数据集成 作业2实验报告

0 简介

0.1 作业2方向

路线二

0.2 团队情况

组号：第7组

团队成员：

- 181250013 陈思文
- 181250019 陈子合
- 181250023 戴祺佳
- 181250102 孟俊豪

1 团队成员分工

- 陈思文
  - 根据流数据检测恶意机器人。
- 陈子合
  - Kafka接收流数据并转储进MongoDB数据库。
  - 根据数据库表和感兴趣点挖掘相关特征信息。
- 戴祺佳
  - 思考数据中值得感兴趣的部分，提出数据处理的方向和结果。
  - 负责中期检查的展示。
  - 根据数据库表和流数据的结合挖掘相关特征信息。
- 孟俊豪
  - 搭建Hadoop-Hive平台，将数据库表部分数据导入本地MySQL数据库。
  - 数据库表和流数据的转储。
  - 数据可视化展示数据处理的结果。

2 数据获取

2.1 数据库表部分

安装hadoop，步骤截图如下。

```
**********************************************************/
[wind@localhost hadoop]$ sbin/start-all.sh
WARNING: Attempting to start all Apache Hadoop daemons as wind in 10 seconds.
WARNING: This is not a recommended production deployment configuration.
WARNING: Use CTRL-C to abort.
Starting namenodes on [localhost]
localhost: Warning: Permanently added 'localhost' (ECDSA) to the list of known h
osts.
Starting datanodes
Starting secondary namenodes [localhost.localdomain]
localhost.localdomain: Warning: Permanently added 'localhost.localdomain' (ECDSA
) to the list of known hosts.
Starting resourcemanager
Starting nodemanagers
[wind@localhost hadoop]$ jps
5868 SecondaryNameNode
6108 ResourceManager
6572 Jps
5501 NameNode
6237 NodeManager
5630 DataNode
```

安装hive，步骤截图如下。

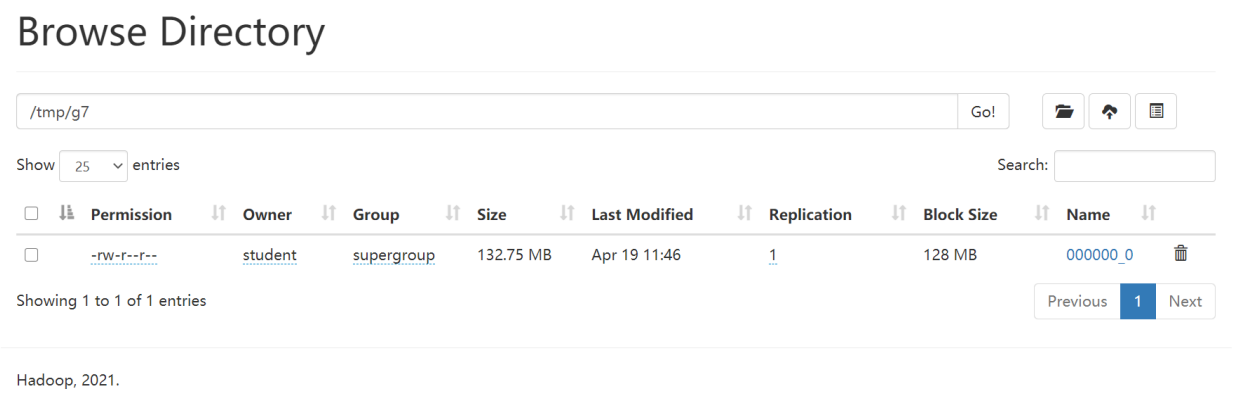
```
[wind@localhost hive]$ cd $HADOOP_HOME
[wind@localhost hadoop]$ sbin/start-all.sh
WARNING: Attempting to start all Apache Hadoop daemons as wind in 10 seconds.
WARNING: This is not a recommended production deployment configuration.
WARNING: Use CTRL-C to abort.
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [localhost.localdomain]
Starting resourcemanager
Starting nodemanagers
[wind@localhost hadoop]$ cd $HIVE_HOME
[wind@localhost hive]$ bin/hive
which: no hbase in (/home/wind/java/bin:/home/wind/java/bin:/home/wind/java/bin:/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/bin:/sbin:/home/wind/.local/bin:/home/wind
/bin)
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/wind/hive/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/wind/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Hive Session ID = ffa31dc7-235d-4803-8c26-1500f26e7de0

Logging initialized using configuration in jar:file:/home/wind/hive/lib/hive-common-3.1.2.jar!/hive-log4j2.properties Async: true
Loading class `com.mysql.jdbc.Driver'. This is deprecated. The new driver class is `com.mysql.cj.jdbc.Driver'. The driver is automatically registered via the SPI and manual
loading of the driver class is generally unnecessary.
Hive Session ID = 71c77933-afb8-46b5-9be1-374626729b62
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
hive>
```

hiveserver2连接，步骤截图如下。

```
wind@localhost:~    wind@localhost:~/hive    wind@localhost:~/hive    wind@localhost:~/hive
[wind@localhost hive]$ bin/hive --service hiveserver2
which: no hbase in (/home/wind/java/bin:/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/bin:/sbin:/home/wind/.local/bin:/home/wind/bin)
2021-05-23 12:57:10: Starting HiveServer2
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/wind/hive/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/wind/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Hive Session ID = 11c19195-fd17-44a7-a1fa-3ff90f34be5f
Loading class `com.mysql.jdbc.Driver'. This is deprecated. The new driver class is `com.mysql.cj.jdbc.Driver'. The driver is automatically registered via the SPI and manual
loading of the driver class is generally unnecessary.
Hive Session ID = f90b766a-4126-4e90-a05b-2e61d3474696
```

```
wind@localhost:~    wind@localhost:~/hive    wind@localhost:~/hive    wind@localhost:~/hive
[wind@localhost hive]$ bin/beeline
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/wind/hive/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/wind/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Beeline version 3.1.2 by Apache Hive
beeline> !connect jdbc:hive2://172.29.4.17:10000
Connecting to jdbc:hive2://172.29.4.17:10000
Enter username for jdbc:hive2://172.29.4.17:10000: student
Enter password for jdbc:hive2://172.29.4.17:10000: *******
Connected to: Apache Hive (version 3.1.2)
Driver: Hive JDBC (version 3.1.2)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://172.29.4.17:10000> show tables;
INFO  : Compiling command(queryId=root_20210523125744_fca8e817-a480-46d4-b32d-608073beb580): show tables
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:tab_name, type:string, comment:from deserializer)], properties:null)
INFO  : Completed compiling command(queryId=root_20210523125744_fca8e817-a480-46d4-b32d-608073beb580); Time taken: 0.022 seconds
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Executing command(queryId=root_20210523125744_fca8e817-a480-46d4-b32d-608073beb580): show tables
INFO  : Starting task [Stage-0:DDL] in serial mode
INFO  : Completed executing command(queryId=root_20210523125744_fca8e817-a480-46d4-b32d-608073beb580); Time taken: 0.087 seconds
INFO  : OK
INFO  : Concurrency mode is disabled, not creating a lock manager
+-----------+
| tab_name  |
+-----------+
| buy_data  |
+-----------+
1 row selected (0.219 seconds)
0: jdbc:hive2://172.29.4.17:10000>
```

导出HDFS，网站截图如下。



安装sqoop，步骤截图如下。

```
[wind@localhost bin]$ ./sqoop version
Error: /usr/local/hadoop does not exist!
Please set $HADOOP_COMMON_HOME to the root of your Hadoop installation.
[wind@localhost bin]$ ./sqoop version
Warning: /home/wind/sqoop/../hbase does not exist! HBase imports will fail.
Please set $HBASE_HOME to the root of your HBase installation.
Warning: /home/wind/sqoop/../hcatalog does not exist! HCatalog jobs will fail.
Please set $HCAT_HOME to the root of your HCatalog installation.
Warning: /home/wind/sqoop/../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
Warning: /home/wind/sqoop/../zookeeper does not exist! Accumulo imports will fail.
Please set $ZOOKEEPER_HOME to the root of your Zookeeper installation.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/wind/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/wind/sqoop/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
2021-05-23 13:05:24,428 INFO sqoop.Sqoop: Running Sqoop version: 1.4.7
Sqoop 1.4.7
git commit id 2328971411f57f0cb683dfb79d19d4d19d185dd8
Compiled by maugli on Thu Dec 21 15:59:58 STD 2017
```

sqoop导入数据，相关命令如下。

```
bin/sqoop export --connect jdbc:mysql://localhost:3306/hive --table buy_data --username root --
password password --export-dir hdfs://172.29.4.17:9000/tmp/buy_gen/part-m-00000
```

查看本地MySQL，截图如下。

```
| WRITE_SET                    |
| buy_data                     |
+------------------------------+
75 rows in set (0.02 sec)

mysql> select * from buy_data limit 50;
+---------+---------+---------+------------+-----+----------+
| id      | user_id | item_id | category_id | type | timestamp |
+---------+---------+---------+------------+-----+----------+
| 1029459 | 4680577 | 2062663 | 1511947159 | buy | 343959 |
| 3598912 | 4680578 | 4209472 | 1511947159 | buy | 380419 |
| 4284875 | 4680579 | 2894092 | 1511947159 | buy | 919275 |
| 2495340 | 4680580 | 3310369 | 1511947159 | buy | 933272 |
| 3720767 | 4680581 | 2154381 | 1511947160 | buy | 95298 |
| 2877672 | 4680582 | 1086988 | 1511947160 | buy | 338694 |
| 502123 | 4680583 | 2631932 | 1511947160 | buy | 398459 |
| 3607361 | 4680584 | 4971002 | 1511947160 | buy | 672870 |
| 375240 | 4680585 | 496831 | 1511947161 | buy | 540175 |
| 3189162 | 4680586 | 3395215 | 1511947161 | buy | 556628 |
| 2736436 | 4680587 | 1764751 | 1511947161 | buy | 859880 |
| 1575622 | 4680588 | 4119959 | 1511947161 | buy | 967224 |
| 479163 | 4680589 | 3018820 | 1511947162 | buy | 718179 |
| 1896675 | 4680590 | 1372491 | 1511947162 | buy | 728389 |
| 715487 | 4680591 | 431878 | 1511947162 | buy | 881957 |
| 4449178 | 4680592 | 3495406 | 1511947163 | buy | 508154 |
| 4756105 | 4680593 | 2187144 | 1511947163 | buy | 765055 |
| 1646753 | 4680594 | 1329868 | 1511947163 | buy | 941610 |
| 2314022 | 4680595 | 2092215 | 1511947163 | buy | 941610 |
| 4197505 | 4680596 | 1865705 | 1511947163 | buy | 946611 |
| 1646753 | 4680597 | 3016830 | 1511947163 | buy | 972530 |
| 1281603 | 4680598 | 3535359 | 1511947164 | buy | 427994 |
| 4145813 | 4680599 | 3997635 | 1511947165 | buy | 47761 |
```

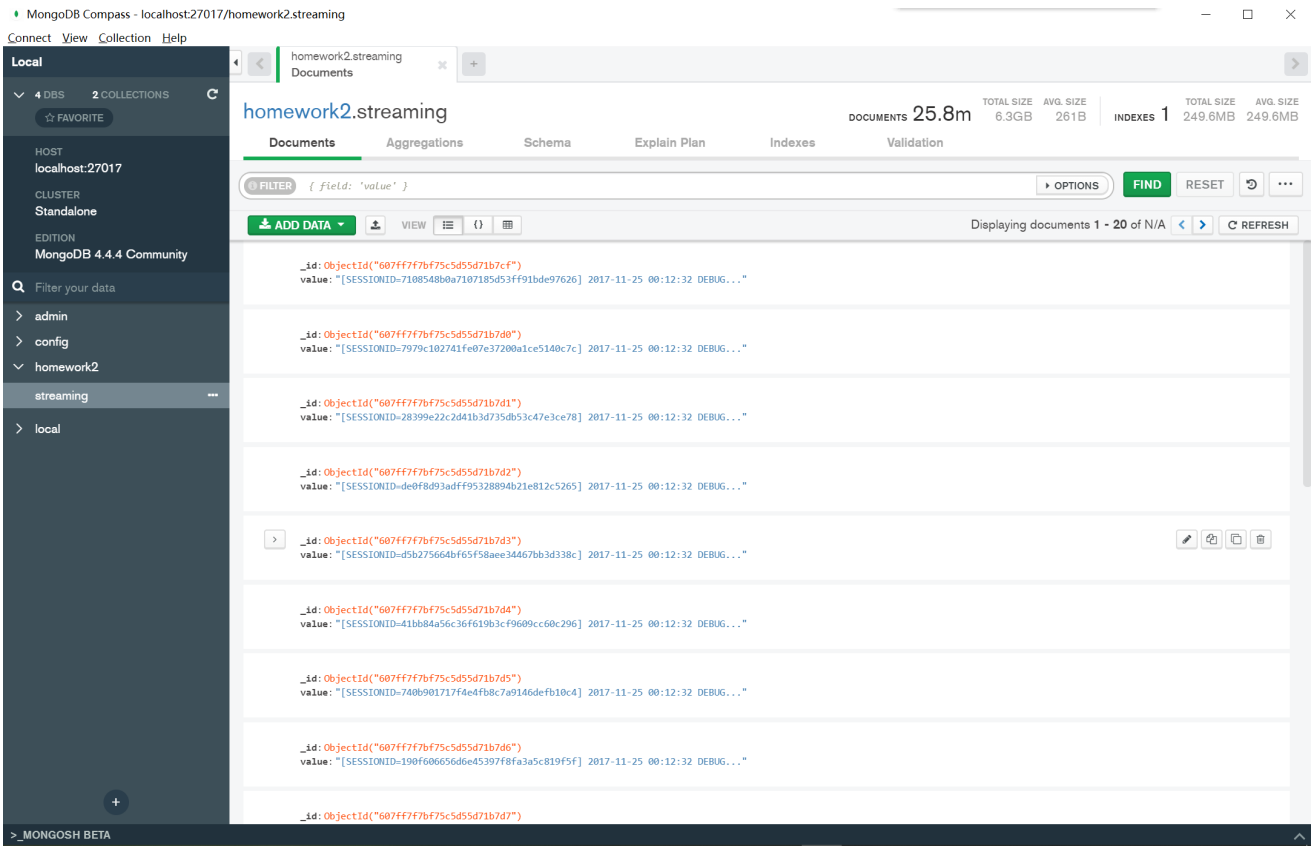
2.2 流数据部分

通过Kafka接收流数据并将流数据存储于本地MongoDB中。

相关代码

```python
from kafka import KafkaConsumer
import pymongo

consumer = KafkaConsumer(
    'foobar',
    bootstrap_servers='172.29.4.17:9092',
    security_protocol='SASL_PLAINTEXT',
    sasl_mechanism='PLAIN',
    sasl_plain_username='student',
    sasl_plain_password='nju2021',
)

# 多个 consumer 可以重复消费相同的日志，每个 consumer 只会消费到它启动后产生的日志，不会拉到之前的余量
dataClient = pymongo.MongoClient(host="localhost:27017", username="root", password="hyzyj2007")
db = dataClient['dataIntegration']
collection = db['robots']

for msg in consumer:
    line = msg.value.decode("utf-8")
    collection.insert_one({"value": line})
```

步骤截图（此时数据已经从一位组员的MongoDB中转储于另一位组员的MongoDB中）



2.3 数据转储

2.3.1 数据库表部分

数据库表数据已经存储于本地MySQL中。

2.3.2 流数据部分

将流数据从MongoDB中取出转储于本地MySQL中。

相关代码

```
if __name__ == '__main__':
    mongodb = mongo_test()
    mysql = mysql_test()
    cursor = mysql.cursor()

    # sql = "select * from buy_data limit 50"
    # cursor.execute(sql)
    # rows = cursor.fetchall()
    # for row in rows:
    #     print(row)

    count = 0
    for document in mongodb.find():
        value = document["value"]
        SESSIONID = value.split("[SESSIONID=")[1].split("] ")[0]
        url = value.split(": uri=")[1].split(" | ")[0]
        try:
            requestBody = eval(value.split("requestBody = ")[1])
        except:
```

```
            requestBody = eval(value.split("requestBody=")[1])
        try:
            date = value.split(" DEBUG [")[0].split("] ")[2]
        except:
            date = value.split(" DEBUG [")[0].split("] ")[1]
        if url == "/user/login":
            IPADDR = value.split("IPADDR=")[1].split("] [")[0]
            userId = getValue(requestBody, "userId")
            password = getValue(requestBody, "password")
            authCode = getValue(requestBody, "authCode")
            success = getValue(requestBody, "success")
            sql = "insert into streaming" \
                  "(id, ipAddr, sessionId, date, url, userId, password, authCode, success)" \
                  "values" \
                  "('%d','%s','%s','%s','%s','%d','%s','%s','%d')" % \
                  (count, IPADDR, SESSIONID, date, url, userId, password, authCode, success)
            try:
                cursor.execute(sql)
                mysql.commit()
            except:
                mysql.rollback()
        else:
            userId = getValue(requestBody, "userId")
            itemId = getValue(requestBody, "itemId")
            categoryId = getValue(requestBody, "categoryId")
            if url == "/item/buy":
                isSecondKill = getValue(requestBody, "isSecondKill")
                sql = "insert into streaming" \
                      "(id, sessionId, date, url, userId, itemId, categoryId, isSecondKill)" \
                      "values" \
                      "('%d','%s','%s','%s','%d','%d','%d','%d')" % \
                      (count, SESSIONID, date, url, userId, itemId, categoryId, isSecondKill)
                cursor.execute(sql)
                try:
                    mysql.commit()
                except:
                    mysql.rollback()
            else:
                sql = "insert into streaming" \
                      "(id, sessionId, date, url, userId, itemId, categoryId)" \
                      "values" \
                      "('%d','%s','%s','%s','%d','%d','%d')" % \
                      (count, SESSIONID, date, url, userId, itemId, categoryId)
                cursor.execute(sql)
                try:
                    mysql.commit()
                except:
                    mysql.rollback()
        count += 1
    mysql.close()
```

步骤截图

```
mysql> use hive;
Database changed
mysql> show tables;
+----------------+
| Tables_in_hive |
+----------------+
| buy_data       |
| streaming      |
+----------------+
2 rows in set (0.00 sec)

mysql> select * from streaming limit 20;
+----+--------+----------------------------------+---------------------+-----------------+---------+---------+------------+--------------+----------+----------+---------+
| id | ipAddr | sessionId                        | date                | url             | userId  | itemId  | categoryId | isSecondKill | password | authCode | success |
+----+--------+----------------------------------+---------------------+-----------------+---------+---------+------------+--------------+----------+----------+---------+
|  0 | NULL   | 7108548b0a7107185d53ff91bde97626 | 2017-11-25 00:12:32 | /item/getDetail |  850563 | 1344308 |    4145813 |         NULL | NULL     | NULL     |    NULL |
|  1 | NULL   | 7979c102741fe07e37200a1ce5140c7c | 2017-11-25 00:12:32 | /item/getDetail |  924986 |  668581 |    2465336 |         NULL | NULL     | NULL     |    NULL |
|  2 | NULL   | 28399e22c2d41b3d735db53c47e3ce78 | 2017-11-25 00:12:32 | /item/getDetail |  937069 |  992789 |    2520377 |         NULL | NULL     | NULL     |    NULL |
|  3 | NULL   | de0f8d93adff95328894b21e812c5265 | 2017-11-25 00:12:32 | /item/getDetail |  973434 | 3493253 |     982926 |         NULL | NULL     | NULL     |    NULL |
|  4 | NULL   | d5b275664bf65f58aee34467bb3d338c | 2017-11-25 00:12:32 | /item/getDetail |   97905 | 5132811 |    1320293 |         NULL | NULL     | NULL     |    NULL |
|  5 | NULL   | 41bb84a56c36f619b3cf9609cc60c296 | 2017-11-25 00:12:32 | /item/getDetail |  146098 | 1883854 |     344833 |         NULL | NULL     | NULL     |    NULL |
|  6 | NULL   | 740b901717f4e4fb8c7a9146defb10c4 | 2017-11-25 00:12:32 | /item/getDetail |   29221 |   28402 |    1787510 |         NULL | NULL     | NULL     |    NULL |
|  7 | NULL   | 190f606656d6e45397f8fa3a5c819f5f | 2017-11-25 00:12:32 | /item/getDetail |  416018 | 2393824 |     737184 |         NULL | NULL     | NULL     |    NULL |
|  8 | NULL   | 3146219b5273ab5c3ee4f48bdc27cec4 | 2017-11-25 00:12:32 | /item/getDetail |  611739 |  331492 |    1040356 |         NULL | NULL     | NULL     |    NULL |
|  9 | NULL   | 8ae2726f0775aa18be738b665871f384 | 2017-11-25 00:12:32 | /item/getDetail |  643876 |  525233 |    4581579 |         NULL | NULL     | NULL     |    NULL |
| 10 | NULL   | e42817b01ece58453a86e4119d42fd3a | 2017-11-25 00:12:32 | /item/getDetail |  682500 | 1613488 |     982926 |         NULL | NULL     | NULL     |    NULL |
| 11 | NULL   | 8928f9e4e51a8a38c59a695503bc68a6 | 2017-11-25 00:12:32 | /item/getDetail |  687994 | 5116840 |     982926 |         NULL | NULL     | NULL     |    NULL |
| 12 | NULL   | e0947265dfc459b9f924b851dd14c314 | 2017-11-25 00:12:32 | /item/getDetail |  720017 | 1697083 |    1397912 |         NULL | NULL     | NULL     |    NULL |
| 13 | NULL   | ce1d8e718a50565d02c3d1f393c9f745 | 2017-11-25 00:12:32 | /item/getDetail |  793020 | 4660114 |     285583 |         NULL | NULL     | NULL     |    NULL |
| 14 | NULL   | 80481f8fe961f3fe689848217298360b | 2017-11-25 00:12:32 | /item/getDetail |  833504 | 1076288 |    2204852 |         NULL | NULL     | NULL     |    NULL |
| 15 | NULL   | 2e441508139c53ef7c35e28b3e8d3a7f | 2017-11-25 00:12:32 | /item/getDetail | 1013602 | 3161072 |    2465336 |         NULL | NULL     | NULL     |    NULL |
| 16 | NULL   | ba790010d8dad418508f1fd3d84df95a | 2017-11-25 00:12:32 | /item/getDetail |  104145 |  229875 |    1320293 |         NULL | NULL     | NULL     |    NULL |
| 17 | NULL   | 925b2f8e0c86545238d7030a6edc8daf | 2017-11-25 00:12:32 | /item/getDetail |  242439 | 1403169 |    5071267 |         NULL | NULL     | NULL     |    NULL |
| 18 | NULL   | 0cc899eb5ca1970ac127fa4f7d66857f | 2017-11-25 00:12:32 | /item/getDetail |  343766 | 2473574 |    1029459 |         NULL | NULL     | NULL     |    NULL |
| 19 | NULL   | d348bca9766f938d85f8320f5eb3f941 | 2017-11-25 00:12:32 | /item/getDetail |  360160 | 4074937 |     149192 |         NULL | NULL     | NULL     |    NULL |
+----+--------+----------------------------------+---------------------+-----------------+---------+---------+------------+--------------+----------+----------+---------+
20 rows in set (0.00 sec)
```

2.3.3 远端服务器

为了方便团队成员使用，我们将数据转储于服务器上的MySQL中，截图如下。

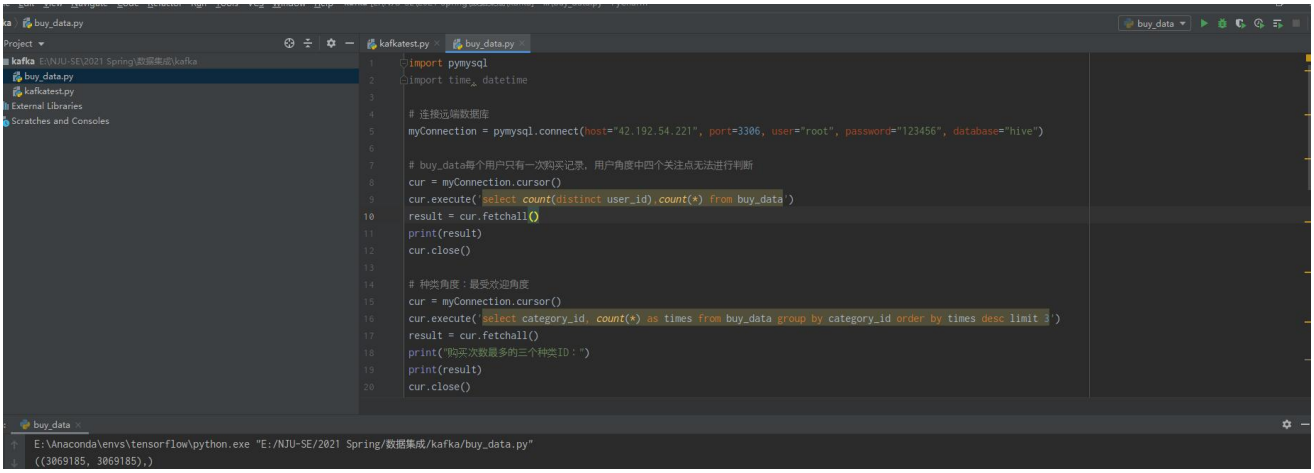
```
PS C:\Users\25702> mysql -h 42.192.54.221 -uroot -p
Enter password: ******
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 213
Server version: 5.7.34 MySQL Community Server (GPL)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use hive;
Database changed
mysql> show tables;
+----------------+
| Tables_in_hive |
+----------------+
| buy_data       |
| streaming      |
+----------------+
2 rows in set (0.01 sec)
```

3 数据处理

3.1 数据库表部分

3.1.1 用户角度

由于在获取的流数据当中，每位用户只够买了一件商品，因此该流数据不具有从用户角度分析的价值。截图如下。



3.1.2 商品类别角度

3.1.2.1 最受欢迎的商品类型

在数据库表中寻找被购买次数最多的商品类别。

相关代码

```
cur = myConnection.cursor()
cur.execute('select category_id, count(*) as times from buy_data group by category_id order by times desc limit 10')
result = cur.fetchall()
print("购买次数最多的十个种类ID: ")
print(result)
mostPopularC = []
for ret in result:
    mostPopularC.append({"categoryID": ret[0], "count": ret[1]})
cur.close()
```

3.1.2.2 最近热销的商品类型

在数据库表中寻找最近的一周内被购买次数最多的商品类别。

相关代码

```
cur = myConnection.cursor()
cur.execute(
    'select category_id, count(*) as times from buy_data where buy_data.timestamp > 413211 group by category_id order by times desc limit 10')
result = cur.fetchall()
print("最近一周内购买次数前十的种类ID: ")
print(result)
recentPopularC = []
for ret in result:
    recentPopularC.append({"categoryID": ret[0], "count": ret[1]})
cur.close()
```

3.1.3 商品角度

3.1.3.1 最受欢迎的商品

在数据库表中寻找被购买次数最多的商品。

相关代码

```
cur = myConnection.cursor()
cur.execute('select item_id, count(*) as times from buy_data group by item_id order by times desc limit 10')
result = cur.fetchall()
print("购买次数最多的十个商品ID: ")
print(result)
mostPopularI = []
for item in result:
    mostPopularI.append({"itemID": item[0], "count": item[1]})
cur.close()
```

3.1.3.2 最近热销的商品

在数据库表中寻找最近的一周内被购买次数最多的商品。

相关代码

```
cur = myConnection.cursor()
cur.execute(
    'select item_id, count(*) as times from buy_data where buy_data.timestamp > 413211 group by item_id order by times desc limit 10')
result = cur.fetchall()
print("最近一周内购买次数前十的商品ID: ")
print(result)
recentPopularI = []
for item in result:
    recentPopularI.append({"itemID": item[0], "count": item[1]})
cur.close()
```

3.2 流数据部分——恶意机器人

3.2.1 撞库机器人

选择短时间内某特定ID登录的成功率作为主要排序依据，升序排序，筛选全体ID中的前5000（约总ID数的0.5%），筛选结束后将原数据库中的对应ID的所有记录进行删除。

相关代码

```
# 获取撞库机器人
cursor = myConnection.cursor()
print(myConnection)
print(cursor)
sql = "SELECT distinct userId , count(case when success = '1' then '1' end )/count(*) as successrate FROM " \
      "streaming where url = '/user/login' group by userId order by successrate limit 5000 "
cursor.execute(sql)
rr = cursor.fetchall()
```

```
Robot1 = []
result1 = []
for row in rr:
    print(row)
    Robot1.append({"userID": row[0]})
    result1.append(row[0])
for i in result1:
    try:
        sql_delete = "DELETE FROM streaming WHERE userId= '%s'"
        cursor.execute(sql_delete, i)
        myConnection.commit()
        print("done"+str(i))
    except Exception as e:
        print(e)

cursor.close()
```

3.2.2 抢单机器人

选择短时间内特定ID购买成功次数与购买、抢单操作总数的比值作为主要排序依据，进行升序排序，筛选全体ID中的前5000（约总ID数的0.5%），筛选结束后将原数据库中的对应ID的所有记录进行删除。

相关代码

```
# 获取抢单机器人
cursor = myConnection.cursor()
print(myConnection)
print(cursor)
sql = "SELECT distinct userId , count(case when url = '/item/buy' then '1' end )/count(case when
url = '/item/cart' " \
      "or url = '/item/buy'" \
      "then '1' end ) as successrate from streaming where url = '/item/buy' or url =
'/item/cart' group by userId " \
      "order by successrate desc limit 5000 "
cursor.execute(sql)
rr = cursor.fetchall()
Robot2 = []
result2 = []
for row in rr:
    print(row)
    Robot2.append({"userID": row[0]})
    result2.append(row[0])
'''
for i in result2:
    try:
        sql_delete = "DELETE FROM streaming WHERE userId= '%s'"
        cursor.execute(sql_delete, i)
        myConnection.commit()
        print("done"+str(i))
    except Exception as e:
        print(e)
'''
cursor.close()
```

3.2.3 刷单机器人

选择180s内某特定ID购买成功的次数作为主要排序依据，筛选全体ID中的前5000（约总ID数的0.5%），降序排序，筛选结束后将原数据库中的对应ID的所有记录进行删除。

相关代码

```
cursor = myConnection.cursor()
print(myConnection)
print(cursor)
# sql1 = "SELECT * FROM streaming WHERE timestampdiff(MINUTE, SYSDATE(), send_time) <=3 AND
timestampdiff(MINUTE, SYSDATE(), send_time) >= 0 "
sql2 = "SELECT distinct userId, count(itemId) as num from streaming where timestampdiff(MINUTE,
SYSDATE(), send_time) " \
       "<=3 timestampdiff(MINUTE, SYSDATE(), send_time) >= 0 and url = '/item/buy' group by
userId order by " \
       "num desc limit 5000 "
cursor.execute(sql2)
rr = cursor.fetchall()
Robot3 = []
result3 = []
for row in rr:
    print(row)
    Robot3.append({"userID": row[0]})
    result3.append(row[0])
'''
for i in result3:
    try:
        sql_delete = "DELETE FROM streaming WHERE userId= '%s'"
        cursor.execute(sql_delete, i)
        myConnection.commit()
        print("done"+str(i))
    except Exception as e:
        print(e)
'''
cursor.close()
```

3.2.4 爬虫机器人

选择短时间内某特定IP登录的成功登录次数主要排序依据，降序排序，筛选全体IP中的前1000（约总ID数的0.5%）；统计这1000个IP对应的所有ID，筛选结束后将原数据库中的对应ID的所有记录进行删除。

相关代码

```
cursor = myConnection.cursor()
print(myConnection)
print(cursor)
sql = "SELECT ipAddr, count(userId) as num from streaming where url = '/user/login' group by
ipAddr order by num desc " \
      "limit 1000 "

cursor.execute(sql)
rr = cursor.fetchall()
Robot4 = []
result4 = []
for row in rr:
    print(row)
    sql2 = "SELECT distinct userId from streaming where ipAddr =" + "'" + str(row[0]) + "'"
    # print(sql2)
```

```
    cursor.execute(sql2)
    rrtemp = cursor.fetchall()
    for row1 in rrtemp:
        # print(row1)
        Robot4.append({"userId": row1[0]})
        result4.append(row1[0])
'''
for i in result4:
    try:
        sql_delete = "DELETE FROM streaming WHERE userId= '%s'"
        cursor.execute(sql_delete, i)
        myConnection.commit()
        print("done"+str(i))
    except Exception as e:
        print(e)
'''
cursor.close()
```

3.3 数据库表和流数据相结合

使用pandas读取购买数据库，生成dataframe，对时间戳进行预处理，并对购买数据特征进行简单分析，查看各个种类数据出现最多的值及次数

```
engine = create_engine('mysql+pymysql://root:czh13935748710.@localhost:3306/hive')
    sql_query = 'select * from buy_data'
    buy_data = pd.read_sql_query(sql_query, engine)


    # 将timestamp转换成datetime【%Y-%m-%d %H:%M:%S】
    def timestamp_datetime(value):
        format = '%Y-%m-%d %H:%M:%S'
        value = time.localtime(value)
        dt = time.strftime(format, value)
        return dt  # str


    # 时间，datetime64[ns]
    buy_data['time'] = pd.to_datetime(buy_data.timestamp.apply(timestamp_datetime))
    buy_data['day'] = buy_data.time.dt.day
    buy_data['hour'] = buy_data.time.dt.hour
    buy_data['minute'] = buy_data.time.dt.minute

    file['描述0'] = '对购买数据进行基础分析\n'
    # 购买数据基本特征
    file['描述1'] = '购买数据有 {} 行 {} 列'.format(buy_data.shape[0], buy_data.shape[1])

    file['描述2'] = '数据中有' + str(len(buy_data['user_id'].unique())) + '位不同的用户' + str(
        len(buy_data['item_id'].unique())) + '件不同的商品\n'

    # 出现频率最高的各类型数据
    file['描述3'] = '下面是各个列值出现次数最高的次数\n'
    temp = {}
    for x in ['user_id', 'item_id', 'category_id', 'type', 'time', 'day', 'hour', 'minute']:
        temp[x] = buy_data[x].value_counts().head().to_json()
    file['购买数据分析'] = temp
```

```
    file['描述4'] = '\n注意到 5000010 号商品有多达 76169 次购买记录，但最多的被购买类型 1511622034 仅有 151 次购买记录，但某个商品应该属于某个类型，我们认为源数据有一定的问题\n'
```

类似的，使用pandas读取流数据库，并将其根据item_id合并到dataframe，形成以item_id为键的所有有效数据的集合，查看各个种类数据出现最多的值及次数

```
engine = create_engine('mysql+pymysql://root:czh13935748710.@localhost:3306/hive')
    sql_query = 'select * from streaming limit 20000000'
    stream_data = pd.read_sql_query(sql_query, engine)

    stream_data.rename(columns={'userId': 'user_id', 'itemId': 'item_id', 'categoryId':
'category_id'}, inplace=True)

    # 将流数据纳入分析
    mergeData = pd.merge(buy_data, stream_data, 'outer', on=['item_id', 'user_id',
'category_id'])

    file['描述6'] = '对流数据进行基础分析\n'
    file['描述7'] = '下面是各个列值出现次数最高的次数\n'
    # 出现频率最高的各类型数据
    temp = {}
    for x in ['ipAddr', 'sessionId', 'date', 'url', 'user_id', 'item_id', 'category_id',
'isSecondKill', 'password',
              'authCode', 'success', 'type', 'time', 'day', 'hour', 'minute']:
        temp[x] = mergeData[x].value_counts().head().to_json()
    file['流数据分析'] = temp
```

3.3.1 商品成交率

对每件商品（item_id）的购买（type==buy）进行计数形成一个新列（buy_count），对每件商品的交互情况进行计数（url!=NaN）形成一个新列（interact_count），商品的成交率定义为（购买次数 / 交互次数），即用上述两列相除，得到所求（deal_rate）生成一列

```
# 成交率
    file['描述8'] = '计算商品的成交率：'
    buy_count = pd.DataFrame(
        mergeData.loc[(mergeData['type'] == 'buy')]
['item_id'].value_counts()).reset_index().rename(
        columns={'index': 'item_id', 'item_id': 'buy_count'})
    mergeData = pd.merge(mergeData, buy_count, 'left', on=['item_id'])
    interact_count = pd.DataFrame(
        mergeData.loc[(mergeData['url'] != 'NaN')]
['item_id'].value_counts()).reset_index().rename(
        columns={'index': 'item_id', 'item_id': 'interact_count'})
    mergeData = pd.merge(mergeData, interact_count, 'left', on=['item_id'])
    mergeData['deal_rate'] = mergeData['buy_count'] / mergeData['interact_count']
    file['商品成交率'] = mergeData[['item_id', 'deal_rate']].head().to_json()
```

3.3.2 热门商品

使用dataframe对总集合依次进行按照购买次数（buy_count）和交互次数（interact_count）的降序排序，得到最高的几件商品（item_id）即为热门商品

```
# 热门商品
    file['描述12'] = '寻找热门商品：'
    file['热门商品'] =
    (mergeData.sort_values(by='buy_count', ascending=False).sort_values(by='interact_count',
ascending=False))[
        'item_id'].value_counts().head().to_json()
```

3.3.3 商品复购率

计算所有商品的复购次数 / 购买次数

```
    # 商品平均复购率
    file['描述10'] = '计算商品的平均复购率：'
    isSecondKill = mergeData['isSecondKill'].value_counts()[0] / mergeData.shape[0]
    file['商品平均复购率'] = isSecondKill
```

4 数据可视化

交易数据——商品类别角度

G7-Homework2

交易数据 流数据 组合数据

商品类别角度 | 商品角度

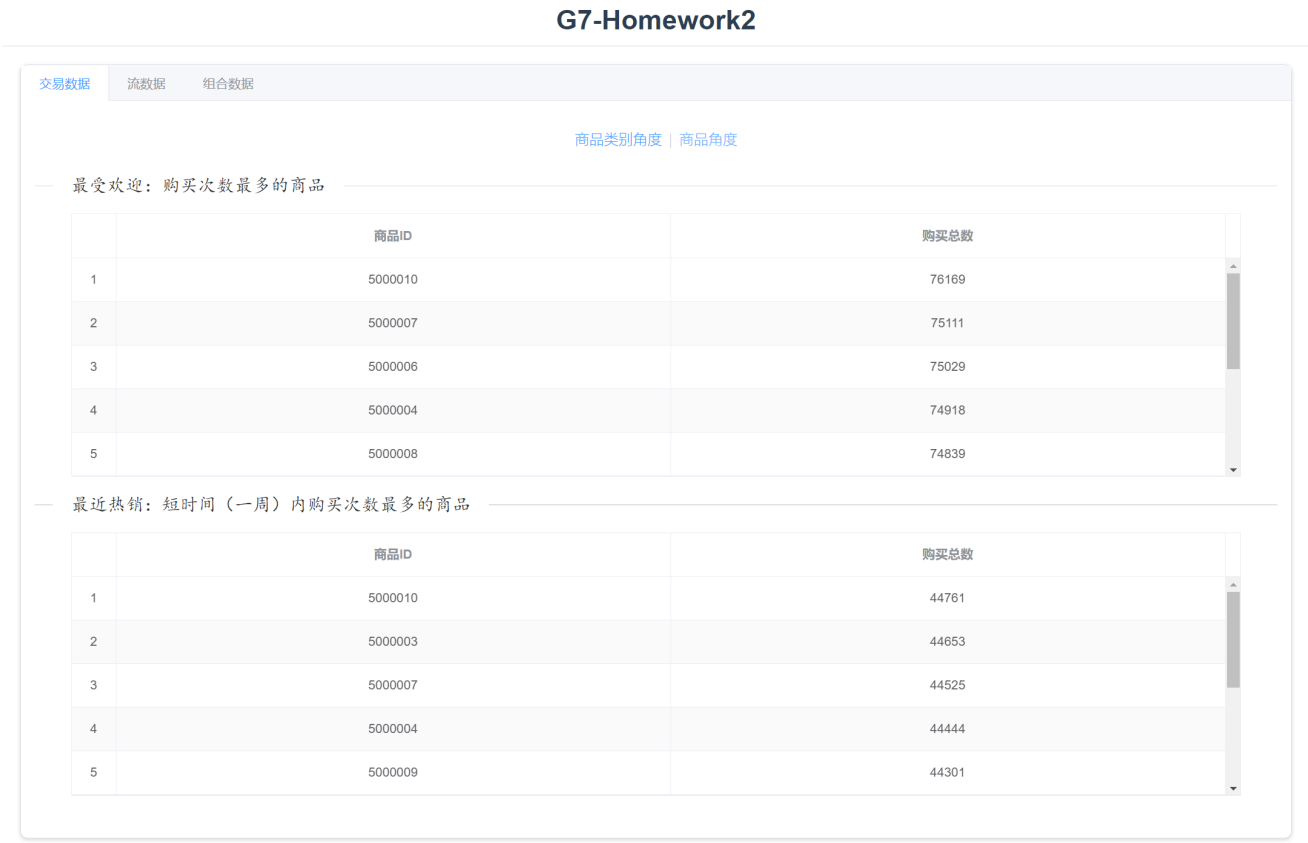
最受欢迎：购买次数最多的商品类别

	商品类别ID	购买总数
1	1511622034	151
2	1511834425	148
3	1511712025	148
4	1511964025	143
5	1511712026	142

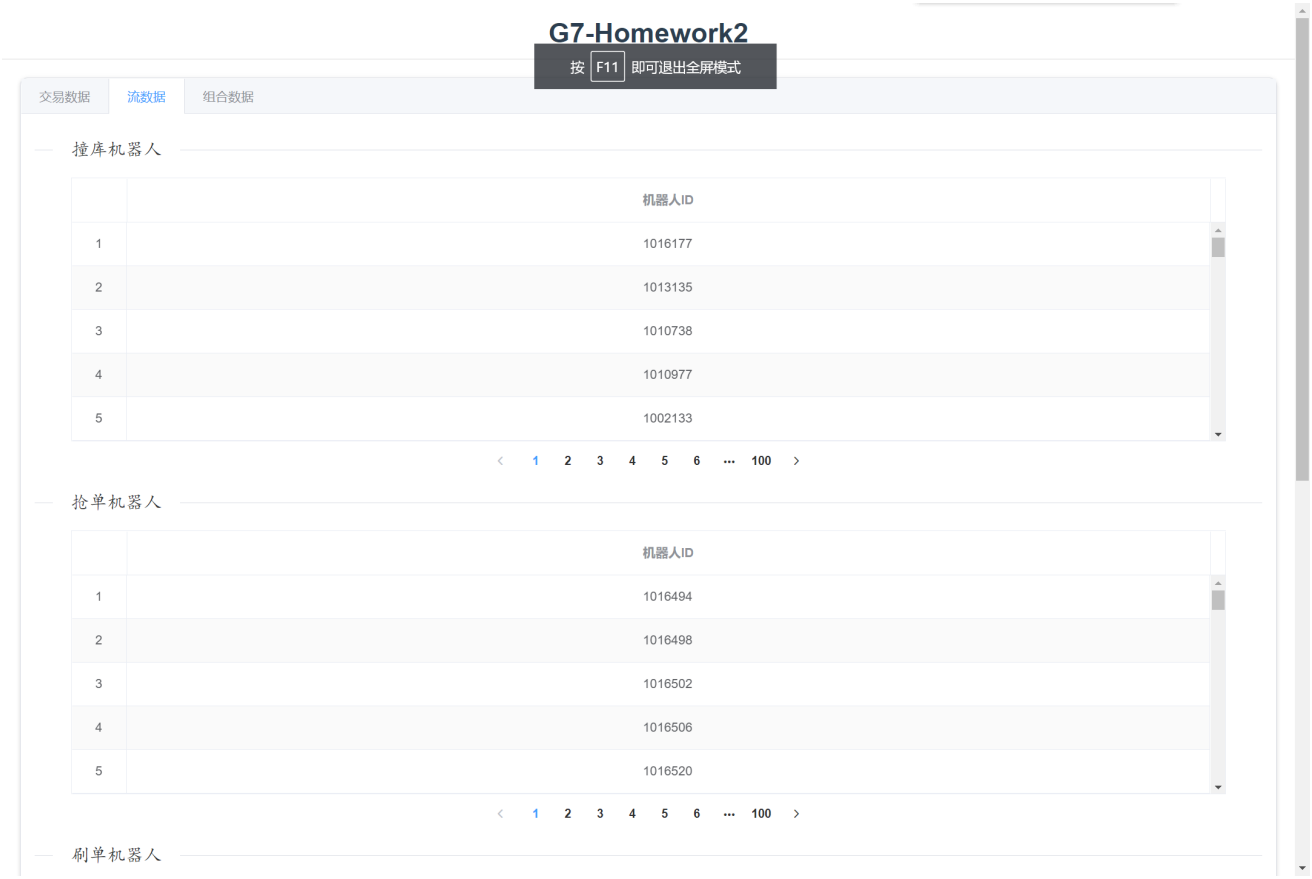
最近热销：短时间内购买次数最多的商品类别

	商品类别ID	购买总数
1	1511834425	106
2	1512115225	101
3	1511622034	100
4	1511712025	97
5	1511791225	94

交易数据——商品角度



流数据——四类机器人



组合数据

G7-Homework2

交易数据 流数据 组合数据

商品成交率：成交次数 / 浏览次数

	商品ID	商品成交率
1	2062663	1
2	2894092	0.5757575758
3	4209472	0.4107142857
4	3310369	0.3504273504
5	2154381	0.0801393728

热门商品：高访问量，高成交量

	热门商品ID	评价数值
1	5000010	90553
2	5000007	89315
3	5000008	89161
4	5000006	89160
5	5000004	88942

商品复购率：复购次数 / 购买次数

	商品ID	复购率