join=true;

设置大表和小表的阈值范围，默认 25M 以下是小表，具体操作命令如下：
hive> SET hive.mapjoin.smalltable.filesize=25000000; step02： 创建所需要的表

8.2.3 任务实施

创建数据库 unit8db 中的大表 bigtable，首先，查询表 bigtable 是否创建成功，如图 8-14 所示。具体操作命令如下：

```
hive>          show tables;
```

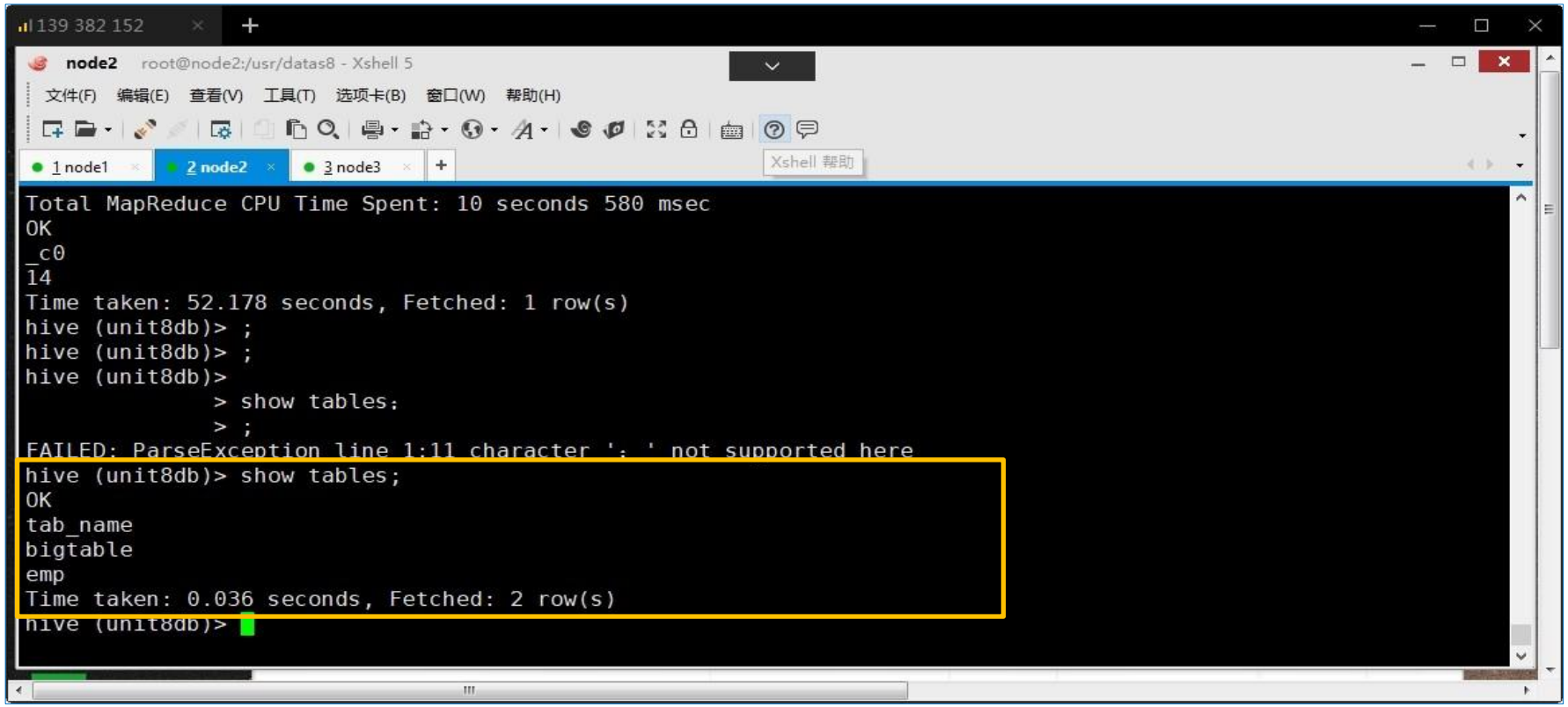

图 8-14 查询数据库的表bigtable 是否已经存在

8.2.3 任务实施

创建数据库unit8db中的小表smalltable，具体操作命令如下：

```
hive> CREATE TABLE smalltable
( id bigint, t bigint, uid string, keyword string, url_rank int,
   click_num int, click_url string)
ROW FORMAT DELIMITED FIELDS TERMINATED by ',';
```

查询表smaltable是否创建成功，成功后，向表中加载数据，如图8-15所示。具体操作命令如下：

```
hive> LOAD DATA LOCAL inpath '/usr/datas8/ smalltable001.csv '
INTO TABLE smalltable;
```

8.2.3 任务实施

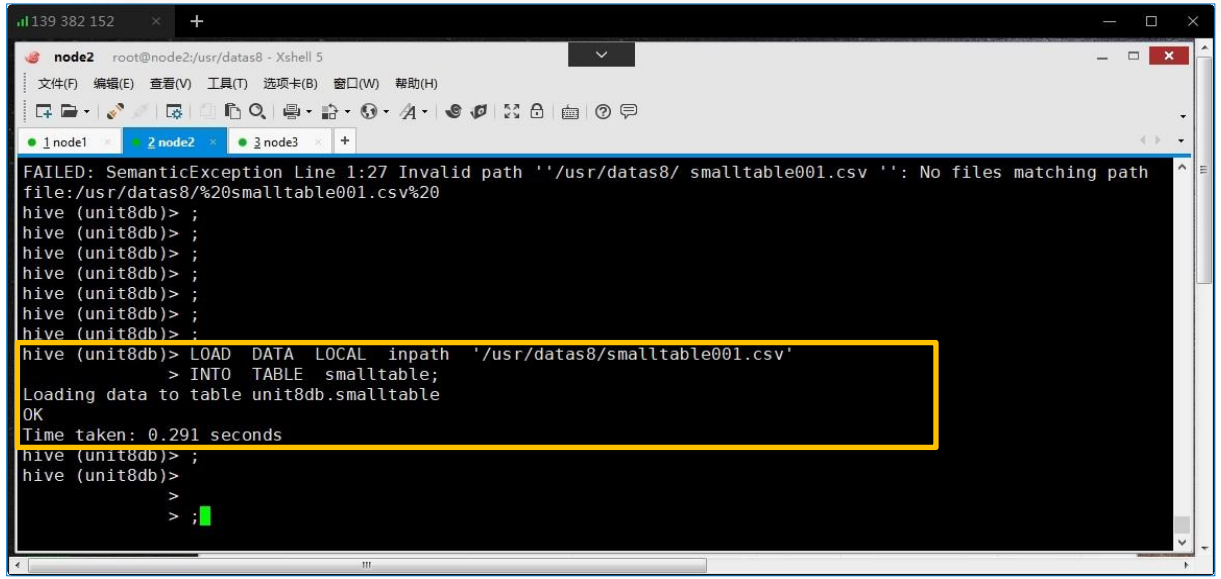

图 8-15

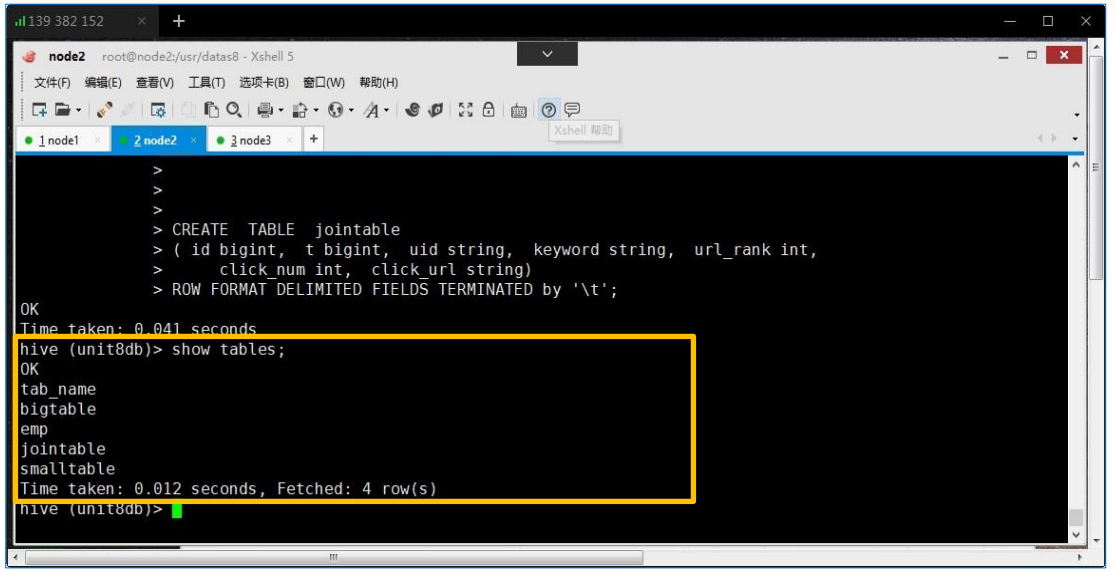

图 8-16

创建数据库 unit8db 中的链接 join 表 jointable，具体操作命令如下： hive> CREATETABLE jointable (id bigint, t bigint, uid string, keyword string, url_rank int, click_num int, click_url string) ROW FORMAT DELIMITED FIELDS TERMINATED by '\t';

查询表 jointable 是否创建成功，如图 8-16 所示。具体操作命令如下： hive> show tables;

8.2.3 任务实施

step03：小表链接大表。查询小表 join 大表，如图 8-17 和图 8-18 所示，具体操作命令如下：

```
hive>  INSERT  OVERWRITE  TABLE  jointable SELECT  b.id,  b.t,  b.uid,
       b.keyword,  b.url_rank, b.click_num,  b.click_url
   FROM  smalltables join  bigtable  b  on  b.id = s.id;
```

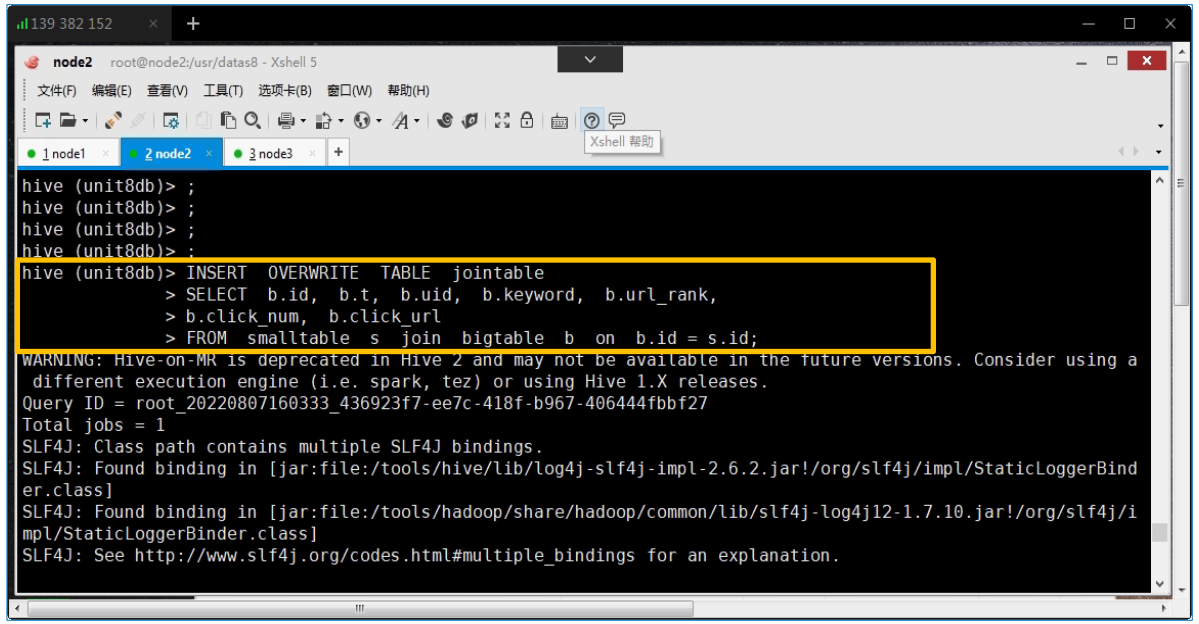

图 8-17 小表join 大表一

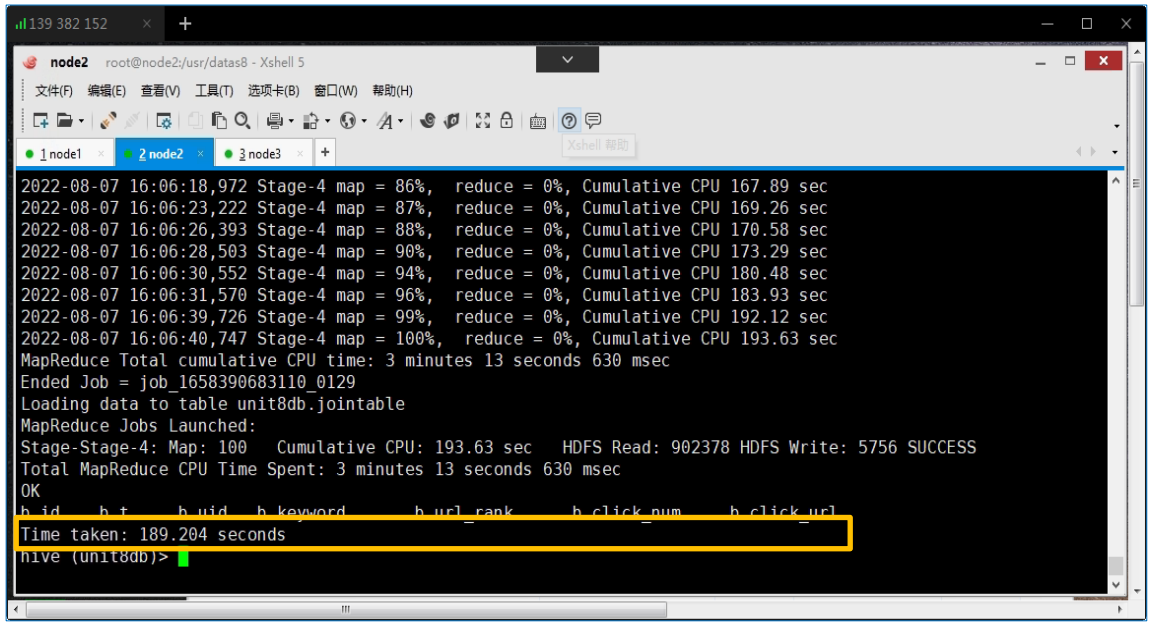

图 8-18 小表join 大表二

8.2.3 任务实施

step04：　　大表链接小表

查询大表 join 小表，如图 8-19 和图 8-20 所示，具体操作命令如下：hive> INSERT OVERWRITE TABLE jointable
SELECT b.id, b.t, b.uid, b.keyword, b.url_rank,
b.click_num, b.click_url
FROM bigtable b join smalltable s on s.id = b.id;

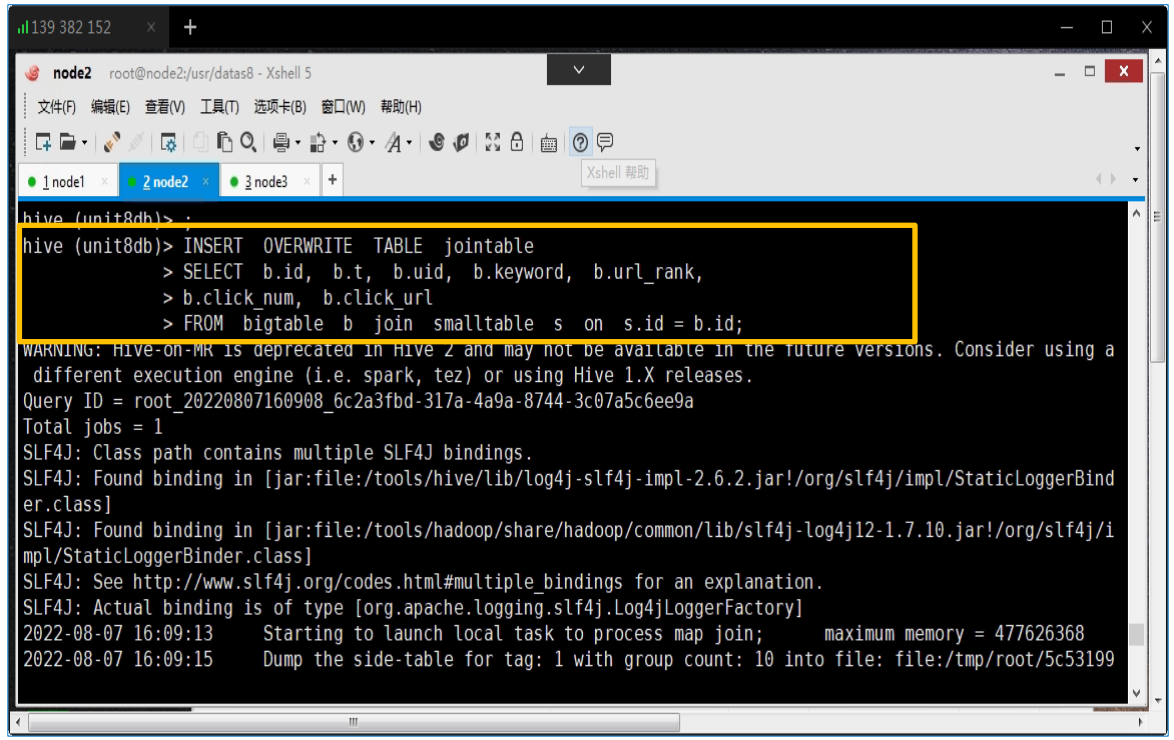

图 8-19 大表join 小表一

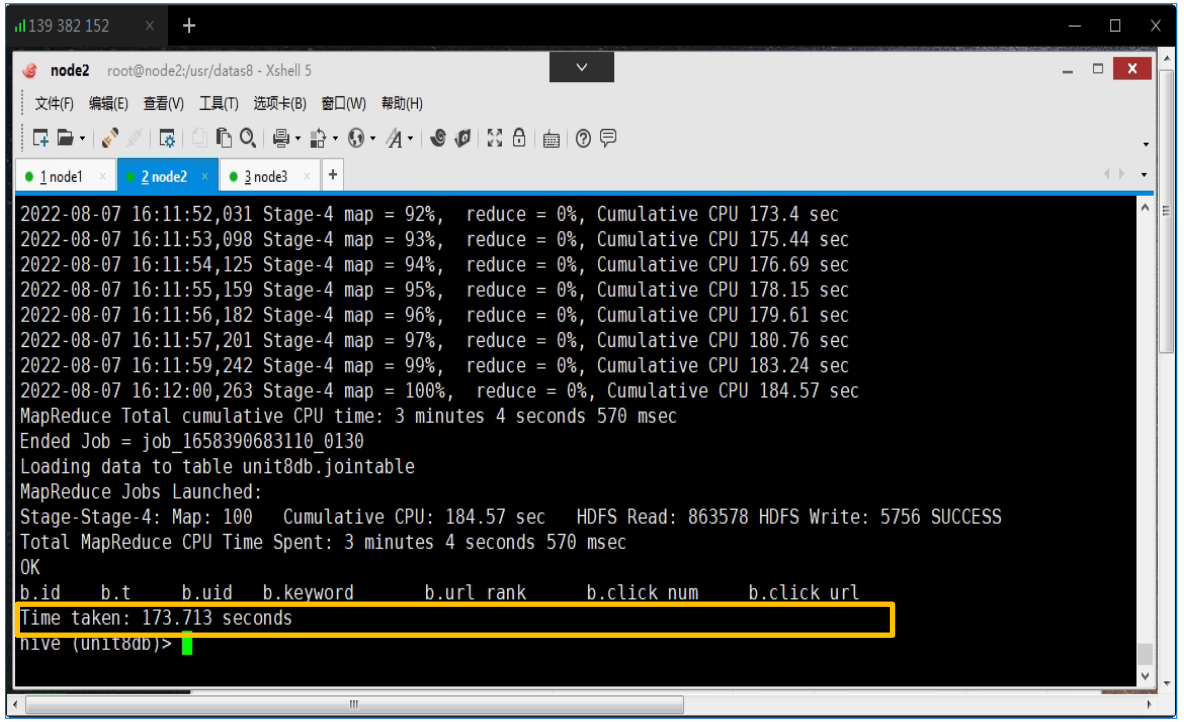

图 8-20 大表join 小表二

8.2.3 任务实施

三、Count（Distinct）优化

数据量小的时候无所谓，数据量大的情况下，由于 Count Distinct 操作需要用一个 Reduce Task 来完成，这一个 Reduce 需要处理的数据量太大，就会导致整

个Job 很难完成，一般Count Distinct 使用先 Group By 再Count 的方式替换，但是需要注意Group By 造成的数据倾斜问题。**例题8：**Count(　　)优化案例。

操作过程：

step01：　　创建所需要的表

创建数据库 unit8db 中的大表 bigtable，首先，查询表 bigtable 是否创建成功，如图 8-11 所示，表 bigtable 已经存在，且其中有 320 条数据。

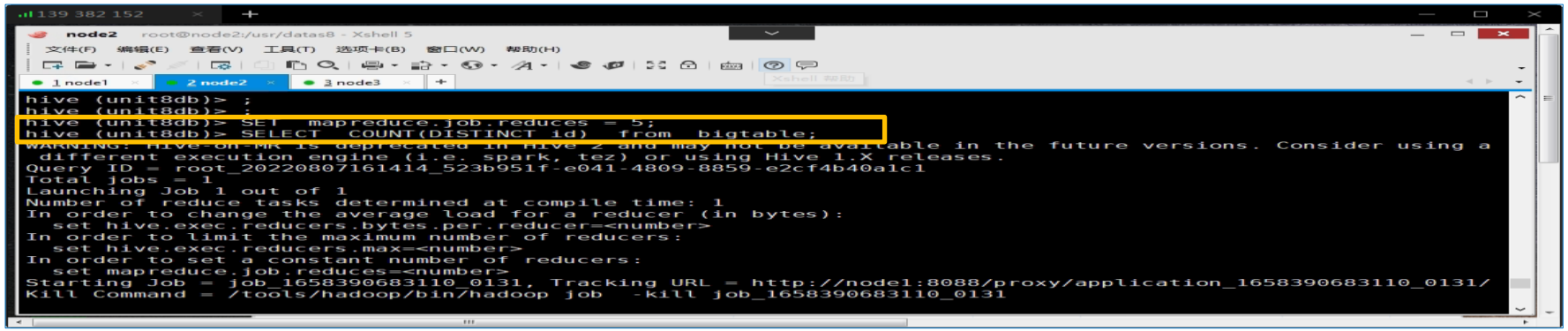

图 8-21 去重查询表bigtable 一

8.2.3 任务实施

step02：　　设置 reduce 的个数为 5

设置 reduce 的个数为 5，具体操作命令如下： hive> SET mapreduce.job.reduces = 5; step03： 执行去重查询

执行去重查询，即使用 Count Distinct 操作进行查询，如图 8-21 和图 8-22 所示，具体操作命令如下：

```
hive>　SELECT　COUNT(DISTINCT id)　from　bigtable;
```

8.2.3 任务实施

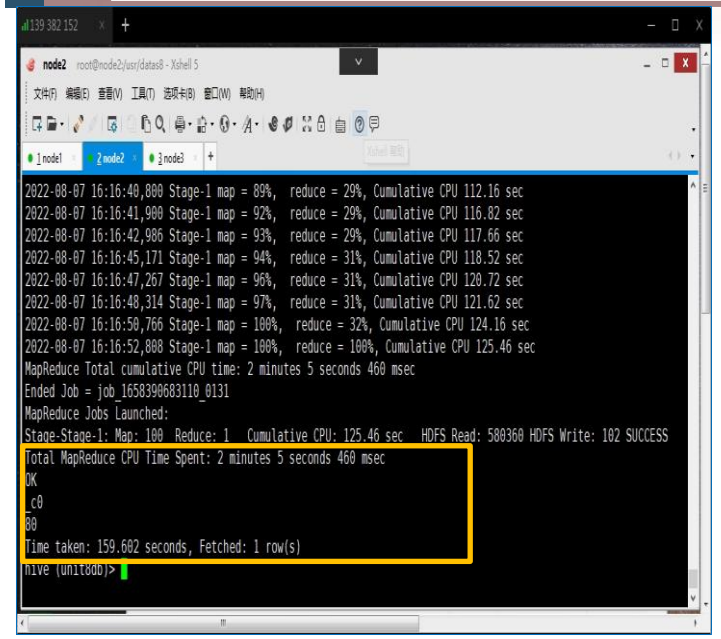

图 8-22

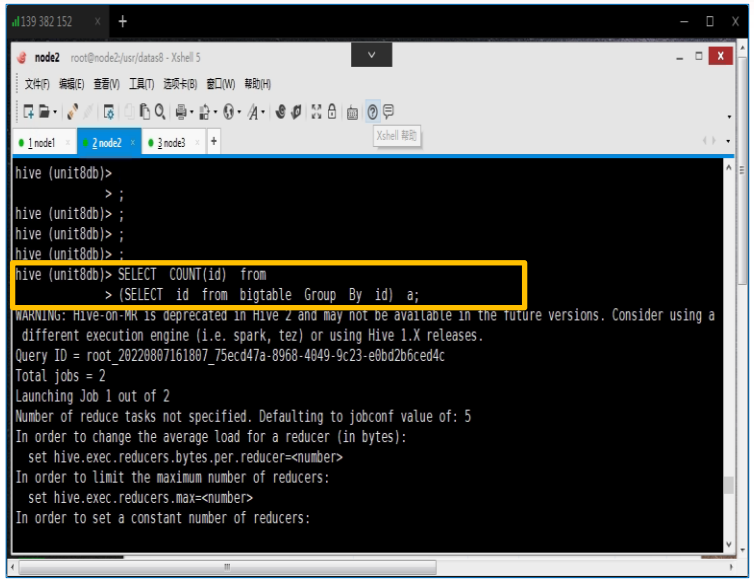

图 8-23

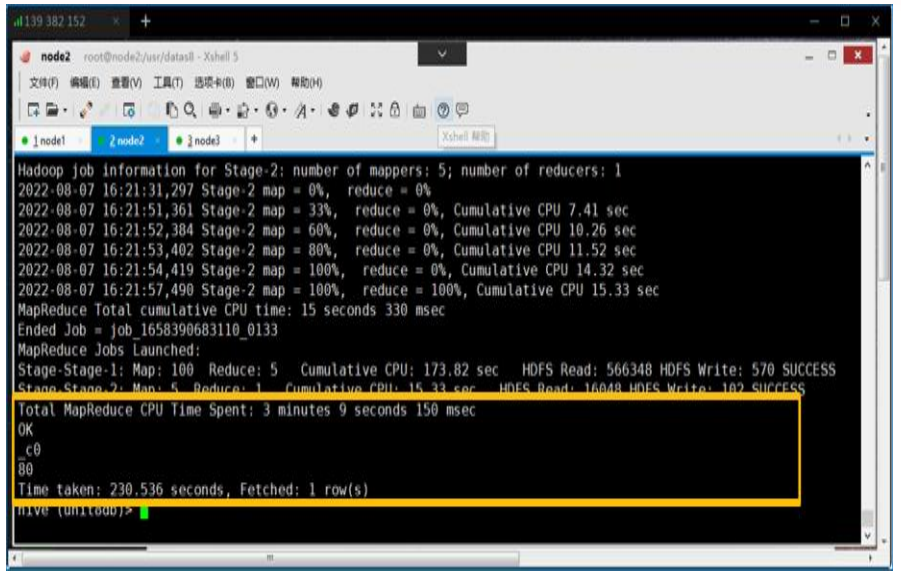

图 8-24

step04： 执行先 Group By 再 Count 的方式查询

执行先 Group By 再 Count 的方式查询，如图 8-23 和图 8-24 所示，具体操作命令如下：

hive>SELECT COUNT(id) from (SELECT id from bigtable Group By id) a;

8.2.3 任务实施

四、行/列过滤优化

1. 列处理

在 SELECT 中，只处理需要的列，如果有分区，尽量使用分区过滤，少用 SELECT *。

2. 行处理

在分区剪裁中，当使用外关联时，如果将副表的过滤条件写在 Where 后面，那么就会先全表关联，之后再过滤。

8.2.3 任务实施

例题9：行/列过滤优化案例。

操作过程：

创建所需要的表

创建数据库unit8db中的大表bigtable，首先，查询表bigtable是否创建成功，如图8-11所示，表bigtable已经存在，且其中有数据。

执行先关联表，再子查询的方式查询

执行先关联表再子查询的方式查询，如图8-25和图8-26所示，具体操作命令如下：

```
hive>  SELECT  o.id
FROM  bigtable  b  join  bigtable  o  on  o.id = b.id
    WHERE  o.id <= 10;
```

8.2.3 任务实施

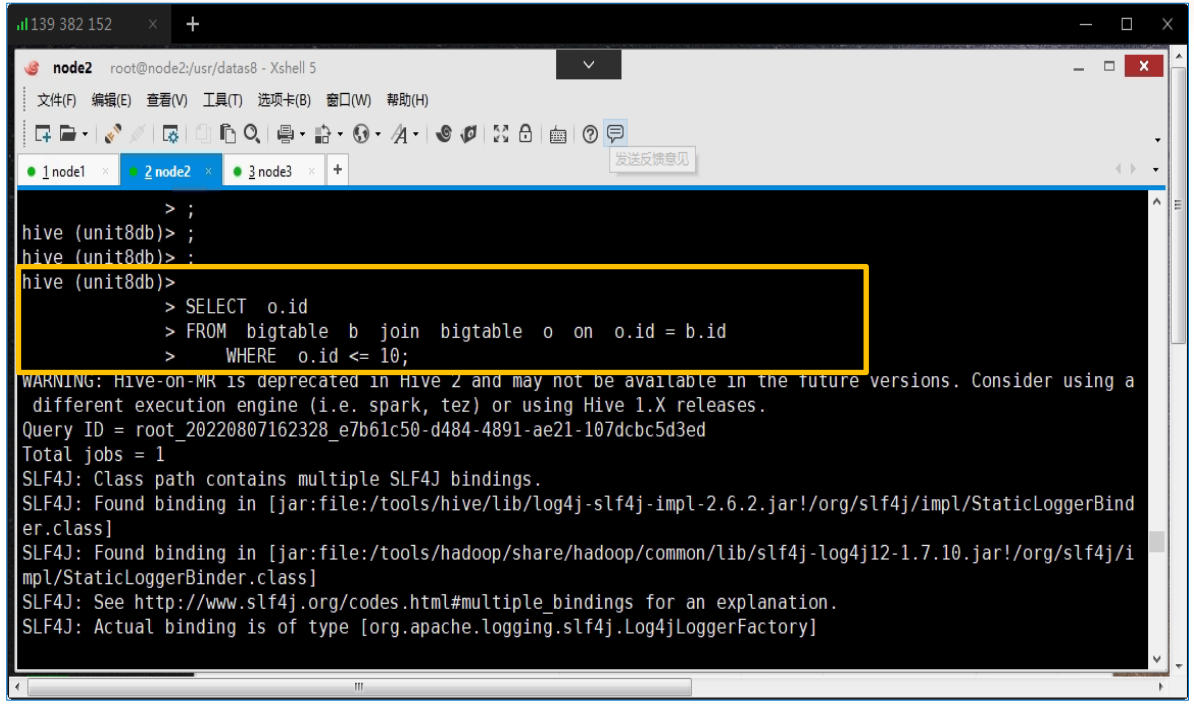

图 8-25 先关联表再子查询的方式查询一

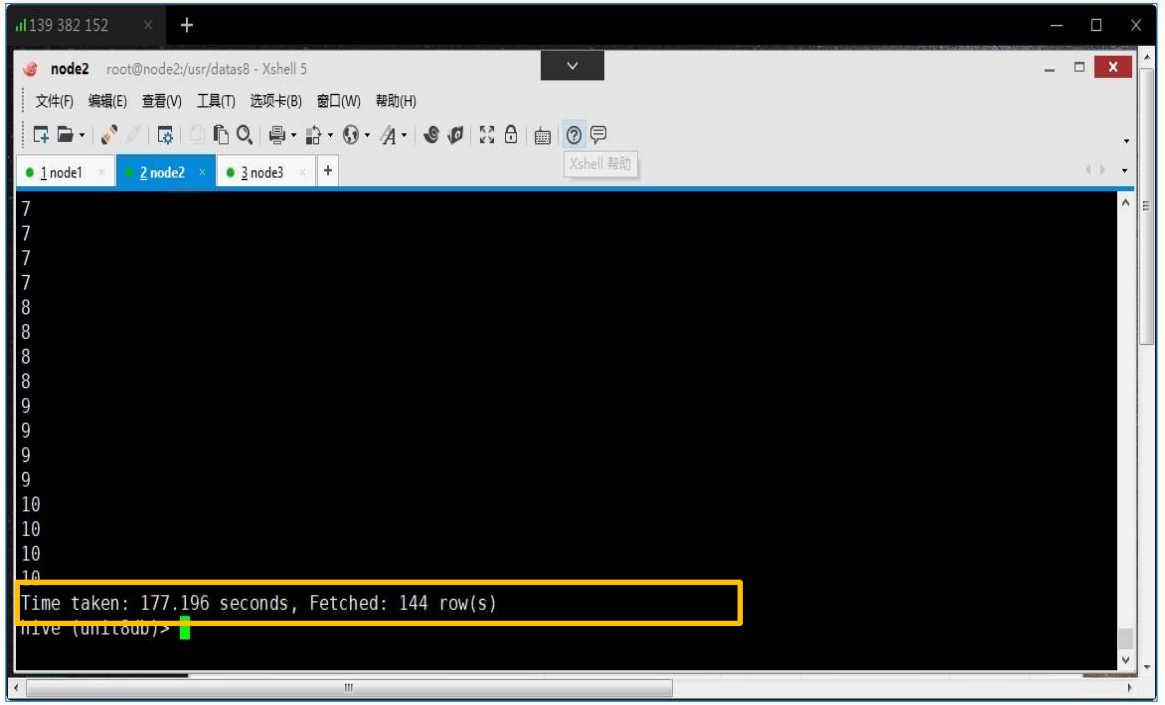

图 8-26 先关联表再子查询的方式查询二

step03： 执行先子查询表，再关联表的方式查询

执行先关联表再查询的方式查询，如图 8-27 和图 8-28 所示，具体操作命令如下：

```
hive> SELECT b.id from bigtable b join
    (SELECT id from bigtable where id <= 10) o on b.id = o.id;
```

8.2.3 任务实施

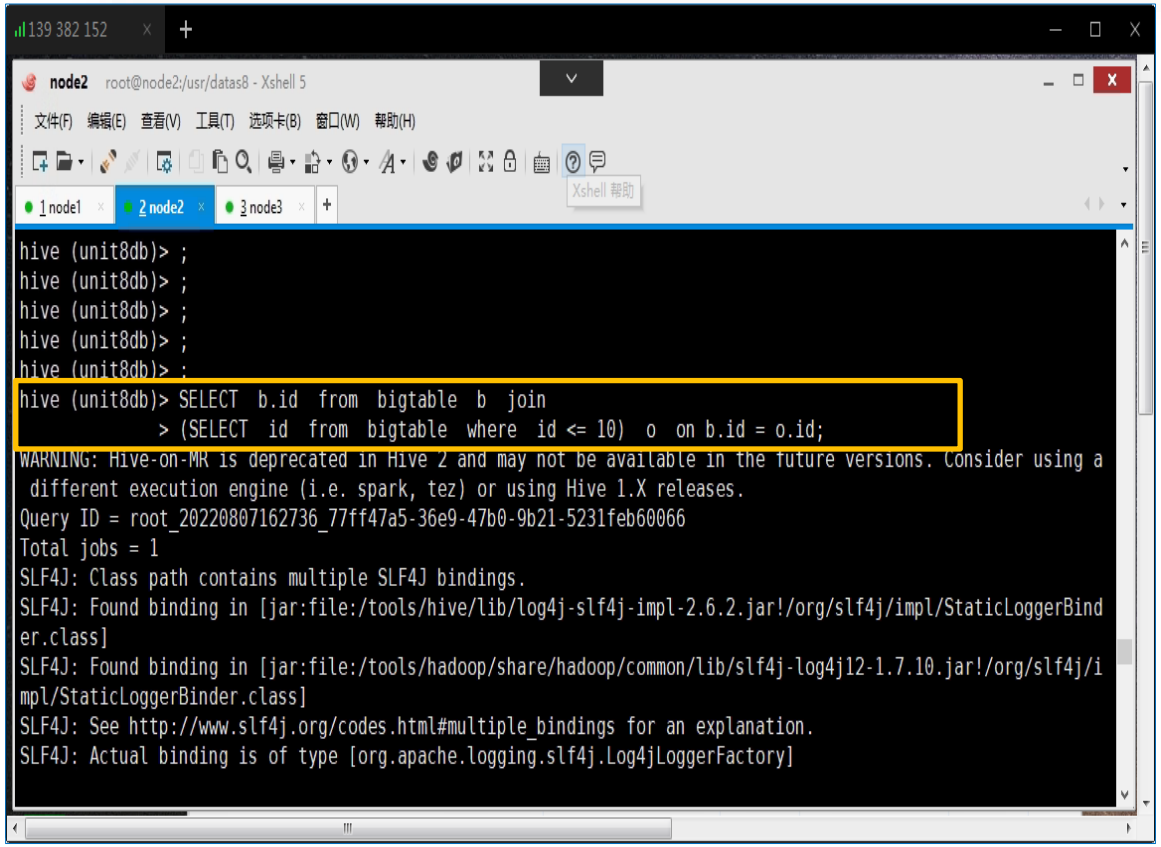

图 8-27 先子查询再关联表的方式查询一

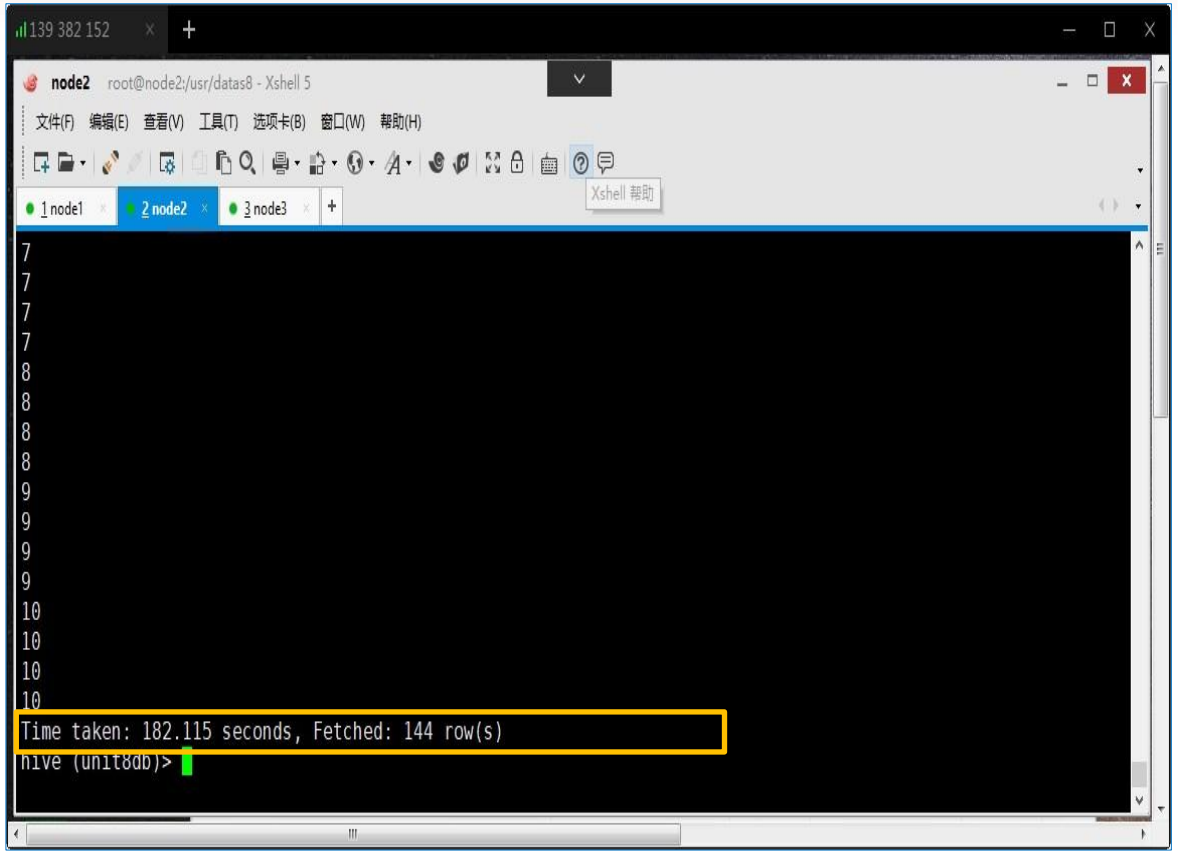

图 8-28 先子查询再关联表的方式查询二

任务 8-3

本章小结

8.3.1 任务描述

我们在本章中首先学习了Hive生态系统中参数优化的五种方式，其中，重点介绍了配置MapReduce压缩、配置Map个数、配置Reduce个数、合并小文件等方式；然后，本章还详细介绍了Hive生态系统中HQL优化的四种方式，其中，Group By的优化、行/列过滤优化等方式进行了重点说明。在本章中，不仅理解和掌握Hive生态系统中参数优化的几种方式，同时理解和掌握Hive生态系统中HQL优化的几种方式，是日常生活中应用Hive高效解决实际问题的重要基础。







任务 8-4

本章习题

8.4.1 任务描述

1、Hive生态系统中参数优化有哪几种方式?

2、Hive生态系统中HQL优化有哪几种方式?

3、请结合实际，从本专业的角度，说明在日常生活中应用Hive高效解决实际问题的时候，需要注意哪些方面?







感谢您的观看

