第三次实验报告

姓名	学号	邮箱	院系
姜帅	221220115	jsissosix1221@gmail.com	计算机学院

任务一: HBase 安装及命令行操作

0 安装Zookeeper并完成配置

下载Zookeeper： 首先，从 Apache 官方网站下载 ZooKeeper 3.4.12 的 tarball 文件。下载完成后解压

```
wget https://archive.apache.org/dist/zookeeper/zookeeper-3.4.12/zookeeper-3.4.12.tar.gz
tar -xzf zookeeper-3.4.12.tar.gz
```

配置Zookeeper： 进入安装目录并创建一个配置文件

```
cd /home/js/Zookeeper/zookeeper-3.4.12/conf
cp zoo_sample.cfg zoo.cfg
```

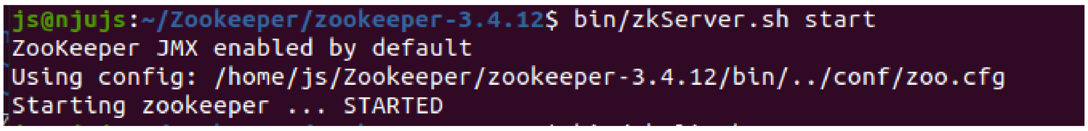
编辑`zoo.cfg`文件，将dataDir修改为自己创建的路径，例如：

```
# ...
dataDir=/home/js/Zookeeper/zookeeper-3.4.12/zkdata
# ...
```

启动Zookeeper： 进入 ZooKeeper 的 `bin` 目录并启动服务

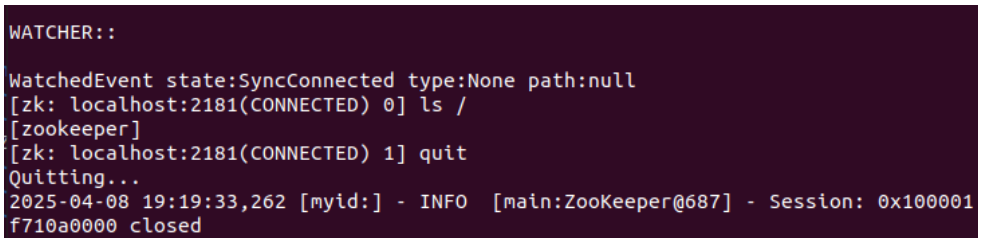
```
cd /home/js/Zookeeper/zookeeper-3.4.12/bin
./zkServer.sh start
```

可以看到以下输出：



```
js@njujs:~/Zookeeper/zookeeper-3.4.12$ bin/zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /home/js/Zookeeper/zookeeper-3.4.12/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
```

使用 ZooKeeper 客户端连接测试： 输入`./zkCli.sh`,如下图所示可以进入Zookeeper的命令行界面, 运行命令（`ls /`）, 安装成功.



```
WATCHER::

WatchedEvent state:SyncConnected type:None path:null
[zk: localhost:2181(CONNECTED) 0] ls /
[zookeeper]
[zk: localhost:2181(CONNECTED) 1] quit
Quitting...
2025-04-08 19:19:33,262 [myid:] - INFO  [main:ZooKeeper@687] - Session: 0x100001
f710a0000 closed
```

1 安装HBase并完成配置

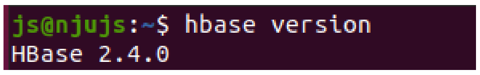
下载HBase：从Apache官网下载HBase-2.4.0的压缩包，解压下载的文件

```
wget https://archive.apache.org/dist/hbase/2.4.0/hbase-2.4.0-bin.tar.gz
tar -xzf hbase-2.4.0-bin.tar.gz
```

编辑`~/.bashrc`文件，在文件末尾添加以下内容

```
export HBASE_HOME=/home/js/HBase/hbase-2.4.0
export PATH=$PATH:$HBASE_HOME/bin
```

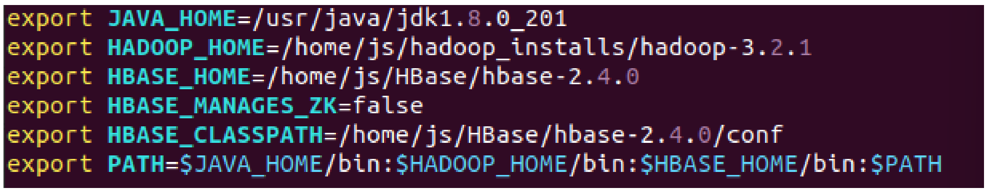
使更改生效：`source ~/.bashrc`，通过`hbase version`验证安装是否成功，如图：



```
js@njujs:~$ hbase version
HBase 2.4.0
```

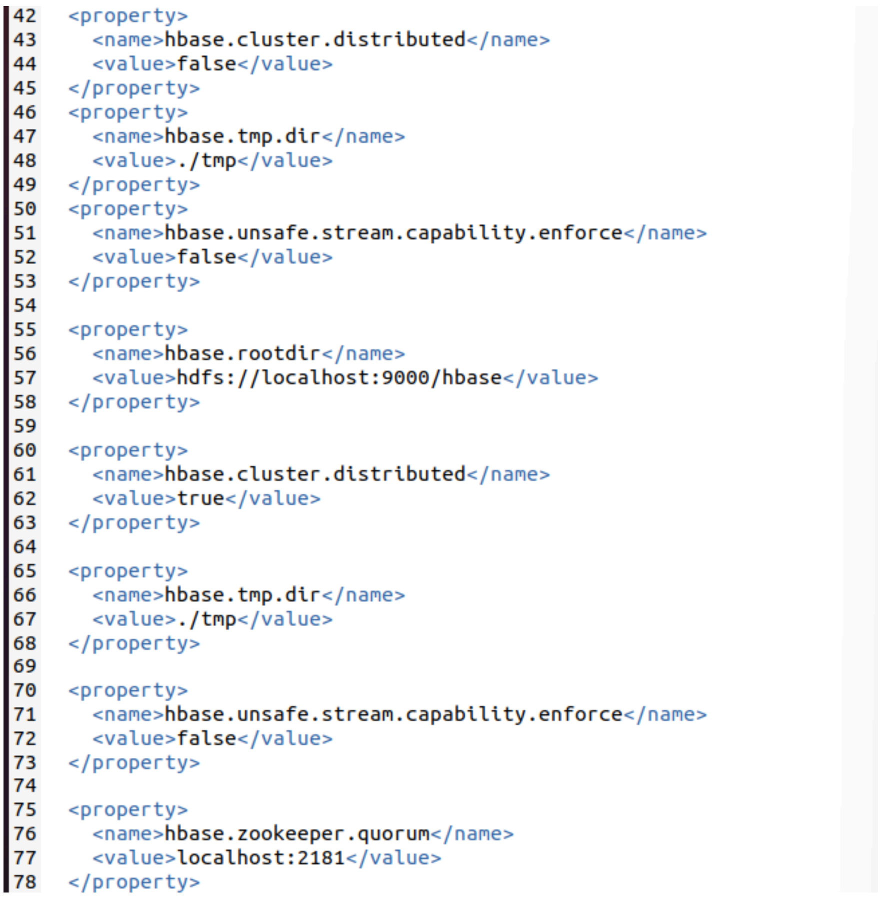
配置HBase：

1. 编辑`conf/hbase-env.sh`,在末尾添加如下内容：



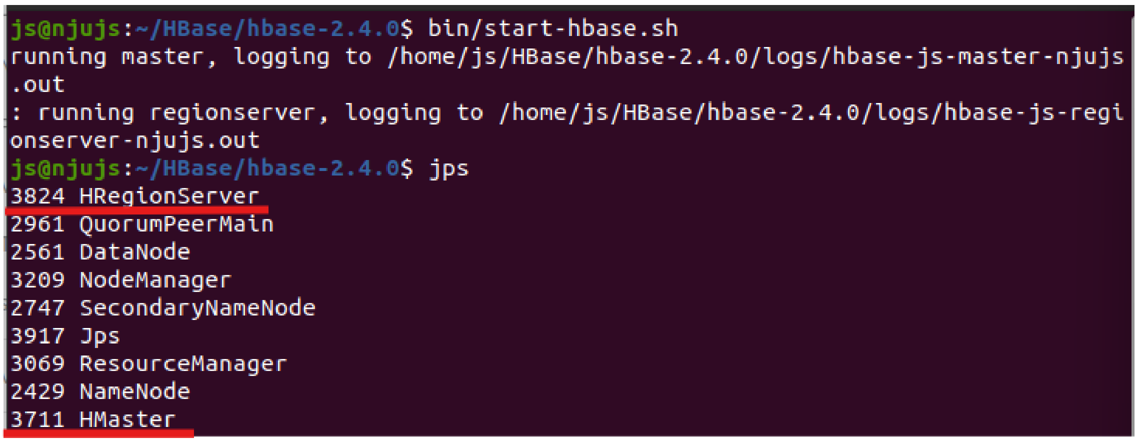
```
export JAVA_HOME=/usr/java/jdk1.8.0_201
export HADOOP_HOME=/home/js/hadoop_installs/hadoop-3.2.1
export HBASE_HOME=/home/js/HBase/hbase-2.4.0
export HBASE_MANAGES_ZK=false
export HBASE_CLASSPATH=/home/js/HBase/hbase-2.4.0/conf
export PATH=$JAVA_HOME/bin:$HADOOP_HOME/bin:$HBASE_HOME/bin:$PATH
```

2. 编辑hbase-site.xml:



```
  <property>
    <name>hbase.cluster.distributed</name>
    <value>false</value>
  </property>
  <property>
    <name>hbase.tmp.dir</name>
    <value>./tmp</value>
  </property>
  <property>
    <name>hbase.unsafe.stream.capability.enforce</name>
    <value>false</value>
  </property>

  <property>
    <name>hbase.rootdir</name>
    <value>hdfs://localhost:9000/hbase</value>
  </property>

  <property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
  </property>

  <property>
    <name>hbase.tmp.dir</name>
    <value>./tmp</value>
  </property>

  <property>
    <name>hbase.unsafe.stream.capability.enforce</name>
    <value>false</value>
  </property>

  <property>
    <name>hbase.zookeeper.quorum</name>
    <value>localhost:2181</value>
  </property>
```

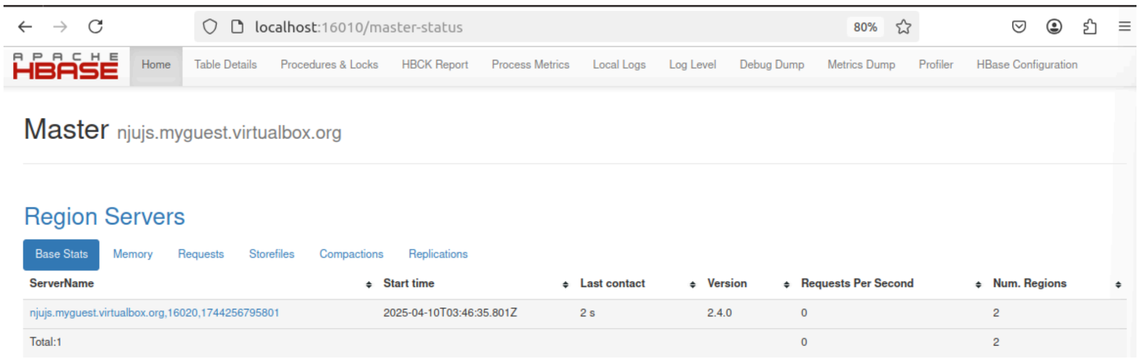
3. 验证HBase是否运行：启动HBase，输入jps检查进程，结果如下（需要先启动HDFS和ZooKeeper）



```
js@njujs:~/HBase/hbase-2.4.0$ bin/start-hbase.sh
running master, logging to /home/js/HBase/hbase-2.4.0/logs/hbase-js-master-njujs
.out
: running regionserver, logging to /home/js/HBase/hbase-2.4.0/logs/hbase-js-regi
onserver-njujs.out
js@njujs:~/HBase/hbase-2.4.0$ jps
3824 HRegionServer
2961 QuorumPeerMain
2561 DataNode
3209 NodeManager
2747 SecondaryNameNode
3917 Jps
3069 ResourceManager
2429 NameNode
3711 HMaster
```

可以看到HMaster和HRegionServer, HBase服务正常启动

在浏览器中访问本地的16010端口，成功访问：



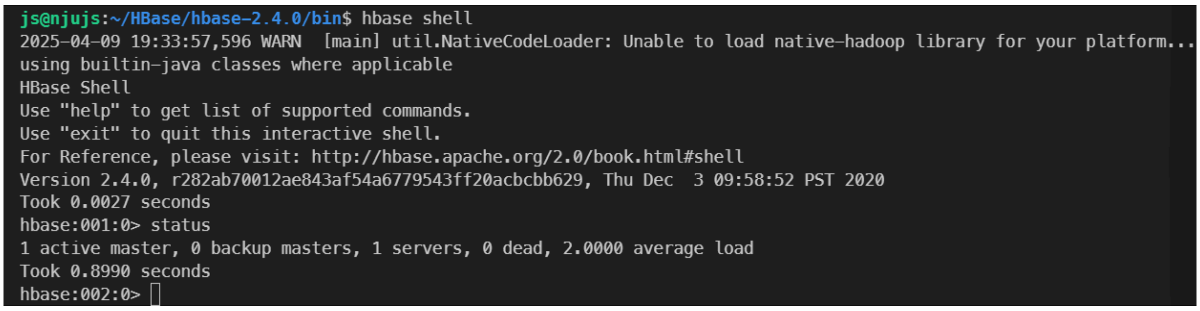
Master njujs.myguest.virtualbox.org

Region Servers

ServerName	Start time	Last contact	Version	Requests Per Second	Num. Regions
njujs.myguest.virtualbox.org,16020,1744256795801	2025-04-10T03:46:35.801Z	2 s	2.4.0	0	2
Total:1				0	2

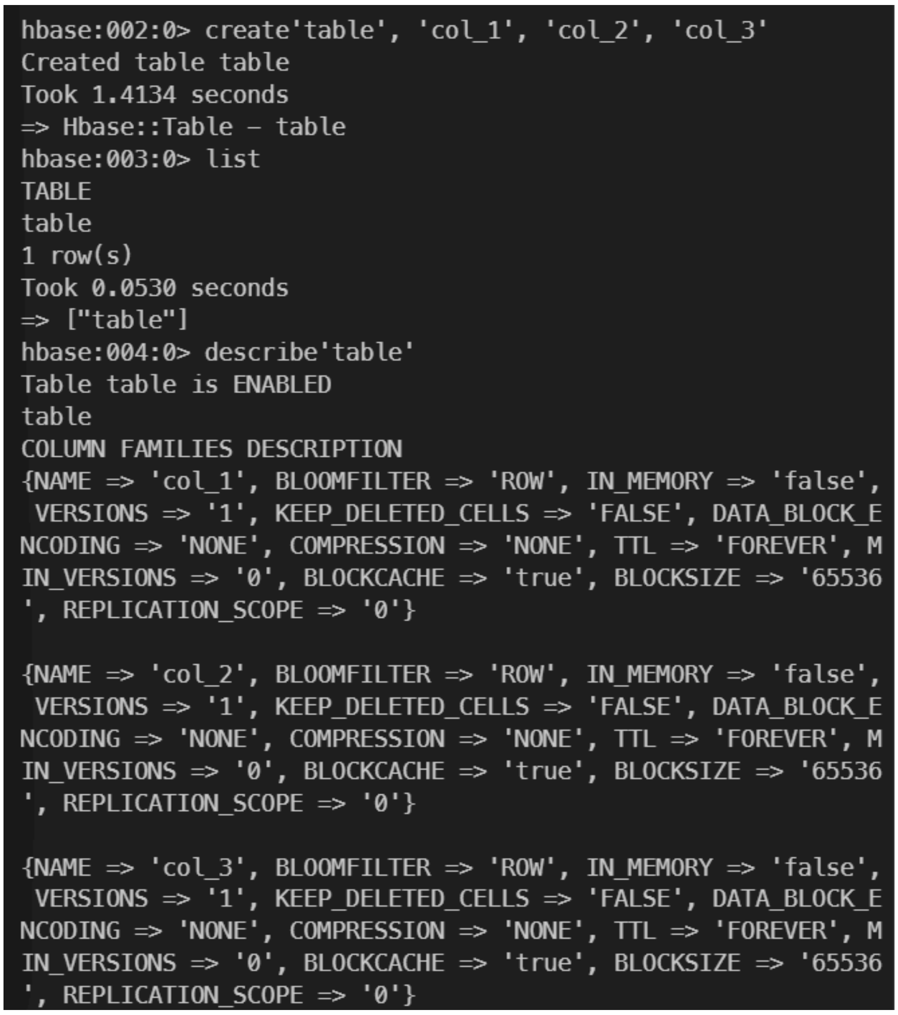
2 进入HBase shell命令行，熟悉HBase基础操作

启动HBase后输入hbase shell进入命令行并输入`status`查看状态：



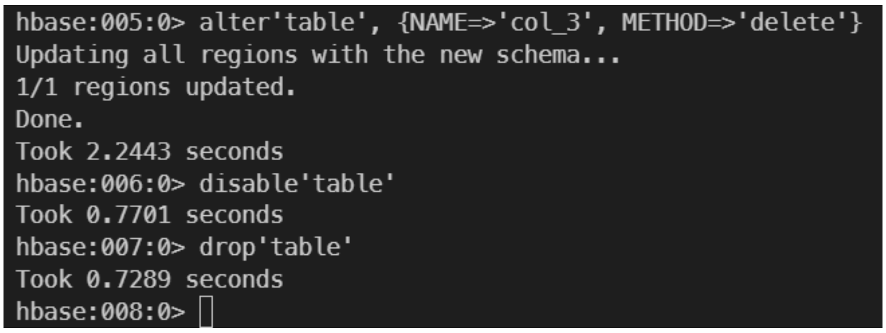
```
js@njujs:~/HBase/hbase-2.4.0/bin$ hbase shell
2025-04-09 19:33:57,596 WARN  [main] util.NativeCodeLoader: Unable to load native-hadoop library for your platform...
using builtin-java classes where applicable
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
For Reference, please visit: http://hbase.apache.org/2.0/book.html#shell
Version 2.4.0, r282ab70012ae843af54a6779543ff20acbcbb629, Thu Dec  3 09:58:52 PST 2020
Took 0.0027 seconds
hbase:001:0> status
1 active master, 0 backup masters, 1 servers, 0 dead, 2.0000 average load
Took 0.8990 seconds
hbase:002:0> 
```

- 创建表，列出所有表，获取表的描述：



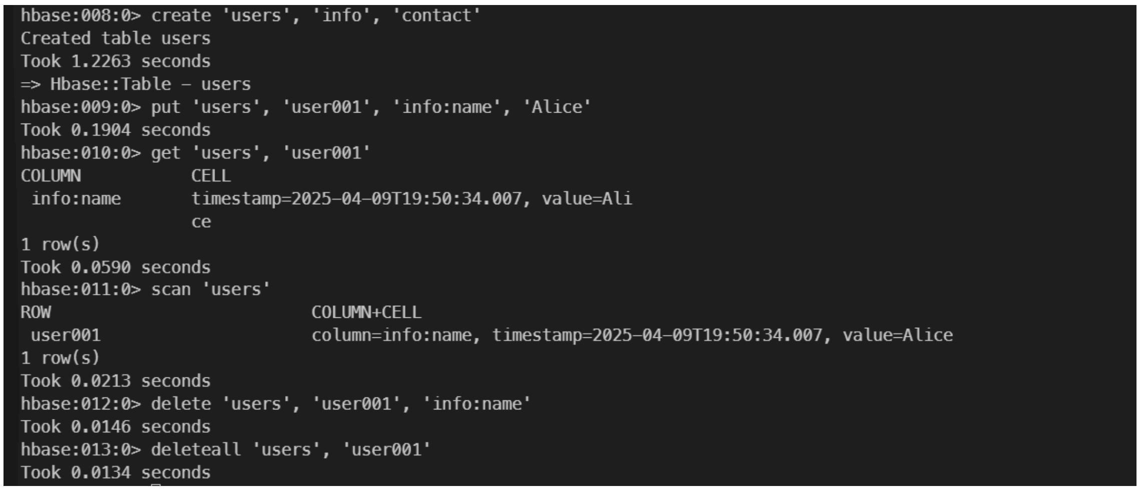
```
hbase:002:0> create'table', 'col_1', 'col_2', 'col_3'
Created table table
Took 1.4134 seconds
=> Hbase::Table - table
hbase:003:0> list
TABLE
table
1 row(s)
Took 0.0530 seconds
=> ["table"]
hbase:004:0> describe'table'
Table table is ENABLED
table
COLUMN FAMILIES DESCRIPTION
{NAME => 'col_1', BLOOMFILTER => 'ROW', IN_MEMORY => 'false',
 VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_E
NCODING => 'NONE', COMPRESSION => 'NONE', TTL => 'FOREVER', M
IN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536
', REPLICATION_SCOPE => '0'}

{NAME => 'col_2', BLOOMFILTER => 'ROW', IN_MEMORY => 'false',
 VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_E
NCODING => 'NONE', COMPRESSION => 'NONE', TTL => 'FOREVER', M
IN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536
', REPLICATION_SCOPE => '0'}

{NAME => 'col_3', BLOOMFILTER => 'ROW', IN_MEMORY => 'false',
 VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_E
NCODING => 'NONE', COMPRESSION => 'NONE', TTL => 'FOREVER', M
IN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536
', REPLICATION_SCOPE => '0'}
```

- 删除列族，删除表：



```
hbase:005:0> alter'table', {NAME=>'col_3', METHOD=>'delete'}
Updating all regions with the new schema...
1/1 regions updated.
Done.
Took 2.2443 seconds
hbase:006:0> disable'table'
Took 0.7701 seconds
hbase:007:0> drop'table'
Took 0.7289 seconds
hbase:008:0> 
```

- 插入、更新、扫描、删除数据：



```
hbase:008:0> create 'users', 'info', 'contact'
Created table users
Took 1.2263 seconds
=> Hbase::Table - users
hbase:009:0> put 'users', 'user001', 'info:name', 'Alice'
Took 0.1904 seconds
hbase:010:0> get 'users', 'user001'
COLUMN               CELL
 info:name           timestamp=2025-04-09T19:50:34.007, value=Ali
                     ce
1 row(s)
Took 0.0590 seconds
hbase:011:0> scan 'users'
ROW                               COLUMN+CELL
 user001                          column=info:name, timestamp=2025-04-09T19:50:34.007, value=Alice
1 row(s)
Took 0.0213 seconds
hbase:012:0> delete 'users', 'user001', 'info:name'
Took 0.0146 seconds
hbase:013:0> deleteall 'users', 'user001'
Took 0.0134 seconds
```

- 此外，可以使用 `help` 命令查看所有命令或某个具体命令的帮助

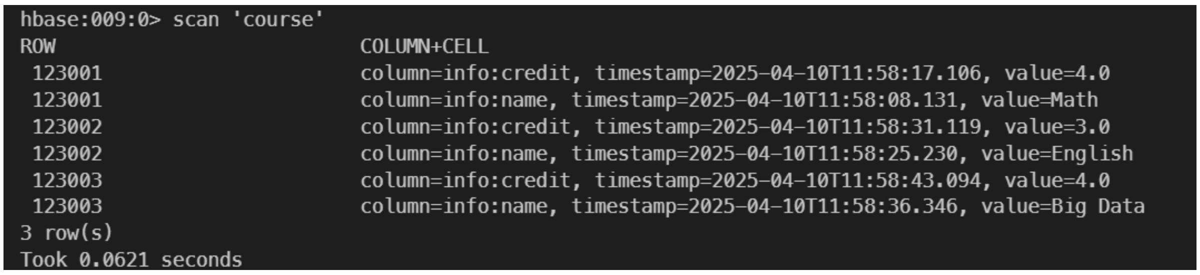
3 Hbase与关系型数据库

设计并创建表：

对于课程表（course），行键采用C_No, 列族为info，列限定符为name和credit。通过create创建表然后使用put插入数据，插入后通过scan扫描，具体操作

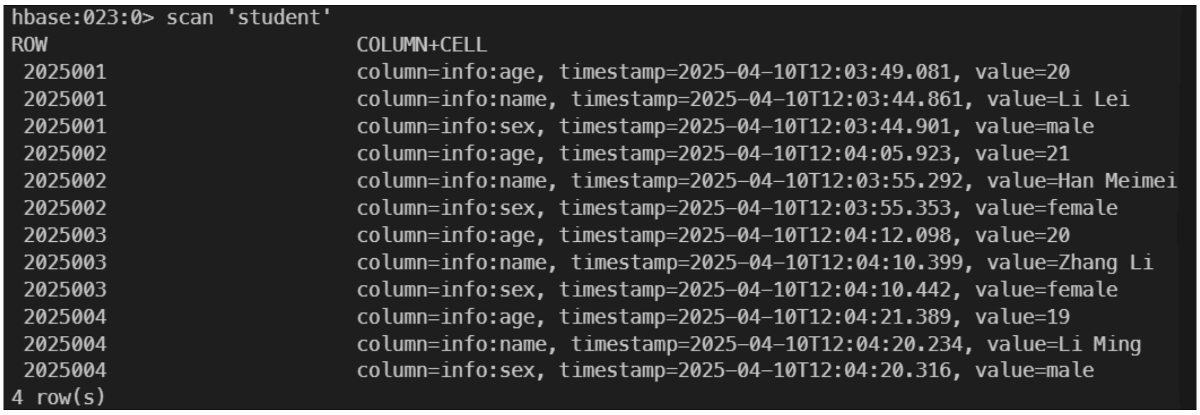
```
create 'course', 'info'

put 'course', '123001', 'info:name', 'Math'
put 'course', '123001', 'info:credit', '4.0'

put 'course', '123002', 'info:name', 'English'
put 'course', '123002', 'info:credit', '3.0'

put 'course', '123003', 'info:name', 'Big Data'
put 'course', '123003', 'info:credit', '4.0'
```

可以看到该表已成功创建：



```
hbase:009:0> scan 'course'
ROW                          COLUMN+CELL
 123001                      column=info:credit, timestamp=2025-04-10T11:58:17.106, value=4.0
 123001                      column=info:name, timestamp=2025-04-10T11:58:08.131, value=Math
 123002                      column=info:credit, timestamp=2025-04-10T11:58:31.119, value=3.0
 123002                      column=info:name, timestamp=2025-04-10T11:58:25.230, value=English
 123003                      column=info:credit, timestamp=2025-04-10T11:58:43.094, value=4.0
 123003                      column=info:name, timestamp=2025-04-10T11:58:36.346, value=Big Data
3 row(s)
Took 0.0621 seconds
```

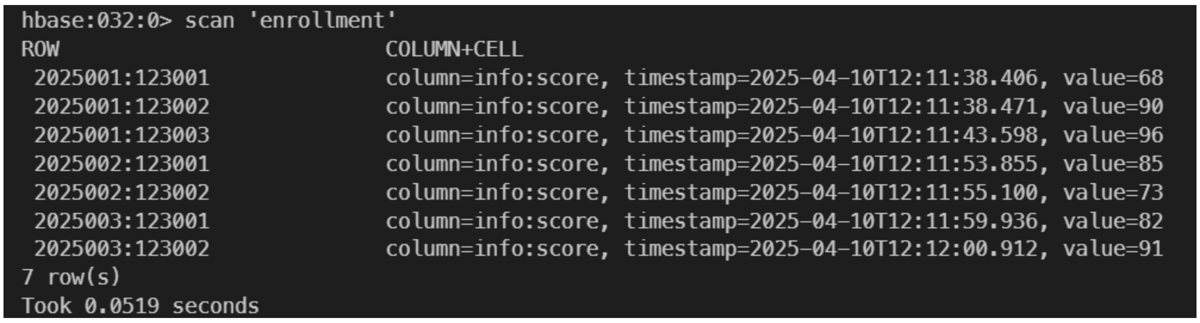
对于学生表（student），行键为S_No, 列族info下的具体字段为name, sex, age, 创建方式同上，结果如下：



```
hbase:023:0> scan 'student'
ROW                          COLUMN+CELL
 2025001                     column=info:age, timestamp=2025-04-10T12:03:49.081, value=20
 2025001                     column=info:name, timestamp=2025-04-10T12:03:44.861, value=Li Lei
 2025001                     column=info:sex, timestamp=2025-04-10T12:03:44.901, value=male
 2025002                     column=info:age, timestamp=2025-04-10T12:04:05.923, value=21
 2025002                     column=info:name, timestamp=2025-04-10T12:03:55.292, value=Han Meimei
 2025002                     column=info:sex, timestamp=2025-04-10T12:03:55.353, value=female
 2025003                     column=info:age, timestamp=2025-04-10T12:04:12.098, value=20
 2025003                     column=info:name, timestamp=2025-04-10T12:04:10.399, value=Zhang Li
 2025003                     column=info:sex, timestamp=2025-04-10T12:04:10.442, value=female
 2025004                     column=info:age, timestamp=2025-04-10T12:04:21.389, value=19
 2025004                     column=info:name, timestamp=2025-04-10T12:04:20.234, value=Li Ming
 2025004                     column=info:sex, timestamp=2025-04-10T12:04:20.316, value=male
4 row(s)
```

对于选课表(enrollment), 行键采用`S_No:C_No`的组合, 列族info, info:score 存储成绩, 对应hbase指令以及结果:

```
create 'enrollment', 'info'

put 'enrollment', '2025001:123001', 'info:score', '68'
put 'enrollment', '2025001:123002', 'info:score', '90'
put 'enrollment', '2025001:123003', 'info:score', '96'
...
```

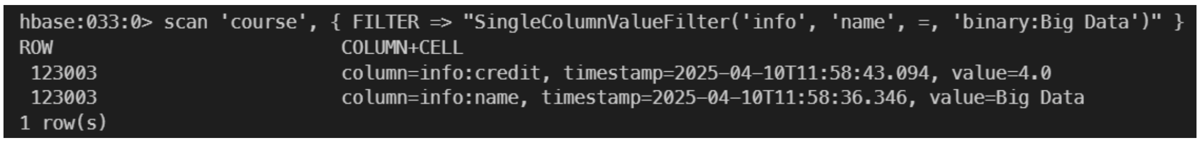


```
hbase:032:0> scan 'enrollment'
ROW                           COLUMN+CELL
 2025001:123001               column=info:score, timestamp=2025-04-10T12:11:38.406, value=68
 2025001:123002               column=info:score, timestamp=2025-04-10T12:11:38.471, value=90
 2025001:123003               column=info:score, timestamp=2025-04-10T12:11:43.598, value=96
 2025002:123001               column=info:score, timestamp=2025-04-10T12:11:53.855, value=85
 2025002:123002               column=info:score, timestamp=2025-04-10T12:11:55.100, value=73
 2025003:123001               column=info:score, timestamp=2025-04-10T12:11:59.936, value=82
 2025003:123002               column=info:score, timestamp=2025-04-10T12:12:00.912, value=91
7 row(s)
Took 0.0519 seconds
```

查询选修Big Data的学生的成绩

首先在course表中通过课程名获取Big Data对应的课程号:

```
scan 'course', { FILTER => "SingleColumnValueFilter('info', 'name', =, 'binary:Big Data')" }
```

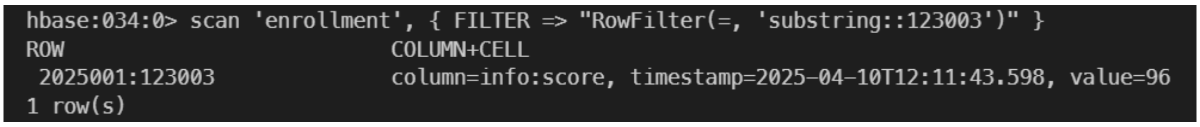


```
hbase:033:0> scan 'course', { FILTER => "SingleColumnValueFilter('info', 'name', =, 'binary:Big Data')" }
ROW                           COLUMN+CELL
 123003                       column=info:credit, timestamp=2025-04-10T11:58:43.094, value=4.0
 123003                       column=info:name, timestamp=2025-04-10T11:58:36.346, value=Big Data
1 row(s)
```

可以看到Big Data的课程号为 `123003`

再在选课表中通过RowFilter查找行键中包含课程号为123003的记录:

```
scan 'enrollment', { FILTER => "RowFilter(=, 'substring::123003')" }
```



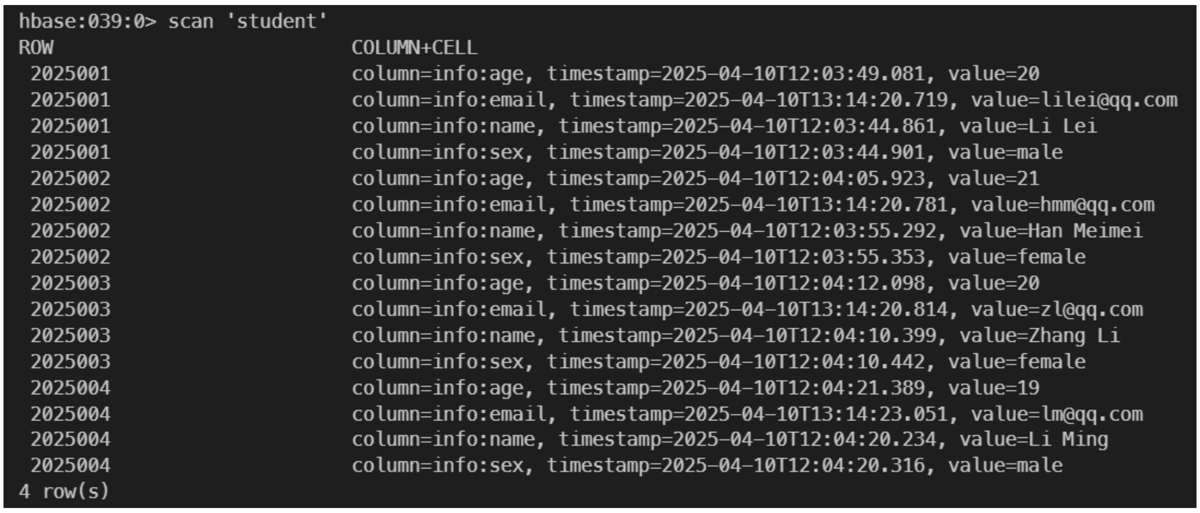
```
hbase:034:0> scan 'enrollment', { FILTER => "RowFilter(=, 'substring::123003')" }
ROW                           COLUMN+CELL
 2025001:123003               column=info:score, timestamp=2025-04-10T12:11:43.598, value=96
1 row(s)
```

可以看到学生2025001选修了Big Data课程, 成绩为 96

学生表增加联系方式

从表中可以看到学生表新增了email这一字段, 只需要在info列族中增加 `info:email` 即可, 直接使用put命令:

```
put 'student', '2025001', 'info:email', 'lilei@qq.com'
put 'student', '2025002', 'info:email', 'hmm@qq.com'
put 'student', '2025003', 'info:email', 'zl@qq.com'
put 'student', '2025004', 'info:email', 'lm@qq.com'
```

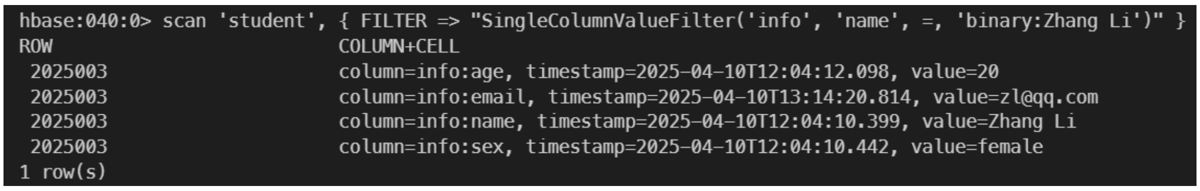


```
hbase:039:0> scan 'student'
ROW                            COLUMN+CELL
 2025001                       column=info:age, timestamp=2025-04-10T12:03:49.081, value=20
 2025001                       column=info:email, timestamp=2025-04-10T13:14:20.719, value=lilei@qq.com
 2025001                       column=info:name, timestamp=2025-04-10T12:03:44.861, value=Li Lei
 2025001                       column=info:sex, timestamp=2025-04-10T12:03:44.901, value=male
 2025002                       column=info:age, timestamp=2025-04-10T12:04:05.923, value=21
 2025002                       column=info:email, timestamp=2025-04-10T13:14:20.781, value=hmm@qq.com
 2025002                       column=info:name, timestamp=2025-04-10T12:03:55.292, value=Han Meimei
 2025002                       column=info:sex, timestamp=2025-04-10T12:03:55.353, value=female
 2025003                       column=info:age, timestamp=2025-04-10T12:04:12.098, value=20
 2025003                       column=info:email, timestamp=2025-04-10T13:14:20.814, value=zl@qq.com
 2025003                       column=info:name, timestamp=2025-04-10T12:04:10.399, value=Zhang Li
 2025003                       column=info:sex, timestamp=2025-04-10T12:04:10.442, value=female
 2025004                       column=info:age, timestamp=2025-04-10T12:04:21.389, value=19
 2025004                       column=info:email, timestamp=2025-04-10T13:14:23.051, value=lm@qq.com
 2025004                       column=info:name, timestamp=2025-04-10T12:04:20.234, value=Li Ming
 2025004                       column=info:sex, timestamp=2025-04-10T12:04:20.316, value=male
4 row(s)
```

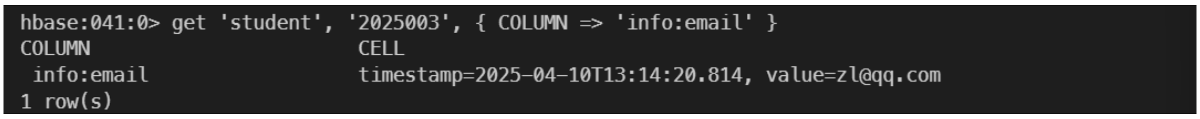
查询Zhang Li的联系方式

在student表中使用 SingleColumnValueFilter 过滤器查找 info:name 等于 "Zhang Li" 的记录

```
scan 'student', { FILTER => "SingleColumnValueFilter('info', 'name', =,
'binary:Zhang Li')" }
get 'student', '2025003', { COLUMN => 'info:email' }
```



```
hbase:040:0> scan 'student', { FILTER => "SingleColumnValueFilter('info', 'name', =, 'binary:Zhang Li')" }
ROW                            COLUMN+CELL
 2025003                       column=info:age, timestamp=2025-04-10T12:04:12.098, value=20
 2025003                       column=info:email, timestamp=2025-04-10T13:14:20.814, value=zl@qq.com
 2025003                       column=info:name, timestamp=2025-04-10T12:04:10.399, value=Zhang Li
 2025003                       column=info:sex, timestamp=2025-04-10T12:04:10.442, value=female
1 row(s)
```



```
hbase:041:0> get 'student', '2025003', { COLUMN => 'info:email' }
COLUMN                         CELL
 info:email                    timestamp=2025-04-10T13:14:20.814, value=zl@qq.com
1 row(s)
```

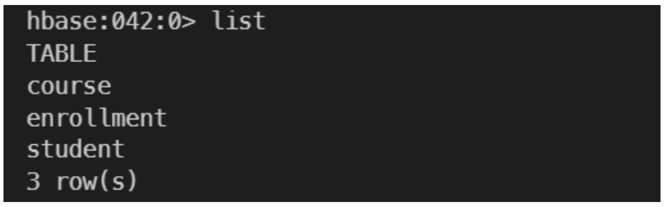
从结果中可以看到, 姓名"Zhang Li"对应的联系方式是zl@qq.com

这种方法的缺陷是需要扫描整个表,效率较低,为了更高效的按姓名查询,可以创建一个辅助表student_by_name，以姓名作为行键，存储学号等信息.

删除创建的表

在 HBase 中，删除表需要先禁用表（disable），然后才能执行删除操作（drop）

首先确认当前有哪些表: `hbase: > list`

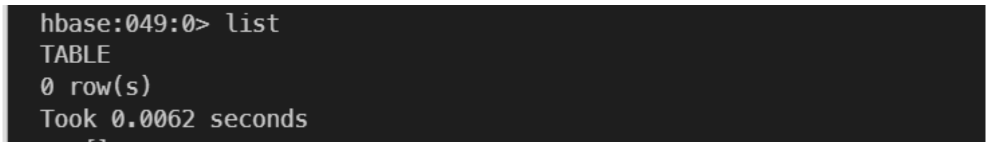


```
hbase:042:0> list
TABLE
course
enrollment
student
3 row(s)
```

我们需要删除的表是 `course`、`enrollment` 和 `student`, 输入命令:

```
disable 'course'
drop 'course'

disable 'student'
drop 'student'

disable 'enrollment'
drop 'enrollment'
```

再次运行 list 命令，确认表已删除：



```
hbase:049:0> list
TABLE
0 row(s)
Took 0.0062 seconds
```

任务二: MapReduce编程与Hive外部表管理

MapReduce编程

我们需要编写一个 MapReduce 程序来处理 IOlog.trace 数据，筛选 op_name=2 的记录，并按 user_namespace 统计 op_count 的总和

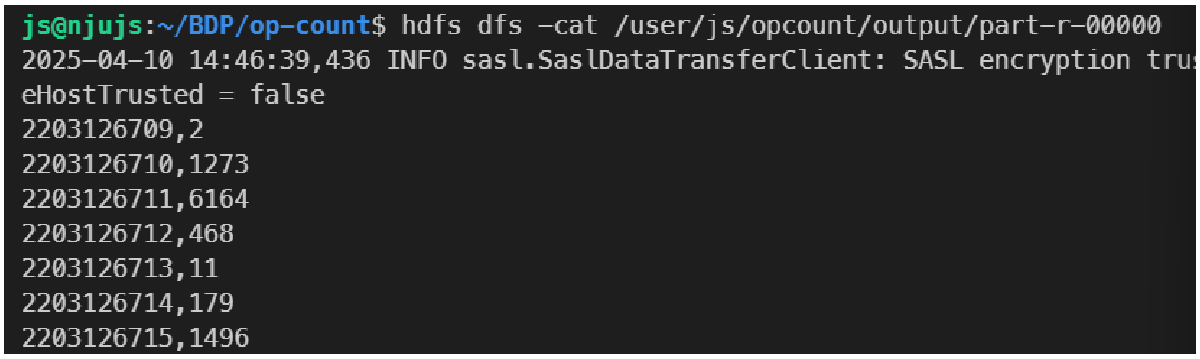
在Map阶段, 我们对每行数据按照空格分隔,获取 op_name 和 user_namespace, op_count, 筛选 op_name = 2的记录, 以`<userNamespace, opCount>` 的格式输出

在Reduce阶段, 对同一个namespace的opcout进行累加, 按照`<user_namespace, sum(op_count)>`格式输出.

编写完成后输入 `mvn clean install` 打包成jar包, 将IOlog.trace上传到HDFS, 输入以下命令运行:

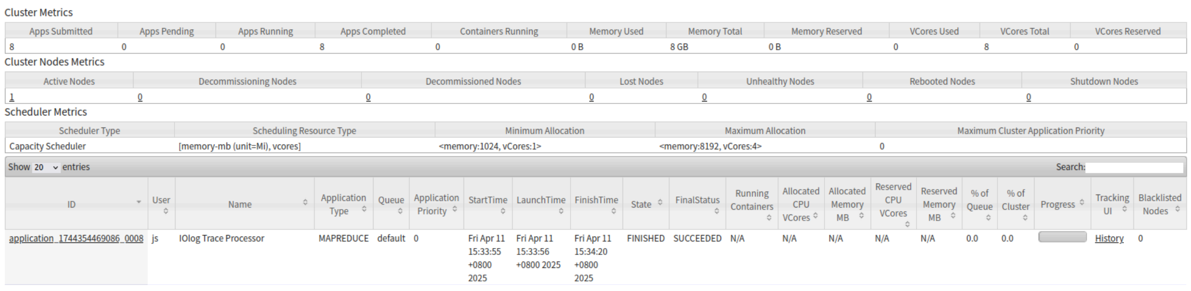
```
hadoop jar op-count-1.0-SNAPSHOT.jar com.example.IOlogTraceProcessor
/user/js/opcount/input/IOlog.trace /user/js/opcount/output
```

任务运行结束后, 输入 `hdfs dfs -cat /user/js/opcount/output/part-r-00000` 查看输出结果(部分):



```
js@njujs:~/BDP/op-count$ hdfs dfs -cat /user/js/opcount/output/part-r-00000
2025-04-10 14:46:39,436 INFO sasl.SaslDataTransferClient: SASL encryption tru
eHostTrusted = false
2203126709,2
2203126710,1273
2203126711,6164
2203126712,468
2203126713,11
2203126714,179
2203126715,1496
```

Yarn监控截图


Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved
8	0	0	8	0	0 B	8 GB	0 B	0	8	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes
1	0	0	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>	0

Show 20 entries Search:

ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	Reserved CPU VCores	Reserved Memory MB	% of Queue	% of Cluster	Progress	Tracking UI	Blacklisted Nodes
application_1744354469086_0008	js	IOlog Trace Processor	MAPREDUCE	default	0	Fri Apr 11 15:33:55 +0800 2025	Fri Apr 11 15:33:56 +0800 2025	Fri Apr 11 15:34:20 +0800 2025	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0

在 Hive 中创建表映射 MapReduce 输出

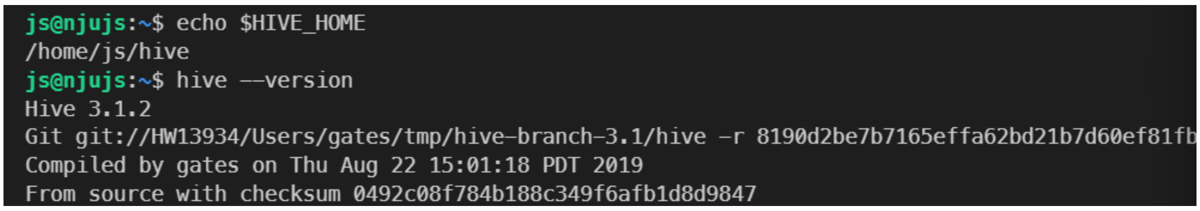
安装并配置Hive

安装,解压并配置bashrc:

```bash
wget https://archive.apache.org/dist/hive/hive-3.1.2/apache-hive-3.1.2-bin.tar.gz
tar -xzf apache-hive-3.1.2-bin.tar.gz
sudo mv apache-hive-3.1.2-bin /home/js/hive

nano ~/.bashrc

# 在文件末尾添加以下内容
export HIVE_HOME=/home/js/hive
export PATH=$PATH:$HIVE_HOME/bin

# 使更改生效
source ~/.bashrc
```

验证环境变量和hive版本:

```bash
echo $HIVE_HOME
hive --version
```



```
js@njujs:~$ echo $HIVE_HOME
/home/js/hive
js@njujs:~$ hive --version
Hive 3.1.2
Git git://HW13934/Users/gates/tmp/hive-branch-3.1/hive -r 8190d2be7b7165effa62bd21b7d60ef81fb
Compiled by gates on Thu Aug 22 15:01:18 PDT 2019
From source with checksum 0492c08f784b188c349f6afb1d8d9847
```

配置hive: 进入hive的conf目录, 创建`hive-site.xml`文件, 添加如下的配置信息, 并在HDFS上创建目录: `hdfs dfs -mkdir -p /user/hive/warehouse`

```xml
<configuration>
    <property>
        <name>hive.metastore.warehouse.dir</name>
        <value>hdfs://localhost:9000/user/hive/warehouse</value>
    </property>
    <property>
        <name>javax.jdo.option.ConnectionURL</name>

 <value>jdbc:derby:;databaseName=metastore_db;create=true</value>
    </property>
    ...
</configuration>
```

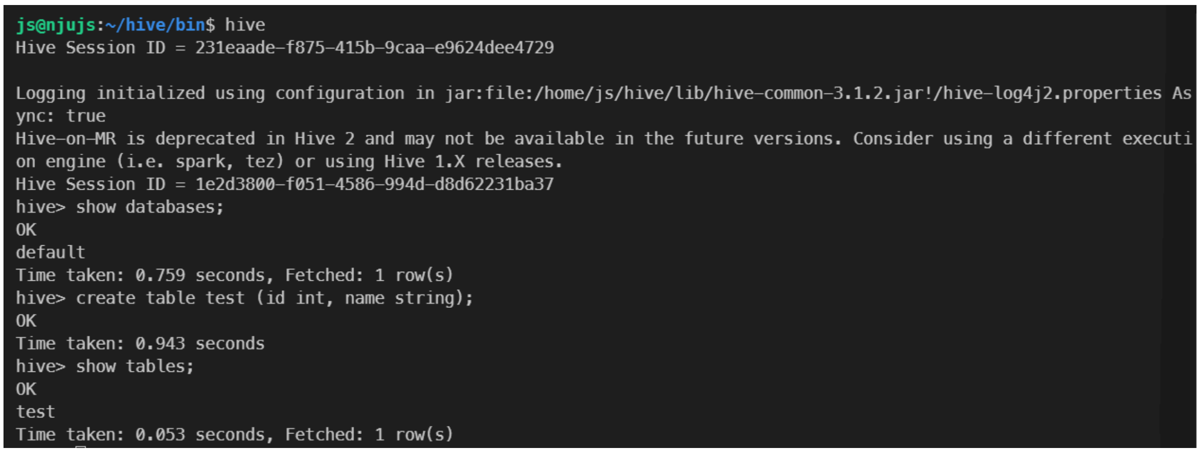
初始化元数据: `$HIVE_HOME/bin/schematool -initSchema -dbType derby`

启动Hive CLI: `hive`

在Hive Shell中运行以下命令:

```sql
show databases;
create table test (id int, name string);
show tables;
```

结果如下图所示, 可以看到Hive正常运行, 安装成功.

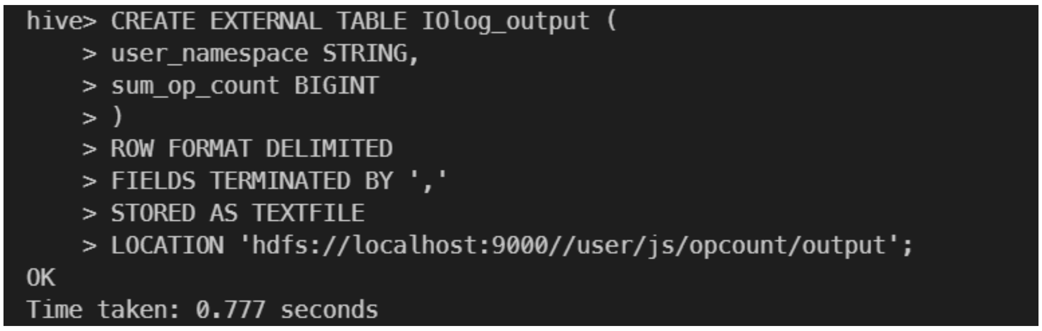


```
js@njujs:~/hive/bin$ hive
Hive Session ID = 231eaade-f875-415b-9caa-e9624dee4729

Logging initialized using configuration in jar:file:/home/js/hive/lib/hive-common-3.1.2.jar!/hive-log4j2.properties As
ync: true
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different executi
on engine (i.e. spark, tez) or using Hive 1.X releases.
Hive Session ID = 1e2d3800-f051-4586-994d-d8d62231ba37
hive> show databases;
OK
default
Time taken: 0.759 seconds, Fetched: 1 row(s)
hive> create table test (id int, name string);
OK
Time taken: 0.943 seconds
hive> show tables;
OK
test
Time taken: 0.053 seconds, Fetched: 1 row(s)
```

在Hive中创建表管理 MapReduce 的输出数据

MapReduce 输出存储在 HDFS 的 `/user/js/opcount/output` 目录中, 我们需要通过 Hive 创建一个外部表映射这个数据, 进入 Hive Shell, 输入以下指令创建表:

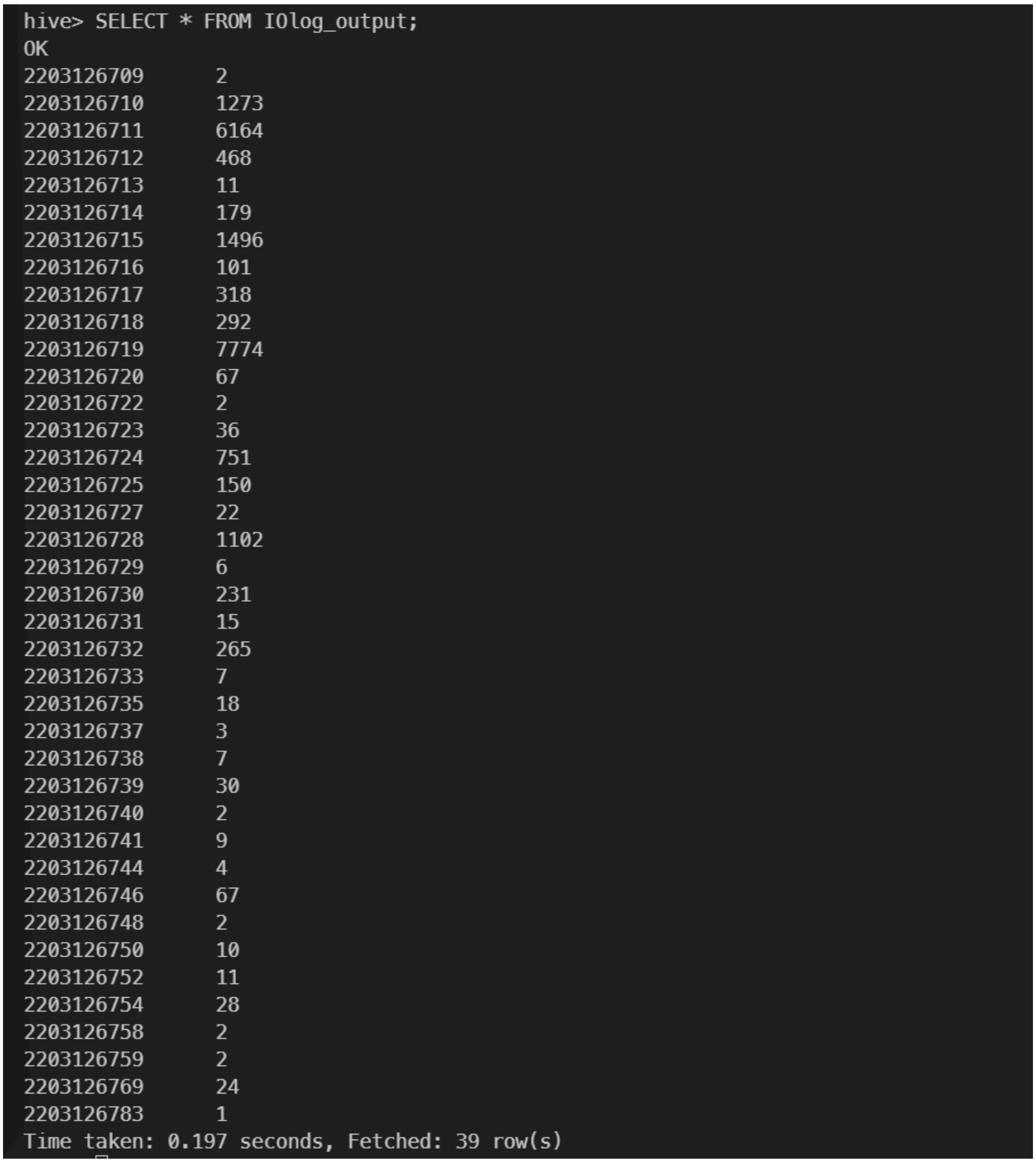
```sql
CREATE EXTERNAL TABLE IOlog_output (
    user_namespace STRING,
    sum_op_count BIGINT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION 'hdfs://localhost:9000//user/js/opcount/output';
```



```
hive> CREATE EXTERNAL TABLE IOlog_output (
    > user_namespace STRING,
    > sum_op_count BIGINT
    > )
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE
    > LOCATION 'hdfs://localhost:9000//user/js/opcount/output';
OK
Time taken: 0.777 seconds
```

输入以下命令查看所有数据

```
SELECT * FROM IOlog_output;
```



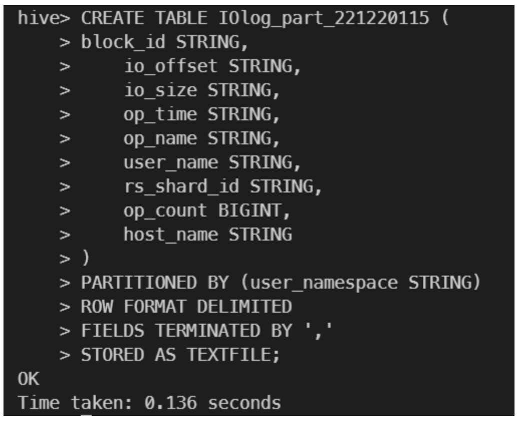
```
hive> SELECT * FROM IOlog_output;
OK
2203126709	2
2203126710	1273
2203126711	6164
2203126712	468
2203126713	11
2203126714	179
2203126715	1496
2203126716	101
2203126717	318
2203126718	292
2203126719	7774
2203126720	67
2203126722	2
2203126723	36
2203126724	751
2203126725	150
2203126727	22
2203126728	1102
2203126729	6
2203126730	231
2203126731	15
2203126732	265
2203126733	7
2203126735	18
2203126737	3
2203126738	7
2203126739	30
2203126740	2
2203126741	9
2203126744	4
2203126746	67
2203126748	2
2203126750	10
2203126752	11
2203126754	28
2203126758	2
2203126759	2
2203126769	24
2203126783	1
Time taken: 0.197 seconds, Fetched: 39 row(s)
```

任务三:Hive分区分桶表与HQL实践

1 Hive中创建分区表

在Hive中创建一张分区表 `IOlog_part_221220115`, 以命名空间(`user_namespace`)为分区条件

```sql
CREATE TABLE iolog_part_221220115 (
    block_id STRING,
    io_offset STRING,
    io_size STRING,
    op_time STRING,
    op_name STRING,
    user_name STRING,
    rs_shard_id STRING,
    op_count BIGINT,
    host_name STRING
)
PARTITIONED BY (user_namespace STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' '
STORED AS TEXTFILE;
```



```
hive> CREATE TABLE IOlog_part_221220115 (
    > block_id STRING,
    >     io_offset STRING,
    >     io_size STRING,
    >     op_time STRING,
    >     op_name STRING,
    >     user_name STRING,
    >     rs_shard_id STRING,
    >     op_count BIGINT,
    >     host_name STRING
    > )
    > PARTITIONED BY (user_namespace STRING)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE;
OK
Time taken: 0.136 seconds
```

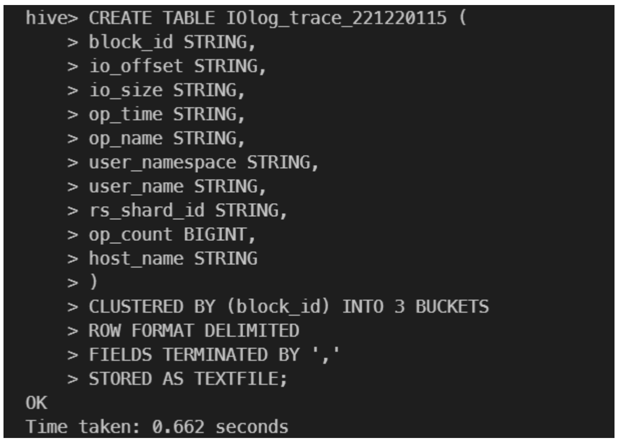
2 Hive中创建分桶表

创建分桶表 `IOlog_trace_221220115`

```sql
CREATE TABLE IOlog_trace_221220115 (
    block_id STRING,
    io_offset STRING,
    io_size STRING,
    op_time STRING,
    op_name STRING,
    user_namespace STRING,
    user_name STRING,
    rs_shard_id STRING,
    op_count BIGINT,
    host_name STRING
)
CLUSTERED BY (block_id) INTO 3 BUCKETS
```

```
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' '
STORED AS TEXTFILE;
```

- `CLUSTERED BY (block_id) INTO 3 BUCKETS`：指定以 `block_id` 作为分桶字段，分桶数量为 **3**。



```
hive> CREATE TABLE IOlog_trace_221220115 (
    > block_id STRING,
    > io_offset STRING,
    > io_size STRING,
    > op_time STRING,
    > op_name STRING,
    > user_namespace STRING,
    > user_name STRING,
    > rs_shard_id STRING,
    > op_count BIGINT,
    > host_name STRING
    > )
    > CLUSTERED BY (block_id) INTO 3 BUCKETS
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE;
OK
Time taken: 0.662 seconds
```

3 导入数据到分区表分桶表

- 导入任意5条数据到分区表, 在向分区表中插入数据时，需要指定分区列的值。（首先需要启用动态分区）

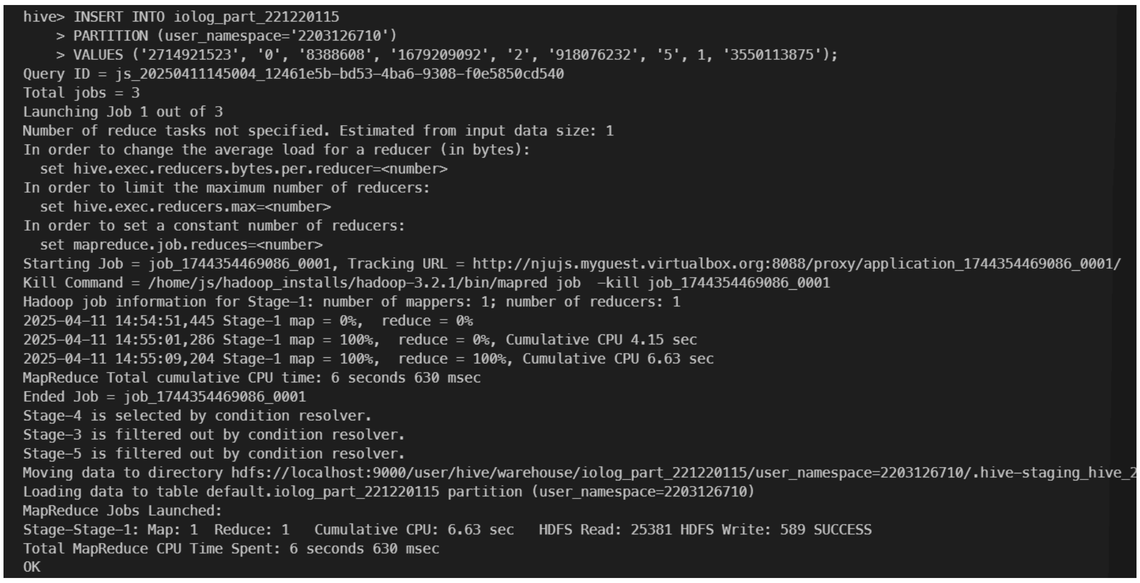
```
SET hive.exec.dynamic.partition=true;
SET hive.exec.dynamic.partition.mode=nonstrict;
```

```
INSERT INTO iolog_part_221220115
PARTITION (user_namespace='xxx')
VALUES (1, 'data1'), (2, 'data2'), (3, 'data3'), (4, 'data4'), (5, 'data5'), (6, 'data6'), (7, 'data7'), (8, 'data8'), (9, 'data9');
```

我们将以下五行数据按上述格式插入到分区表中: (2203126710 * 1, 2203126711 * 3, 2203126715* 1)

```
2714921523 0 8388608 1679209092 2 2203126710 918076232 5 1 3550113875
2714826509 0 4038842 1679209093 5 2203126711 918076230 8 1 3550112758
2714815266 0 256 1679209094 2 2203126711 918076231 0 1 3550112742
2714980205 0 8388608 1679209095 5 2203126711 918076230 10 1 3550114154
2714911581 1563335 4948276 1679209100 2 2203126715 918076229 8 1 3550113732
```

以其中第一条数据为例, 运行过程如下:

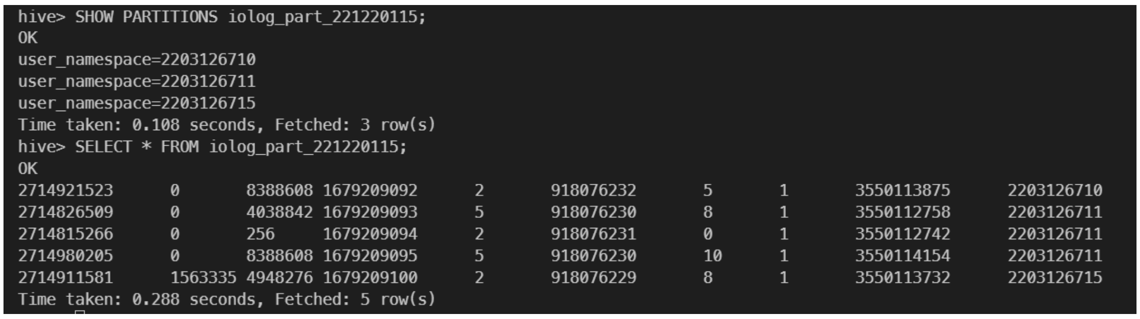


```
hive> INSERT INTO iolog_part_221220115
    > PARTITION (user_namespace='2203126710')
    > VALUES ('2714921523', '0', '8388608', '1679209092', '2', '918076232', '5', 1, '3550113875');
Query ID = js_20250411145004_12461e5b-bd53-4ba6-9308-f0e5850cd540
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks not specified. Estimated from input data size: 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 mapreduce.job.reduces=<number>
Starting Job = job_1744354469086_0001, Tracking URL = http://njujs.myguest.virtualbox.org:8088/proxy/application_1744354469086_0001/
Kill Command = /home/js/hadoop_installs/hadoop-3.2.1/bin/mapred job  -kill job_1744354469086_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2025-04-11 14:54:51,445 Stage-1 map = 0%,  reduce = 0%
2025-04-11 14:55:01,286 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 4.15 sec
2025-04-11 14:55:09,204 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.63 sec
MapReduce Total cumulative CPU time: 6 seconds 630 msec
Ended Job = job_1744354469086_0001
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 directory hdfs://localhost:9000/user/hive/warehouse/iolog_part_221220115/user_namespace=2203126710/.hive-staging_hive_2
Loading data to table default.iolog_part_221220115 partition (user_namespace=2203126710)
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 6.63 sec   HDFS Read: 25381 HDFS Write: 589 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 630 msec
OK
```

Hive 默认使用 MapReduce 作为执行引擎, 会将操作转换为 MapReduce 任务, 日志中的 Total jobs = 3 表示 Hive 将插入操作 分解为 3 个 MapReduce 作业. 按照同样的操作插入剩余四条数据.

- 验证导入是否成功:

```
SHOW PARTITIONS iolog_part_221220115;
SELECT * FROM iolog_part_221220115;
```



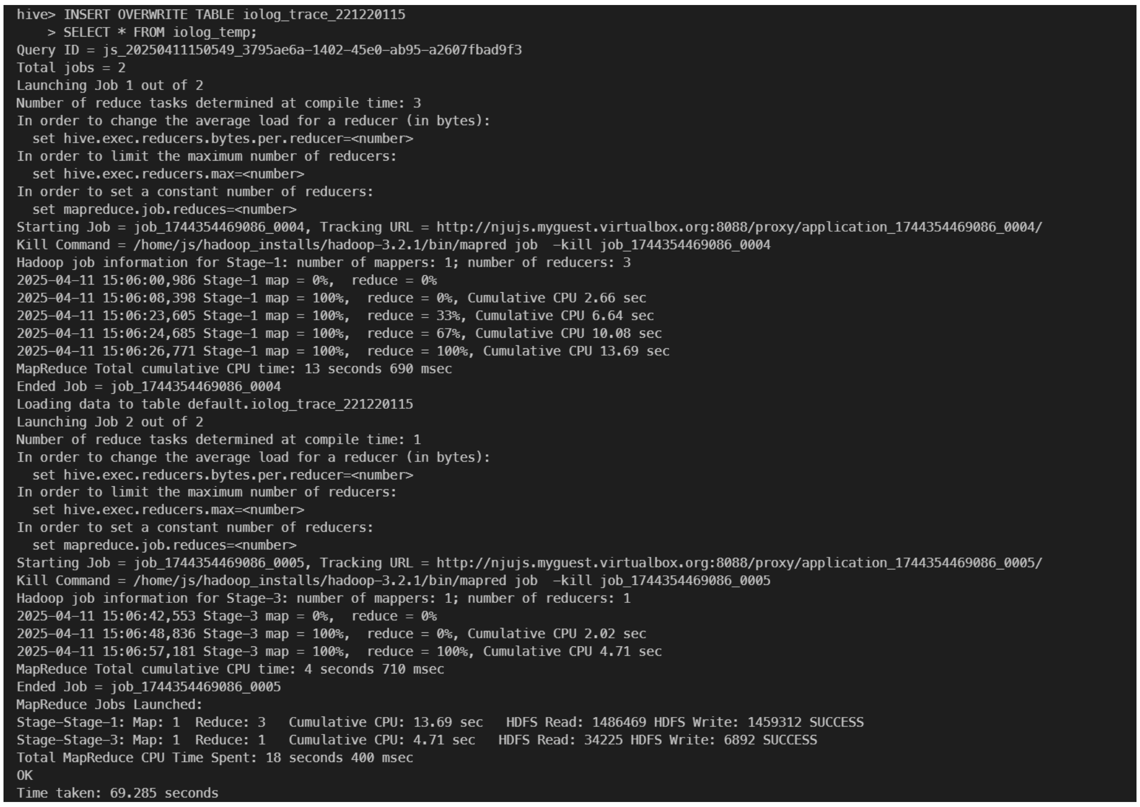
```
hive> SHOW PARTITIONS iolog_part_221220115;
OK
user_namespace=2203126710
user_namespace=2203126711
user_namespace=2203126715
Time taken: 0.108 seconds, Fetched: 3 row(s)
hive> SELECT * FROM iolog_part_221220115;
OK
2714921523      0       8388608 1679209092      2       918076232       5       1       3550113875      2203126710
2714826509      0       4038842 1679209093      5       918076230       8       1       3550112758      2203126711
2714815266      0       256     1679209094      2       918076231       0       1       3550112742      2203126711
2714980205      0       8388608 1679209095      5       918076230       10      1       3550114154      2203126711
2714911581      1563335 4948276 1679209100      2       918076229       8       1       3550113732      2203126715
Time taken: 0.288 seconds, Fetched: 5 row(s)
```

可以看到5条数据成功插入

- 从HDFS将IOlog.trace导入到分桶表中 (需要启动分桶写入`SET hive.enforce.bucketing=true;`)
  首先创建一个临时表IOlog_temp加载IOlog.trace数据:

```
CREATE TABLE IOlog_temp (
    block_id STRING,
    io_offset STRING,
    ...
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION 'hdfs://localhost:9000//user/IOlog/';
```

```
INSERT OVERWRITE TABLE iolog_trace_221220115
SELECT * FROM iolog_temp;
```



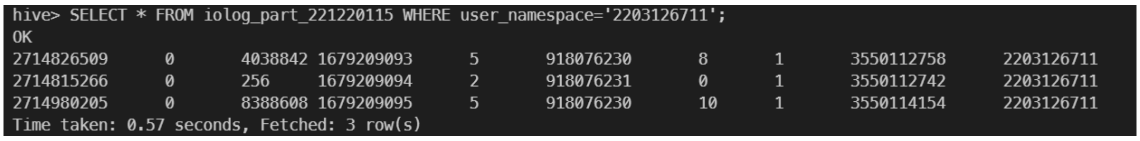
```
hive> INSERT OVERWRITE TABLE iolog_trace_221220115
    > SELECT * FROM iolog_temp;
Query ID = js_20250411150549_3795ae6a-1402-45e0-ab95-a2607fbad9f3
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks determined at compile time: 3
In order to change the average load for a reducer (in bytes):
  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 mapreduce.job.reduces=<number>
Starting Job = job_1744354469086_0004, Tracking URL = http://njujs.myguest.virtualbox.org:8088/proxy/application_1744354469086_0004/
Kill Command = /home/js/hadoop_installs/hadoop-3.2.1/bin/mapred job  -kill job_1744354469086_0004
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 3
2025-04-11 15:06:00,986 Stage-1 map = 0%,  reduce = 0%
2025-04-11 15:06:08,398 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.66 sec
2025-04-11 15:06:23,605 Stage-1 map = 100%,  reduce = 33%, Cumulative CPU 6.64 sec
2025-04-11 15:06:24,685 Stage-1 map = 100%,  reduce = 67%, Cumulative CPU 10.08 sec
2025-04-11 15:06:26,771 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 13.69 sec
MapReduce Total cumulative CPU time: 13 seconds 690 msec
Ended Job = job_1744354469086_0004
Loading data to table default.iolog_trace_221220115
Launching Job 2 out of 2
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 mapreduce.job.reduces=<number>
Starting Job = job_1744354469086_0005, Tracking URL = http://njujs.myguest.virtualbox.org:8088/proxy/application_1744354469086_0005/
Kill Command = /home/js/hadoop_installs/hadoop-3.2.1/bin/mapred job  -kill job_1744354469086_0005
Hadoop job information for Stage-3: number of mappers: 1; number of reducers: 1
2025-04-11 15:06:42,553 Stage-3 map = 0%,  reduce = 0%
2025-04-11 15:06:48,836 Stage-3 map = 100%,  reduce = 0%, Cumulative CPU 2.02 sec
2025-04-11 15:06:57,181 Stage-3 map = 100%,  reduce = 100%, Cumulative CPU 4.71 sec
MapReduce Total cumulative CPU time: 4 seconds 710 msec
Ended Job = job_1744354469086_0005
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 3   Cumulative CPU: 13.69 sec   HDFS Read: 1486469 HDFS Write: 1459312 SUCCESS
Stage-Stage-3: Map: 1  Reduce: 1   Cumulative CPU: 4.71 sec   HDFS Read: 34225 HDFS Write: 6892 SUCCESS
Total MapReduce CPU Time Spent: 18 seconds 400 msec
OK
Time taken: 69.285 seconds
```

4 HQL查询

- 查询分区表中某个分区下的所有数据

  这里选择查询`user_namespace='2203126711'`的所有数据, 应该出现三条. 结果如下:

```sql
SELECT * FROM iolog_part_221220115 WHERE
user_namespace='2203126711';
```



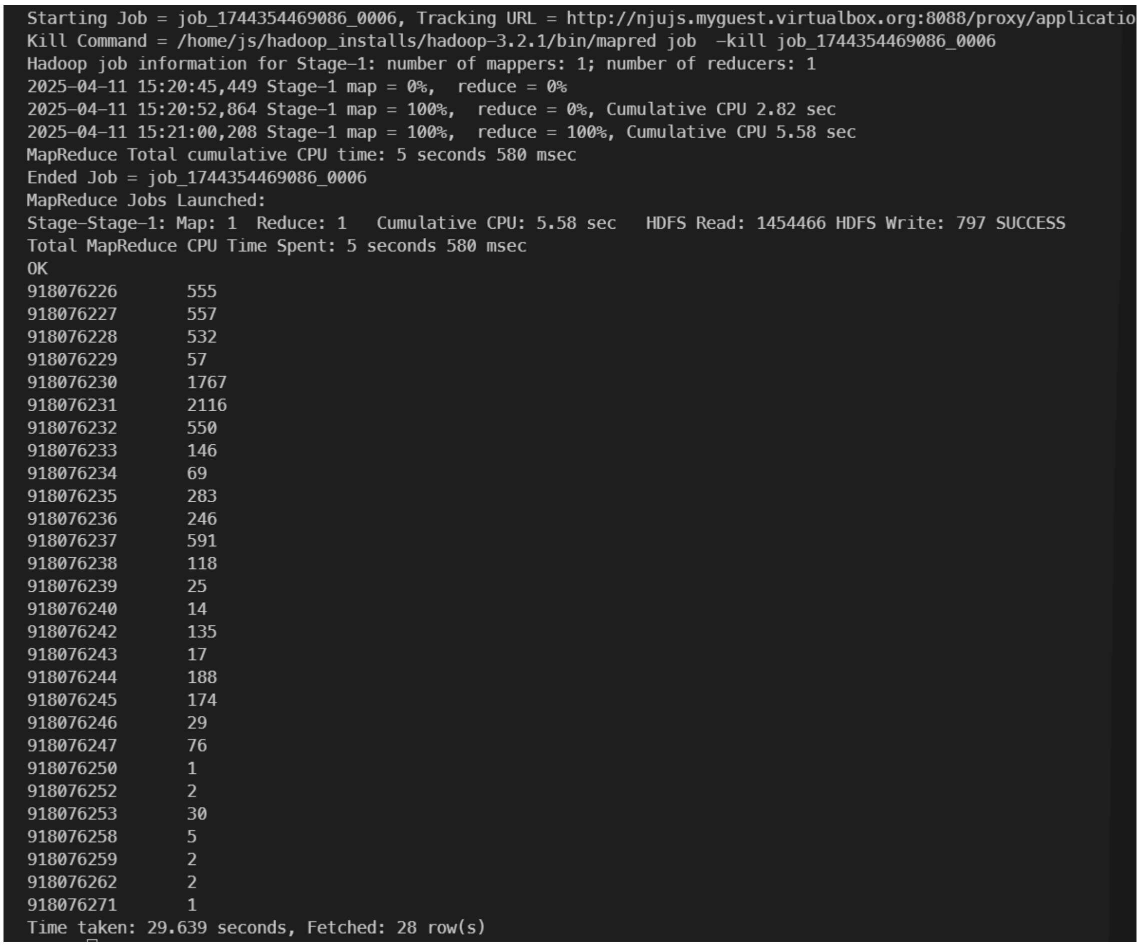
```
hive> SELECT * FROM iolog_part_221220115 WHERE user_namespace='2203126711';
OK
2714826509      0       4038842 1679209093      5       918076230       8       1       3550112758      2203126711
2714815266      0       256     1679209094      2       918076231       0       1       3550112742      2203126711
2714980205      0       8388608 1679209095      5       918076230       10      1       3550114154      2203126711
Time taken: 0.57 seconds, Fetched: 3 row(s)
```

- 查询分桶表中每个用户有几个不同的主机地址(host_name)

  使用查询语句:

```sql
SELECT user_name, COUNT(DISTINCT host_name) AS host_count
FROM iolog_trace_221220115
GROUP BY user_name;
```

  验证结果:



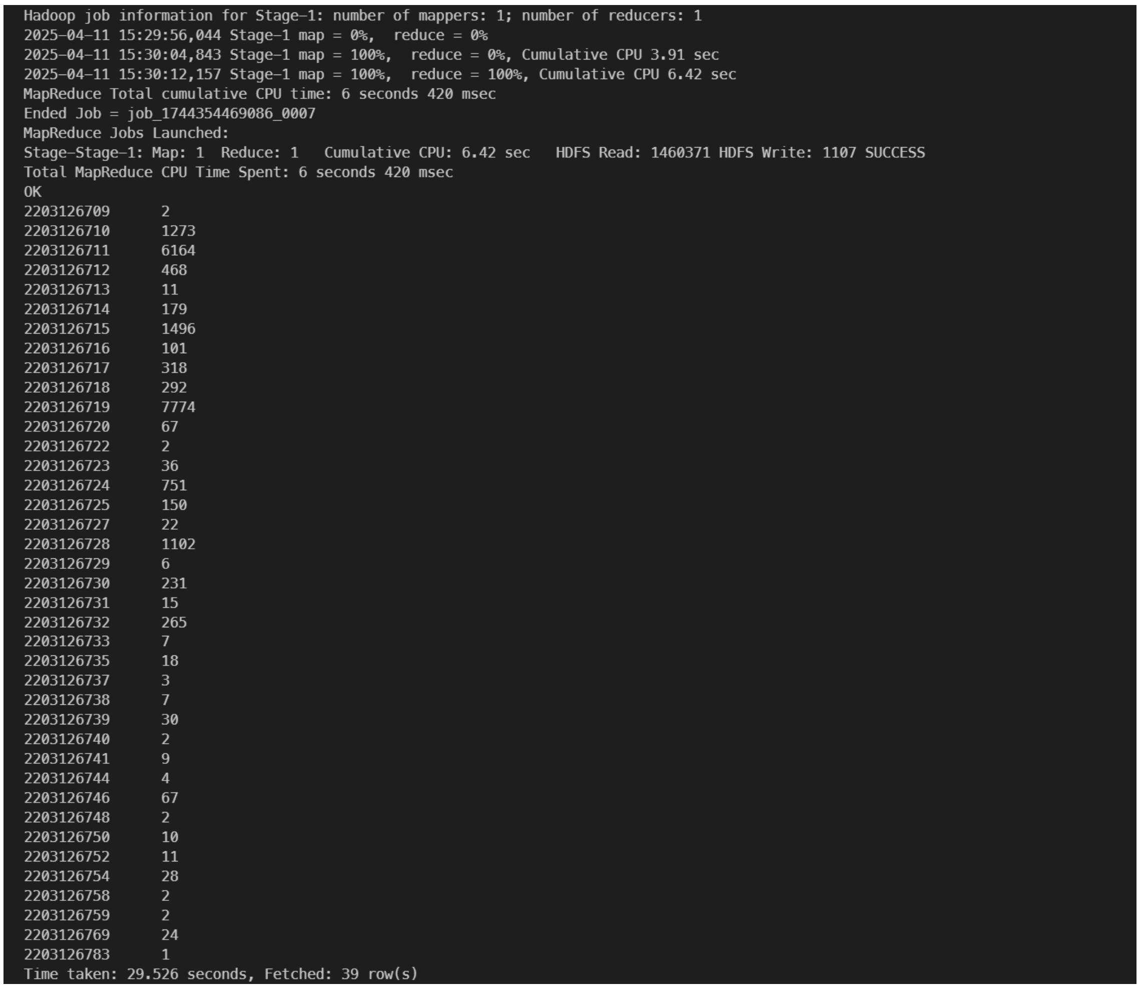
```
Starting Job = job_1744354469086_0006, Tracking URL = http://njujs.myguest.virtualbox.org:8088/proxy/applicatio
Kill Command = /home/js/hadoop_installs/hadoop-3.2.1/bin/mapred job  -kill job_1744354469086_0006
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2025-04-11 15:20:45,449 Stage-1 map = 0%,  reduce = 0%
2025-04-11 15:20:52,864 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.82 sec
2025-04-11 15:21:00,208 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 5.58 sec
MapReduce Total cumulative CPU time: 5 seconds 580 msec
Ended Job = job_1744354469086_0006
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 5.58 sec   HDFS Read: 1454466 HDFS Write: 797 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 580 msec
OK
918076226       555
918076227       557
918076228       532
918076229       57
918076230       1767
918076231       2116
918076232       550
918076233       146
918076234       69
918076235       283
918076236       246
918076237       591
918076238       118
918076239       25
918076240       14
918076242       135
918076243       17
918076244       188
918076245       174
918076246       29
918076247       76
918076250       1
918076252       2
918076253       30
918076258       5
918076259       2
918076262       2
918076271       1
Time taken: 29.639 seconds, Fetched: 28 row(s)
```

- 查询分桶表中每个命名空间下所有用户的写操作(op_name=2)的总次数

  使用HQL查询语句：

```sql
SELECT user_namespace, SUM(op_count) AS total_write_count
FROM iolog_trace_221220115
WHERE op_name = '2'
GROUP BY user_namespace;
```

验证结果：



```
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2025-04-11 15:29:56,044 Stage-1 map = 0%,  reduce = 0%
2025-04-11 15:30:04,843 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 3.91 sec
2025-04-11 15:30:12,157 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.42 sec
MapReduce Total cumulative CPU time: 6 seconds 420 msec
Ended Job = job_1744354469086_0007
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 6.42 sec   HDFS Read: 1460371 HDFS Write: 1107 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 420 msec
OK
2203126709      2
2203126710      1273
2203126711      6164
2203126712      468
2203126713      11
2203126714      179
2203126715      1496
2203126716      101
2203126717      318
2203126718      292
2203126719      7774
2203126720      67
2203126722      2
2203126723      36
2203126724      751
2203126725      150
2203126727      22
2203126728      1102
2203126729      6
2203126730      231
2203126731      15
2203126732      265
2203126733      7
2203126735      18
2203126737      3
2203126738      7
2203126739      30
2203126740      2
2203126741      9
2203126744      4
2203126746      67
2203126748      2
2203126750      10
2203126752      11
2203126754      28
2203126758      2
2203126759      2
2203126769      24
2203126783      1
Time taken: 29.526 seconds, Fetched: 39 row(s)
```