离线数据处理任务书(一)-参考代码

数据抽取

编写Scala工程代码，将MySQL的shtd_store库中表CUSTOMER、NATION、PART、PARTSUPP、REGION、SUPPLIER的数据全量抽取到Hive的ods库中对应表customer，nation，part，partsupp，region，supplier中，将表ORDERS、LINEITEM的数据增量抽取到Hive的ods库中对应表ORDERS，LINEITEM中。

任务一、全量抽取

Task1

1、 抽取shtd_store库中CUSTOMER的全量数据进入Hive的ods库中表customer。字段排序、类型不变，同时添加静态分区，分区字段类型为String，且值为当前比赛日的前一天日期（分区字段格式为yyyyMMdd）。并在hive cli执行show partitions ods.customer命令，将结果截图复制粘贴至对应报告中；

hql

```
show partitions ods.customer;
```

输出结果:

```
hive (default)> show partitions ods.customer;
OK
partition
etldate=20220506
Time taken: 0.837 seconds, Fetched: 1 row(s)
```

数据结果验证

查看mysql表的数据条数

```
select count(*) from shtd_store.CUSTOMER;
```

```
mysql> select count(*) from shtd_store.CUSTOMER;
+----------+
| count(*) |
+----------+
|   150000 |
+----------+
1 row in set (0.08 sec)
```

查看hive表的数据条数

```
select count(*) from ods.customer;
```

```
hive (ods)> select count(*) from ods.customer;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a diff
Query ID = root_20220516232535_660c86c1-5a9a-4040-8e1e-8c4f182c049b
Total 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 mapreduce.job.reduces=<number>
Starting Job = job_1652714304866_0003, Tracking URL = http://master:8088/proxy/application_1652714304866_0003/
Kill Command = /opt/hadoop-2.7.7/bin/hadoop job  -kill job_1652714304866_0003
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-05-16 23:25:43,317 Stage-1 map = 0%,  reduce = 0%
2022-05-16 23:25:48,655 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.86 sec
2022-05-16 23:25:52,867 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.97 sec
MapReduce Total cumulative CPU time: 4 seconds 970 msec
Ended Job = job_1652714304866_0003
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 4.97 sec   HDFS Read: 24204871 HDFS Write: 106 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 970 msec
OK
_c0
150000
Time taken: 25.184 seconds, Fetched: 1 row(s)
hive (ods)>
```

Task2

2、 抽取shtd_store库中NATION的全量数据进入Hive的ods库中表nation。字段排序、类型不变，同时添加静态分区，分区字段类型为String，且值为当前比赛日的前一天日期（分区字段格式为yyyyMMdd）。并在hive cli执行show partitions ods.nation命令，将结果截图复制粘贴至对应报告中；

hql

```
show partitions ods.nation;
```

输出结果:

```
hive (default)> show partitions ods.nation;
OK
partition
etldate=20220506
Time taken: 0.052 seconds, Fetched: 1 row(s)
```

数据验证

查看mysql表的数据条数

```
select count(*) from shtd_store.NATION;
```

```
mysql> select count(*) from shtd_store.NATION;
+----------+
| count(*) |
+----------+
|       25 |
+----------+
1 row in set (0.00 sec)
```

查看hive表的数据条数

```
select count(*) from ods.nation;
```

```
hive (ods)> select count(*) from ods.nation;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a diffe
Query ID = root_20220516232753_547c7770-5fd0-4e21-a6e6-440712030ebc
Total 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 mapreduce.job.reduces=<number>
Starting Job = job_1652714304866_0004, Tracking URL = http://master:8088/proxy/application_1652714304866_0004/
Kill Command = /opt/hadoop-2.7.7/bin/hadoop job  -kill job_1652714304866_0004
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-05-16 23:27:58,267 Stage-1 map = 0%,  reduce = 0%
2022-05-16 23:28:02,453 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.44 sec
2022-05-16 23:28:07,646 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.53 sec
MapReduce Total cumulative CPU time: 3 seconds 530 msec
Ended Job = job_1652714304866_0004
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 3.53 sec   HDFS Read: 10391 HDFS Write: 102 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 530 msec
OK
_c0
25
Time taken: 19.703 seconds, Fetched: 1 row(s)
hive (ods)>
```

Task3

3、 抽取shtd_store库中PART的全量数据进入Hive的ods库中表part。字段排序、类型不变，同时添加静态分区，分区字段类型为String，且值为当前比赛日的前一天日期（分区字段格式为yyyyMMdd）。并在hive cli执行show partitions ods.part命令，将结果截图复制粘贴至对应报告中；

hql

```
show partitions ods.part;
```

输出结果:

```
hive (default)> show partitions ods.part;
OK
partition
etldate=20220506
Time taken: 0.045 seconds, Fetched: 1 row(s)
```

数据验证

查看mysql表的数据条数

```
select count(*) from shtd_store.PART;
```

```
mysql> select count(*) from shtd_store.PART;
+----------+
| count(*) |
+----------+
|   200000 |
+----------+
1 row in set (0.10 sec)
```

查看hive表的数据条数

```
select count(*) from ods.part;
```

```
hive (ods)> select count(*) from ods.part;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different
Query ID = root_20220516232946_787dc32b-fffd-410d-acb1-a38ac5a8de17
Total 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 mapreduce.job.reduces=<number>
Starting Job = job_1652714304866_0005, Tracking URL = http://master:8088/proxy/application_1652714304866_0005/
Kill Command = /opt/hadoop-2.7.7/bin/hadoop job  -kill job_1652714304866_0005
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-05-16 23:29:50,845 Stage-1 map = 0%,  reduce = 0%
2022-05-16 23:29:56,065 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.94 sec
2022-05-16 23:30:01,299 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 5.01 sec
MapReduce Total cumulative CPU time: 5 seconds 10 msec
Ended Job = job_1652714304866_0005
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 5.01 sec   HDFS Read: 23943924 HDFS Write: 106 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 10 msec
OK
_c0
200000
Time taken: 22.582 seconds, Fetched: 1 row(s)
```

Task4

4、 抽取shtd_store库中PARTSUPP的全量数据进入Hive的ods库中表partsupp。字段排序、类型不变，同时添加静态分区，分区字段类型为String，且值为当前比赛日的前一天日期（分区字段格式为yyyyMMdd）。并在hive cli执行show partitions ods.partsupp命令，将结果截图复制粘贴至对应报告中；

hql

```
show partitions ods.partsupp;
```

输出结果:

```
hive (default)> show partitions ods.partsupp;
OK
partition
etldate=20220506
Time taken: 0.045 seconds, Fetched: 1 row(s)
```

数据验证

查看mysql表的数据条数

```
select count(*) from shtd_store.PARTSUPP
```

```
mysql> select count(*) from shtd_store.PARTSUPP;
+----------+
| count(*) |
+----------+
|  1600000 |
+----------+
1 row in set (0.80 sec)
```

查看hive表的数据条数

```
select count(*) from ods.partsupp;
```

```
hive (ods)> select count(*) from ods.partsupp;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a differe
Query ID = root_20220516233200_311e801c-93c0-4a16-baf7-f22864733c90
Total 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 mapreduce.job.reduces=<number>
Starting Job = job_1652714304866_0007, Tracking URL = http://master:8088/proxy/application_1652714304866_0007/
Kill Command = /opt/hadoop-2.7.7/bin/hadoop job  -kill job_1652714304866_0007
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-05-16 23:32:04,998 Stage-1 map = 0%,  reduce = 0%
2022-05-16 23:32:11,245 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 4.28 sec
2022-05-16 23:32:15,413 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.39 sec
MapReduce Total cumulative CPU time: 6 seconds 390 msec
Ended Job = job_1652714304866_0007
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 6.39 sec   HDFS Read: 236381776 HDFS Write: 107 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 390 msec
OK
_c0
1600000
Time taken: 22.462 seconds, Fetched: 1 row(s)
```

Task5

5、 抽取shtd_store库中REGION的全量数据进入Hive的ods库中表region，字段排序、类型不变，同时添加静态分区，分区字段类型为String，且值为当前比赛日的前一天日期（分区字段格式为yyyyMMdd）。并在hive cli执行show partitions ods.region命令，将结果截图复制粘贴至对应报告中；

hql

```
show partitions ods.region;
```

输出结果:

```
hive (default)> show partitions ods.region;
OK
partition
etldate=20220506
Time taken: 0.047 seconds, Fetched: 1 row(s)
```

数据验证

查看mysql表的数据条数

```
select count(*) from shtd_store.REGION;
```

```
mysql> select count(*) from shtd_store.REGION;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)
```

查看hive表的数据条数

```
select count(*) from ods.region;
```

```
hive (ods)> select count(*) from ods.region;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different
Query ID = root_20220516233039_b70e8919-6878-4452-a950-c706066e678b
Total 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 mapreduce.job.reduces=<number>
Starting Job = job_1652714304866_0006, Tracking URL = http://master:8088/proxy/application_1652714304866_0006/
Kill Command = /opt/hadoop-2.7.7/bin/hadoop job  -kill job_1652714304866_0006
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-05-16 23:30:44,219 Stage-1 map = 0%,  reduce = 0%
2022-05-16 23:30:49,448 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.47 sec
2022-05-16 23:30:54,666 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.57 sec
MapReduce Total cumulative CPU time: 3 seconds 570 msec
Ended Job = job_1652714304866_0006
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 3.57 sec   HDFS Read: 8435 HDFS Write: 101 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 570 msec
OK
_c0
5
Time taken: 22.53 seconds, Fetched: 1 row(s)
```

Task6

6、 抽取shtd_store库中SUPPLIER的全量数据进入Hive的ods库中表supplier，字段排序、类型不变，同时添加静态分区，分区字段类型为String，且值为当前比赛日的前一天日期（分区字段格式为yyyyMMdd）。并在hive cli执行show partitions ods.supplier命令，将结果截图复制粘贴至对应报告中；

hql

```
show partitions ods.supplier;
```

输出结果:

```
hive (default)> show partitions ods.supplier;
OK
partition
etldate=20220506
Time taken: 0.047 seconds, Fetched: 1 row(s)
```

数据验证

查看mysql表的数据条数

```
select count(*) from shtd_store.SUPPLIER;
```

```
mysql> select count(*) from shtd_store.SUPPLIER;
+----------+
| count(*) |
+----------+
|    10000 |
+----------+
1 row in set (0.01 sec)
```

查看hive表的数据条数

```
select count(*) from ods.supplier;
```

```
hive (ods)> select count(*) from ods.supplier;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different ex
Query ID = root_20220516233706_dc282955-d504-4749-9e6f-8b7ae9d6f359
Total 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 mapreduce.job.reduces=<number>
Starting Job = job_1652714304866_0008, Tracking URL = http://master:8088/proxy/application_1652714304866_0008/
Kill Command = /opt/hadoop-2.7.7/bin/hadoop job  -kill job_1652714304866_0008
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-05-16 23:37:11,642 Stage-1 map = 0%,  reduce = 0%
2022-05-16 23:37:16,873 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.61 sec
2022-05-16 23:37:22,072 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.85 sec
MapReduce Total cumulative CPU time: 3 seconds 850 msec
Ended Job = job_1652714304866_0008
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 3.85 sec   HDFS Read: 1407800 HDFS Write: 105 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 850 msec
OK
_c0
10000
Time taken: 21.413 seconds, Fetched: 1 row(s)
```

1-6参考代码

```
package spark

import org.apache.spark.sql.SparkSession

import java.time.LocalDate
import java.util.{Calendar, Date, Properties}
```

```scala
/**
 * @author smy
 * @see
 * 编写Scala工程代码，将MySQL的shtd_store库中表CUSTOMER、NATION、PART、
PARTSUPP、REGION、SUPPLIER的数据
 * 全量抽取到Hive的ods库中对应表customer，nation，part，partsupp，region，
supplier中。
 * 字段排序，类型不变，同时添加静态分区，分区字段类型为String，且值为当前日期的
前一天（分区字段格式为yyyyMMdd）。
 * 并在hive cli执行show partitions ods.customer命令，将结果截图复制粘贴至对应
报告中；
 * @version
 * scala 2.11.0
 * spark 2.1.1
 * @see
 * 启用hiveserver2 和metastore服务
 * nohup hive --service metastore  > /usr/local/src/hive/metastore.log 2>1
&
 * nohup hive --service hiveserver2  > /usr/local/src/hive/HS2.log 2>1 &
 */
object FullStaticExtraction {
  def main(args: Array[String]): Unit = {
    val warehouse: String = s"hdfs://master:9000/user/hive/warehouse"
    val sparkSession: SparkSession = SparkSession.builder()
      .config("spark.sql.shuffle.partitions", 1000) //设置shuffle的分区数
      .config("spark.sql.warehouse.dir", warehouse) //指定hive仓库地址
      .config("hive.metastore.uris", "thrift://master:9083") // 指定
metastore地址
      .appName("FullStaticExtraction") //设置程序名称
      .enableHiveSupport() // 连接hive
      .getOrCreate()
    val MYSQLDBURL: String = s"jdbc:mysql://master:3306/shtd_store?
useUnicode=true&characterEncoding=utf-8" // mysql url地址
    val properties: Properties = new Properties()
    properties.put("user", "root") //用户名
    properties.put("password", "123456") // 密码
    properties.put("driver", "com.mysql.jdbc.Driver") // 驱动名称
    // 测试当前连接的hive metastore是否正确
    sparkSession.sql("show databases").show()
    val array: Array[String] = Array("CUSTOMER", "NATION", "PART",
"PARTSUPP", "REGION", "SUPPLIER")
    array.foreach(x => {
      // 获取当前日期的前一天
      val theDay: String = LocalDate.now().plusDays(-1).toString.replace("-
", "")
      println(s"=============当日时间
为:${theDay}==========================")
      sparkSession.read.jdbc(MYSQLDBURL, x, properties).createTempView(x)
//读取mysql的数据，创建临时数据表
```

```
      val loadDate: String = s"insert overwrite table ods.${x} partition
(etldate='${theDay}') select * from ${x}" // 将mysql的数据加载到hive表中
      val selectLimitSql: String = s"show partitions ods.${x}" // 查看hive
表的分区
      sparkSession.sql(loadDate)
      sparkSession.sql(selectLimitSql).show()
    })
  }
}
```

任务二、增量抽取

准备数据代码:

```
package spark

import org.apache.spark.sql.functions.{col, desc}
import org.apache.spark.sql.{DataFrame, Dataset, Row, SparkSession}
import java.util.Properties

/**
 * @see
 * 抽取数据用于任务二测试
 * @author smy
 * @see
 * 启用hiveserver2 和metastore服务
 * nohup hive --service metastore  > /usr/local/src/hive/metastore.log 2>1
&
 * nohup hive --service hiveserver2  > /usr/local/src/hive/HS2.log 2>1 &
 */
object ExtractionData {
  def main(args: Array[String]): Unit = {
    val warehouse: String = s"hdfs://master:9000/user/hive/warehouse"
    val sparkSession: SparkSession = SparkSession.builder()
      .config("spark.sql.warehouse.dir", warehouse) //指定hive仓库地址
      .config("hive.metastore.uris", "thrift://master:9083") // 指定
metastore地址
      .config("spark.sql.shuffle.partitions", 1000) //设置shuffle的分区数
      .appName("ExtractionData") //设置程序名称
      .enableHiveSupport() // 连接hive
      .getOrCreate()
    val MYSQLDBURL: String = s"jdbc:mysql://master:3306/shtd_store?
useUnicode=true&characterEncoding=utf-8" // mysql url地址
    val properties: Properties = new Properties()
    properties.put("user", "root") //用户名
    properties.put("password", "123456") // 密码
    properties.put("driver", "com.mysql.jdbc.Driver") // 驱动名称
    val ORDERS: Dataset[Row] = sparkSession.read.jdbc(MYSQLDBURL, "ORDERS",
properties)
      .withColumn("ORDERKEY", col("ORDERKEY").cast("int"))
      .orderBy(col("ORDERKEY"))
```

```
      .limit(50000)
    ORDERS.createTempView("ORDERS")
    val LINEITEM: Dataset[Row] = sparkSession.read.jdbc(MYSQLDBURL,
"LINEITEM", properties)
      .withColumn("ORDERKEY", col("ORDERKEY").cast("int"))
      .orderBy(col("ORDERKEY"))
      .limit(2000)
    LINEITEM.createTempView("LINEITEM")
    val theDay: String = "19971201"
    println(s"=============设置当日时间
为:${theDay}=========================")
    sparkSession.sql(s"insert overwrite table ods.orders partition
(etldate='${theDay}') select * from ORDERS")
    sparkSession.sql(s"insert overwrite table ods.lineitem partition
(etldate='${theDay}') select * from LINEITEM")
    sparkSession.sql("select * from ods.orders")
      .withColumn("orderkey", col("orderkey").cast("int"))
      .orderBy(desc("orderkey"))
      .show()
    sparkSession.sql("select * from ods.lineitem")
      .withColumn("orderkey", col("orderkey").cast("int"))
      .orderBy(desc("orderkey"))
      .show()
  }
}
```

验证hive表数据条数

ods.lineitem

```
select count(*) from ods.lineitem;
```

```
hive (ods)> select count(*) from ods.lineitem;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different
Query ID = root_20220516234625_8508c3f4-fea9-45db-aeb4-3e2368983bbb
Total 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 mapreduce.job.reduces=<number>
Starting Job = job_1652714304866_0012, Tracking URL = http://master:8088/proxy/application_1652714304866_0012/
Kill Command = /opt/hadoop-2.7.7/bin/hadoop job  -kill job_1652714304866_0012
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-05-16 23:46:30,692 Stage-1 map = 0%,  reduce = 0%
2022-05-16 23:46:35,932 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.68 sec
2022-05-16 23:46:40,091 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.75 sec
MapReduce Total cumulative CPU time: 3 seconds 750 msec
Ended Job = job_1652714304866_0012
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 3.75 sec   HDFS Read: 261165 HDFS Write: 104 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 750 msec
OK
_c0
2000
Time taken: 21.944 seconds, Fetched: 1 row(s)
```

查看最大值

```
from ods.lineitem
select orderkey order by cast(orderkey as INT) desc limit 10;
```

```
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 4.51 sec   HDFS Read: 260396 HDFS Write: 287 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 510 msec
OK
orderkey
5992002
5992002
5992002
5992001
5992001
5992001
5992000
5991975
5991975
5991975
Time taken: 21.476 seconds, Fetched: 10 row(s)
hive (ods)>
```

ods.orders

```
select count(*) from ods.orders;
```

```
hive (ods)> select count(*) from ods.orders;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different
Query ID = root_20220516235606_1a30cb2f-ad03-4da9-ab9b-da91036197e4
Total 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 mapreduce.job.reduces=<number>
Starting Job = job_1652714304866_0016, Tracking URL = http://master:8088/proxy/application_1652714304866_0016/
Kill Command = /opt/hadoop-2.7.7/bin/hadoop job  -kill job_1652714304866_0016
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-05-16 23:56:12,937 Stage-1 map = 0%,  reduce = 0%
2022-05-16 23:56:18,175 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.37 sec
2022-05-16 23:56:23,380 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.53 sec
MapReduce Total cumulative CPU time: 4 seconds 530 msec
Ended Job = job_1652714304866_0016
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 4.53 sec   HDFS Read: 5714959 HDFS Write: 105 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 530 msec
OK
_c0
50000
Time taken: 24.191 seconds, Fetched: 1 row(s)
hive (ods)>
```

查看最大值

```
from ods.orders
select orderkey order by cast(orderkey as INT) desc limit 10;
```

```
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 5.74 sec   HDFS Read: 5714329 HDFS Write: 277 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 740 msec
OK
orderkey
200000
199975
199974
199973
199972
199971
199970
199969
199968
199943
Time taken: 24.909 seconds, Fetched: 10 row(s)
hive (ods)>
```

Task1

1、 抽取shtd_store库中ORDERS的增量数据进入Hive的ods库中表orders，要求只取1997年12月1日及之后的数据（包括1997年12月1日），根据ORDERS表中ORDERKEY作为增量字段（提示：对比MySQL和Hive中的表的ORDERKEY大小），只将新增的数据抽入，字段类型不变，同时添加动态分区，分区字段类型为String，且值为ORDERDATE字段的内容（ORDERDATE的格式为yyyy-MM-dd，分区字段格式为yyyyMMdd），。并在hive cli执行select count(distinct(etldate)) from ods.orders命令，将结果截图复制粘贴至对应报告中；

hql

```
select count(distinct(etldate)) from ods.orders;
```

输出结果

```
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 7.18 sec   HDFS Read: 22545113 HDFS Write: 103 SUCCESS
Total MapReduce CPU Time Spent: 7 seconds 180 msec
OK
_c0
245
Time taken: 22.242 seconds, Fetched: 1 row(s)
```

数据验证

查看mysql表符合题意的数据条数

```
select count(*) from shtd_store.ORDERS where cast(ORDERKEY as signed) > 200000 and cast(ORDERDATE as date) >= '1997-12-01';
```

```
mysql> select count(*) from shtd_store.ORDERS where cast(ORDERKEY as signed) > 200000 and cast(ORDERDATE as date) >= '1997-12-01';
+----------+
| count(*) |
+----------+
|   147894 |
+----------+
1 row in set (1.23 sec)
```

查看hive表的数据条数

```
select count(*) from ods.orders;
```

```
hive (ods)> select count(*) from ods.orders;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different
Query ID = root_20220517010000_59609e51-054e-47cd-bb01-a44845e08873
Total 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 mapreduce.job.reduces=<number>
Starting Job = job_1652714304866_0030, Tracking URL = http://master:8088/proxy/application_1652714304866_0030/
Kill Command = /opt/hadoop-2.7.7/bin/hadoop job  -kill job_1652714304866_0030
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-05-17 01:00:05,237 Stage-1 map = 0%,  reduce = 0%
2022-05-17 01:00:11,490 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 4.89 sec
2022-05-17 01:00:16,692 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 7.03 sec
MapReduce Total cumulative CPU time: 7 seconds 30 msec
Ended Job = job_1652714304866_0030
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 7.03 sec   HDFS Read: 22528956 HDFS Write: 106 SUCCESS
Total MapReduce CPU Time Spent: 7 seconds 30 msec
OK
_c0
197894
Time taken: 23.647 seconds, Fetched: 1 row(s)
hive (ods)>
```

参考代码:

```
package training_one.data_extraction.spark

import org.apache.spark.sql.{DataFrame, SaveMode, SparkSession}
import org.apache.spark.sql.functions.{col, date_format, max}

import java.util.Properties

/**
 * @author smy
 * @see
 * 抽取shtd_store库中ORDERS的增量数据进入Hive的ods库中表orders，
 * 要求只取1997年12月1日及之后的数据（包括1997年12月1日），
 * 根据ORDERS表中ORDERKEY作为增量字段（提示：对比MySQL和Hive中的表的ORDERKEY大小），
 * 只将新增的数据抽入，字段类型不变，同时添加动态分区，分区字段类型为String，
 * 且值为ORDERDATE字段的内容（ORDERDATE的格式为yyyy-MM-dd，分区字段格式为yyyyMMdd）。
 * 并在hive cli执行select count(distinct(etldate))from ods.orders命令，
 * 将结果截图复制粘贴至对应报告中；
 * @version
 * scala 2.11.0
 * spark 2.1.1
 * @see
 * 启用hiveserver2 和metastore服务
 * nohup hive --service metastore  > /usr/local/src/hive/metastore.log 2>1 &
 * nohup hive --service hiveserver2  > /usr/local/src/hive/HS2.log 2>1 &
 */
object IncrementalDynamicExtraction {
```

```
  def main(args: Array[String]): Unit = {
    val warehouse: String = s"hdfs://master:9000/user/hive/warehouse"
    val sparkSession: SparkSession = SparkSession.builder()
      .config("spark.sql.warehouse.dir", warehouse) //指定hive仓库地址
      .config("hive.metastore.uris", "thrift://master:9083") // 指定
metastore地址
      .config("spark.sql.shuffle.partitions", 1000) //设置shuffle的分区数
      .config("hive.exec.dynamic.partition", value = true) // 开启动态分区
      .config("hive.exec.dynamic.partition.mode", "nonstrict") // 设置分区
模式为nonstrict
      .config("hive.exec.max.dynamic.partitions", value = 10000) //设置动态
最大分区数
      .appName("IncrementalDynamicExtraction") //设置程序名称
      .enableHiveSupport() // 连接hive
      .getOrCreate()
    // 设置日志等级,默认INFO
//    sparkSession.sparkContext.setLogLevel("ERROR")
    // 查看当前是否开启动态分区
    sparkSession.sql("set hive.exec.dynamic.partition").show(false)
    // 查看当前分区的模式
    sparkSession.sql("set hive.exec.dynamic.partition.mode").show(false)
    val MYSQLDBURL: String = s"jdbc:mysql://master:3306/shtd_store?
useUnicode=true&characterEncoding=utf-8" // mysql url地址
    val properties: Properties = new Properties()
    properties.put("user", "root") //用户名
    properties.put("password", "123456") // 密码
    properties.put("driver", "com.mysql.jdbc.Driver") // 驱动名称
    // 获取ods.orders表里面ORDERKEY的最大值 orderKeyMax
    val orderKeyMax: Int = sparkSession.sql("from ods.orders select
orderkey order by cast(orderkey as INT) desc limit 1")
      .collect()(0).get(0).toString.toInt
    println(s"=====================ods.orders-ORDERKEY最大值
为:${orderKeyMax}================================")
    // 读取mysql数据
    sparkSession.read.jdbc(MYSQLDBURL, "ORDERS",
properties).createTempView("mysql_orders") //读取order表的数据
    // 查询到需要的数据
    sparkSession.sql(
      s"""
         |from mysql_orders
         |select *, date_format(orderdate, 'yyyyMMdd') as etldate
         |where cast(orderkey as INT) > ${orderKeyMax}
         |  and cast(orderdate as date) >= "1997-12-01"
         |""".stripMargin).createTempView("v_orders")
    // 查看数据条数
    sparkSession.sql("select count(*) from v_orders").show(false)
    // 增量使用into
    sparkSession.sql(
      """
        |insert into table ods.orders partition(etldate)
        |select * from v_orders
```

```
        |""".stripMargin)
    sparkSession.sql("select count(distinct(etldate))from
ods.orders").show(false)
  }
}
```

Task2

2、抽取shtd_store库中LINEITEM的增量数据进入Hive的ods库中表lineitem，根据LINEITEM表中orderkey作为增量字段，只将新增的数据抽入，字段类型不变，同时添加静态分区，分区字段类型为String，且值为当前比赛日的前一天日期（分区字段格式为yyyyMMdd）。并在hive cli执行show partitions ods.lineitem命令，将结果截图复制粘贴至对应报告中。

hql

```
show partitions ods.lineitem;
```

输出结果:

```
hive (default)> show partitions ods.lineitem;
OK
partition
etldate=19971201
etldate=20220506
Time taken: 0.039 seconds, Fetched: 2 row(s)
```

数据验证

查看mysql表符合题意的数据条数

```
 select count(*) from shtd_store.LINEITEM where cast(orderkey as signed) >
5992002;
```

```
mysql> select count(*) from shtd_store.LINEITEM where cast(orderkey as signed) > 5992002;
+----------+
| count(*) |
+----------+
|     8000 |
+----------+
1 row in set (0.01 sec)
```

查看hive表的数据条数

```
select count(*) from ods.lineitem;
```

```
hive (ods)> select count(*) from ods.lineitem;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different
Query ID = root_20220517012405_59a190e7-a4f2-4c9e-8ce8-0176d8ff8abd
Total 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 mapreduce.job.reduces=<number>
Starting Job = job_1652714304866_0041, Tracking URL = http://master:8088/proxy/application_1652714304866_0041/
Kill Command = /opt/hadoop-2.7.7/bin/hadoop job  -kill job_1652714304866_0041
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-05-17 01:24:10,832 Stage-1 map = 0%,  reduce = 0%
2022-05-17 01:24:15,072 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.72 sec
2022-05-17 01:24:20,258 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.66 sec
MapReduce Total cumulative CPU time: 4 seconds 660 msec
Ended Job = job_1652714304866_0041
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 4.66 sec   HDFS Read: 1266351 HDFS Write: 105 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 660 msec
OK
_c0
10000
Time taken: 22.317 seconds, Fetched: 1 row(s)
hive (ods)>
```

参考代码:

```scala
package training_one.data_extraction.spark

import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions.{col, max}

import java.time.LocalDate
import java.util.Properties

/**
 * @author smy
 * @see
 * 抽取shtd_store库中LINEITEM的增量数据进入Hive的ods库中表lineitem,
 * 根据LINEITEM表中orderkey作为增量字段,
 * 只将新增的数据抽入，字段类型不变，同时添加静态分区，分区字段类型为String,
 * 且值为当前比赛日的前一天日期（分区字段格式为yyyyMMdd）。
 * 并在hive cli执行show partitions ods.lineitem命令,
 * 将结果截图复制粘贴至对应报告中。
 * @version
 * scala 2.11.0
 * spark 2.1.1
 * @see
 * 启用hiveserver2 和metastore服务
 * nohup hive --service metastore  > /usr/local/src/hive/metastore.log 2>1
&
 * nohup hive --service hiveserver2  > /usr/local/src/hive/HS2.log 2>1 &
 */
object IncrementalStaticExtraction {
  def main(args: Array[String]): Unit = {
```

```
    val warehouse: String = "hdfs://master:9000/user/hive/warehouse"
    val sparkSession: SparkSession = SparkSession.builder()
      .config("spark.sql.shuffle.partitions", 1000) //设置shuffle的分区数
      .config("spark.sql.warehouse.dir", warehouse) //指定hive仓库地址
      .config("hive.metastore.uris", "thrift://master:9083") // 指定
metastore地址
      .appName("IncrementalStaticExtraction") //设置程序名称
      .enableHiveSupport() // 连接hive
      .getOrCreate()
    // mysql url地址
    val MYSQLDBURL: String = "jdbc:mysql://master:3306/shtd_store?
useUnicode=true&characterEncoding=utf-8"
    val properties: Properties = new Properties()
    properties.put("user", "root") //用户名
    properties.put("password", "123456") // 密码
    properties.put("driver", "com.mysql.jdbc.Driver") // 驱动名称
    val array: Array[String] = Array("LINEITEM")
    array.foreach(x => {
      // 获取当日日期的前一天
      val theDay: String = LocalDate.now().plusDays(-1).toString.replace("-
", "")
      // 获取ods.lineitem表里面ORDERKEY的最大值 orderKeyMax
      val orderKeyMax: Int = sparkSession.sql("from ods.lineitem select
orderkey order by cast(orderkey as INT) desc limit 1")
        .collect()(0).get(0).toString.toInt
      println(s"====================ods.lineitem-ORDERKEY最大值
为:${orderKeyMax}==================================")
      println(s"=============当日日期的前一
天:${theDay}=========================")
      sparkSession.read.jdbc(MYSQLDBURL, x,
properties).createTempView(s"mysql_${x}")
      sparkSession.sql(
        s"""
          |from mysql_${x}
          |select * where cast(orderkey as INT) > ${orderKeyMax}
          |""".stripMargin).createTempView(s"v_${x}")
      val loadDate: String = s"insert overwrite table ods.${x} partition
(etldate='${theDay}') select * from v_${x}" // 将mysql的数据加载到hive表中
      sparkSession.sql(s"select count(*) from v_${x}").show(false)
      sparkSession.sql(loadDate)
    })
  }
}
```

数据清洗

编写Scala工程代码，将ods库中相应表数据全量抽取到Hive的dwd库中对应表中。表中有涉及到timestamp类型的，均要求按照yyyy-MM-dd HH:mm:ss，不记录毫秒数，若原数据中只有年月日，则在时分秒的位置添加00:00:00，添加之后使其符合yyyy-MM-dd HH:mm:ss。

Task1

1、 将ods库中customer表数据抽取到dwd库中dim_customer的分区表，分区字段为etldate且值与ods库的相对应表该值相等，并添加dwd_insert_user、dwd_insert_time、dwd_modify_user、dwd_modify_time四列,其中dwd_insert_user、dwd_modify_user均填写“user1”，dwd_insert_time、dwd_modify_time均填写操作时间，并进行数据类型转换。在hive cli中按照custkey顺序排序，查询dim_customer前1条数据，将结果内容复制粘贴至对应报告中；

sql

```
select * from dwd.dim_customer order by custkey limit 1;
```

输出结果:

```
1       Customer#000000001      IVhzIApeRb ot,c,E       15      25-989-741-
2988 711.56  BUILDING        to the even, regular platelets. regular,
ironic epitaphs nag e  user1   2022-04-26 00:27:31     user1   2022-04-26
00:27:31     20220424
```

Task2

2、 将ods库中part表数据抽取到dwd库中dim_part的分区表，分区字段为etldate且值与ods库的相对应表该值相等，并添加dwd_insert_user、dwd_insert_time、dwd_modify_user、dwd_modify_time四列,其中dwd_insert_user、dwd_modify_user均填写“user1”，dwd_insert_time、dwd_modify_time均填写操作时间，并进行数据类型转换。在hive cli中按照partkey顺序排序，查询dim_part前1条数据，将结果内容复制粘贴至对应报告中；

sql

```
select * from dwd.dim_part order by partkey limit 1;
```

输出结果:

```
1       goldenrod lavender spring chocolate lace        Manufacturer#1
Brand#13        PROMO BURNISHED COPPER  7       JUMBO PKG       901.00  ly.
slyly ironi user1   2022-04-26 00:39:31     user1   2022-04-26 00:39:31
20220424
```

Task3

3、 将ods库中nation表数据抽取到dwd库中dim_nation的分区表，分区字段为etldate且值与ods库的相对应表该值相等，并添加dwd_insert_user、dwd_insert_time、dwd_modify_user、dwd_modify_time四列，其中dwd_insert_user、dwd_modify_user均填写“user1”，dwd_insert_time、dwd_modify_time均填写操作时间，并进行数据类型转换。在hive cli中按照nationkey顺序排序，查询dim_nation前1条数据，将结果内容复制粘贴至对应报告中；

sql

```
select * from dwd.dim_nation order by nationkey limit 1;
```

输出结果:

```
0       ALGERIA 0        haggle. carefully final deposits detect slyly agai
    user1   2022-04-26 00:39:34     user1   2022-04-26 00:39:34
20220424
```

Task4

4、 将ods库中region表数据抽取到dwd库中dim_region的分区表，分区字段为etldate且值与ods库的相对应表该值相等，并添加dwd_insert_user、dwd_insert_time、dwd_modify_user、dwd_modify_time四列，其中 dwd_insert_user、dwd_modify_user均填写“user1”，dwd_insert_time、dwd_modify_time均填写操作时间，并进行数据类型转换。在hive cli中按照regionkey顺序排序，查询dim_region表前1条数据，将结果内容复制粘贴至对应报告中；

sql

```
select * from dwd.dim_region order by  regionkey limit 1;
```

输出结果:

```
0       AFRICA  lar deposits. blithely final packages cajole. regular
waters are final requests. regular accounts are according to      user1
2022-04-26 00:39:35     user1   2022-04-26 00:39:35     20220424
```

Task5

5、 将ods库中orders表数据抽取到dwd库中fact_orders的分区表，分区字段为etldate且值与ods库的相对应表该值相等，并添加dwd_insert_user、dwd_insert_time、dwd_modify_user、dwd_modify_time四列，其中dwd_insert_user、dwd_modify_user均填写“user1”，dwd_insert_time、dwd_modify_time均填写操作时间，并进行数据类型转换。在执行hive cli执行select count(distinct(etldate)) from dwd.fact_orders命令，将结果内容复制粘贴至对应报告中；

sql

```
select count(distinct(etldate )) from dwd.fact_orders;
```

输出结果:

```
245
```

查验数据条数

dwd.fact_orders

```
from dwd.fact_orders select count(*);
```

```
hive (ods)> from dwd.fact_orders select count(*);
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different
Query ID = root_20220517014043_a8149070-d1fa-4160-a81e-434f49f17da8
Total 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 mapreduce.job.reduces=<number>
Starting Job = job_1652714304866_0053, Tracking URL = http://master:8088/proxy/application_1652714304866_0053/
Kill Command = /opt/hadoop-2.7.7/bin/hadoop job  -kill job_1652714304866_0053
Hadoop job information for Stage-1: number of mappers: 2; number of reducers: 1
2022-05-17 01:40:49,294 Stage-1 map = 0%,  reduce = 0%
2022-05-17 01:41:02,141 Stage-1 map = 50%,  reduce = 0%, Cumulative CPU 13.14 sec
2022-05-17 01:41:05,259 Stage-1 map = 73%,  reduce = 0%, Cumulative CPU 30.45 sec
2022-05-17 01:41:08,369 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 33.14 sec
2022-05-17 01:41:09,406 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 35.32 sec
MapReduce Total cumulative CPU time: 35 seconds 320 msec
Ended Job = job_1652714304866_0053
MapReduce Jobs Launched:
Stage-Stage-1: Map: 2  Reduce: 1   Cumulative CPU: 35.32 sec   HDFS Read: 12560768 HDFS Write: 106 SUCCESS
Total MapReduce CPU Time Spent: 35 seconds 320 msec
OK
_c0
197894
Time taken: 33.272 seconds, Fetched: 1 row(s)
```

参考代码:

```
package spark

import org.apache.spark.sql.{DataFrame, SaveMode, SparkSession}
import org.apache.spark.sql.functions.{col, current_timestamp, date_format,
lit}

import java.util.Properties

/**
 * @author smy
 * @see
 * 1、编写Scala工程代码，将ods库中表customer,part，nation，region，orders
 * 全量抽取到Hive的dwd库中的dim_customer，dim_part,dim_nation，dim_region，
fact_orders对应表中，
 * 分区字段为etldate且值与ods库的相对应表该值相等，
 * 并添加dwd_insert_user、dwd_insert_time、dwd_modify_user、dwd_modify_time
四列，
 * 其中dwd_insert_user、dwd_modify_user均填写“user1”，dwd_insert_time、
dwd_modify_time均填写当前操作时间。
 * 表中有涉及到timestamp类型的，均要求按照yyyy-MM-dd HH:mm:ss，不记录毫秒数，
 * 若原数据中只有年月日，则在时分秒的位置添加00:00:00，添加之后使其符合yyyy-MM-
dd HH:mm:ss。
 **在hive cli中按照custkey顺序排序，查询dim_customer前1条数据，将结果内容复制
粘贴至对应报告中；**
 **在hive cli中按照suppkey顺序排序，查询dim_supplier前1条数据，将结果内容复制
粘贴至对应报告中；**
```

```
 **在hive cli中按照nationkey顺序排序，查询dim_nation前1条数据，将结果内容复制粘贴至对应报告中；**
 **在hive cli中按照regionkey顺序排序，查询dim_region表前1条数据，将结果内容复制粘贴至对应报告中；**
 **在执行hive cli执行select count(distinct(etldate)) from dwd.fact_orders命令，将结果内容复制粘贴至对应报告中；**
 * @version
 * scala 2.11.0
 * spark 2.1.1
 */
object CleanTask1 {
  def main(args: Array[String]): Unit = {
    val warehouse: String = s"hdfs://master:9000/user/hive/warehouse"
    val sparkSession: SparkSession = SparkSession.builder()
      .config("spark.sql.warehouse.dir", warehouse) //指定hive仓库地址
      .config("hive.metastore.uris", "thrift://master:9083") // 指定metastore地址
      .config("spark.sql.shuffle.partitions", 1000) //设置shuffle的分区
      .appName("CleanTask1") //设置程序名称
      .enableHiveSupport() // 连接hive
      .getOrCreate()
    val MYSQLDBURL: String = s"jdbc:mysql://master:3306/shtd_store?useUnicode=true&characterEncoding=utf-8" // mysql url地址
    val properties: Properties = new Properties()
    properties.put("user", "root") //用户名
    properties.put("password", "123456") // 密码
    properties.put("driver", "com.mysql.jdbc.Driver") // 驱动名称
    val array: Array[String] = Array("customer","part","nation", "region", "orders")
    array.foreach(x=>{
      val dataFrame: DataFrame = sparkSession.sql(s"select * from ods.${x}")
        .withColumn("dwd_insert_user", lit("user1")) // 新增一列dwd_insert_user数据，值为user1
        .withColumn("dwd_insert_time", lit(date_format(current_timestamp(), "yyyy-MM-dd HH:mm:ss")).cast("timestamp")) // 新增一列，获取当前日期
        .withColumn("dwd_modify_user", lit("user1"))
        .withColumn("dwd_modify_time", lit(date_format(current_timestamp(), "yyyy-MM-dd HH:mm:ss")).cast("timestamp"))
      if (x.trim.toLowerCase().equals("orders")){
        dataFrame
          .withColumn("orderdate", col("orderdate").cast("timestamp"))
          .write.mode(SaveMode.Overwrite)
          .partitionBy("etldate") // 设置分区字段
          .saveAsTable(s"dwd.fact_${x}") // 保存表

 println(s"======================fact_${x}=========================================")
        sparkSession.sql(s"select count(distinct(etldate)) from dwd.fact_${x}").show()
      }else{
```

```
          dataFrame.write.mode(SaveMode.Overwrite)
            .partitionBy("etldate") // 设置分区字段
            .saveAsTable(s"dwd.dim_${x}") // 保存表
        }
      })
    }
}
```

Task6

6、 待任务5完成以后，需删除ods.orders中的分区，仅保留最近的三个分区。并在hive cli执行show partitions ods.orders命令，将结果截图粘贴至对应报告中；

sql

```
show partitions ods.orders;
```

输出结果:

```
hive (default)> show partitions ods.orders;
OK
partition
etldate=19980731
etldate=19980801
etldate=19980802
Time taken: 0.047 seconds, Fetched: 3 row(s)
```

查验数据条数

ods.orders

```
from ods.orders select count(*);
```

```
hive (ods)> from ods.orders select count(*);
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different
Query ID = root_20220517014332_2a84e420-5d04-46c9-91ad-d55ee1968816
Total 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 mapreduce.job.reduces=<number>
Starting Job = job_1652714304866_0055, Tracking URL = http://master:8088/proxy/application_1652714304866_0055/
Kill Command = /opt/hadoop-2.7.7/bin/hadoop job  -kill job_1652714304866_0055
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-05-17 01:43:37,488 Stage-1 map = 0%,  reduce = 0%
2022-05-17 01:43:42,704 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.87 sec
2022-05-17 01:43:47,904 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.29 sec
MapReduce Total cumulative CPU time: 4 seconds 290 msec
Ended Job = job_1652714304866_0055
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 4.29 sec   HDFS Read: 209649 HDFS Write: 104 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 290 msec
OK
_c0
1752
Time taken: 22.273 seconds, Fetched: 1 row(s)
hive (ods)>
```

参考代码:

```
package spark

import org.apache.spark.sql.{Row, SparkSession}
import org.apache.spark.sql.functions.{array, col, date_format, desc,
from_unixtime, unix_timestamp}

import java.util.Properties

/**
 * @author smy
 * @see
 * 待上述任务完成以后，需删除ods.orders中的分区，仅保留最近的三个分区。
 * 并在hive cli执行show partitions ods.orders命令，将结果截图粘贴至对应报告
中；
 * @version spark 2.1.1
 *          scala 2.11.0
 * @see
 * 启用hiveserver2 和metastore服务
 * nohup hive --service metastore  > /usr/local/src/hive/metastore.log 2>1
&
 * nohup hive --service hiveserver2  > /usr/local/src/hive/HS2.log 2>1 &
 */
object CleanTask2 {
  def main(args: Array[String]): Unit = {
    val warehouse: String = s"hdfs://master:9000/user/hive/warehouse"
    val sparkSession: SparkSession = SparkSession.builder()
      .config("spark.sql.warehouse.dir", warehouse) //指定hive仓库地址
      .config("hive.metastore.uris", "thrift://master:9083") // 指定
metastore地址
      .config("spark.sql.shuffle.partitions", 1000) //设置shuffle的分区数
      .appName("CleanTask2") //设置程序名称
      .enableHiveSupport() // 连接hive
      .getOrCreate()
    val MYSQLDBURL: String = s"jdbc:mysql://master:3306/shtd_store?
useUnicode=true&characterEncoding=utf-8" // mysql url地址
    val properties: Properties = new Properties()
    properties.put("user", "root") //用户名
    properties.put("password", "123456") // 密码
    properties.put("driver", "com.mysql.jdbc.Driver") // 驱动名称
    val rows: Array[Row] = sparkSession.sql("select etldate from
ods.ORDERS")
      .dropDuplicates("etldate") // 含参数去重
      .withColumn("etldate", from_unixtime(unix_timestamp(col("etldate"),
"yyyyMMdd"), "yyyy-MM-dd"))
      .sort(desc("etldate"))
      .withColumn("etldate", date_format(col("etldate"), "yyyyMMdd"))
      .collect() // 将排序后的数据转换为
    for (x <- 3 until (rows.length)) {
      val value: Any = rows(x).get(0)
```

```
 println(s"======================${value}================================="
)
      sparkSession.sql(s"alter table ods.orders drop if exists
partition(etldate=${value})")
    }
    println("======================查看剩余分区=========================")
    sparkSession.sql("show partitions ods.orders").show()
  }
}
```

Task7

7、 将ods库中lineitem表数据抽取到dwd库中fact_lineitem的分区表，分区字段为etldate且值与ods库的相对应表该值相等，抽取的条件为根据orderkey和partkey进行去重，并添加dwd_insert_user、dwd_insert_time、dwd_modify_user、dwd_modify_time四列，其中dwd_insert_user、dwd_modify_user均填写“user1”，dwd_insert_time、dwd_modify_time均填写操作时间，并进行数据类型转换。在hive cli执行show partitions dwd.fact_lineitem命令，将结果截图粘贴至对应报告中。

sql

```
show partitions dwd.fact_lineitem;
```

输出结果

```
hive (default)> show partitions dwd.fact_lineitem;
OK
partition
etldate=19971201
etldate=20220506
Time taken: 0.048 seconds, Fetched: 2 row(s)
```

查验数据条数

dwd.fact_lineitem

```
 from dwd.fact_lineitem select count(*);
```

```
hive (ods)> from dwd.fact_lineitem select count(*);
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different e
Query ID = root_20220517014535_0caeb857-c10f-4fb1-9c23-16d655bc2606
Total 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 mapreduce.job.reduces=<number>
Starting Job = job_1652714304866_0057, Tracking URL = http://master:8088/proxy/application_1652714304866_0057/
Kill Command = /opt/hadoop-2.7.7/bin/hadoop job  -kill job_1652714304866_0057
Hadoop job information for Stage-1: number of mappers: 2; number of reducers: 1
2022-05-17 01:45:40,228 Stage-1 map = 0%,  reduce = 0%
2022-05-17 01:45:51,252 Stage-1 map = 14%,  reduce = 0%, Cumulative CPU 10.95 sec
2022-05-17 01:45:52,716 Stage-1 map = 50%,  reduce = 0%, Cumulative CPU 14.0 sec
2022-05-17 01:45:54,808 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 28.42 sec
2022-05-17 01:45:57,935 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 30.63 sec
MapReduce Total cumulative CPU time: 30 seconds 630 msec
Ended Job = job_1652714304866_0057
MapReduce Jobs Launched:
Stage-Stage-1: Map: 2  Reduce: 1   Cumulative CPU: 30.63 sec   HDFS Read: 3972852 HDFS Write: 105 SUCCESS
Total MapReduce CPU Time Spent: 30 seconds 630 msec
OK
_c0
10000
Time taken: 30.092 seconds, Fetched: 1 row(s)
```

参考代码:

```
package training_one.data_clean.spark

import org.apache.spark.sql.functions.{col, current_timestamp, date_format, lit}
import org.apache.spark.sql.{DataFrame, SaveMode, SparkSession}

import java.util.Properties

/**
 * @author smy
 * @see
 * 将ods库中lineitem表数据抽取到dwd库中fact_lineitem的分区表，
 * 分区字段为etldate且值与ods库的相对应表该值相等，
 * 抽取的条件为根据orderkey和partkey进行去重，
 * 并添加dwd_insert_user、dwd_insert_time、dwd_modify_user、dwd_modify_time四列，
 * 其中dwd_insert_user、dwd_modify_user均填写“user1”，
 * dwd_insert_time、dwd_modify_time均填写当前操作时间，并进行数据类型转换。
 * 在hive cli执行show partitions dwd.fact_lineitem命令，将结果截图粘贴至对应报告中。
 * 表中有涉及到timestamp类型的，均要求按照yyyy-MM-dd HH:mm:ss，不记录毫秒数，
 * 若原数据中只有年月日，则在时分秒的位置添加00:00:00，添加之后使其符合yyyy-MM-dd HH:mm:ss。
 * @see
 * 启用hiveserver2 和metastore服务
 * nohup hive --service metastore  > /usr/local/src/hive/metastore.log 2>1 &
```

```
 * nohup hive --service hiveserver2  > /usr/local/src/hive/HS2.log 2>1 &
 */
object CleanTask3 {
  def main(args: Array[String]): Unit = {
    val warehouse: String = s"hdfs://master:9000/user/hive/warehouse"
    val sparkSession: SparkSession = SparkSession.builder()
      .config("spark.sql.warehouse.dir", warehouse) //指定hive仓库地址
      .config("hive.metastore.uris", "thrift://master:9083") // 指定
metastore地址
      .appName("CleanTask3") //设置程序名称
      .enableHiveSupport() // 连接hive
      .getOrCreate()
    val MYSQLDBURL: String = s"jdbc:mysql://master:3306/shtd_store?
useUnicode=true&characterEncoding=utf-8" // mysql url地址 // mysql url地址
    val properties: Properties = new Properties()
    properties.put("user", "root") //用户名
    properties.put("password", "123456") // 密码
    properties.put("driver", "com.mysql.jdbc.Driver") // 驱动名称
    val array: Array[String] = Array("LINEITEM")
    array.foreach(x => {
      val dataFrame: DataFrame = sparkSession.sql(s"select * from
ods.${x}")
        .dropDuplicates("orderkey", "partkey") // 去重
        .withColumn("dwd_insert_user", lit("user1")) // 新增一列
dwd_insert_user数据，值为user1
        .withColumn("dwd_insert_time", lit(date_format(current_timestamp(),
"yyyy-MM-dd HH:mm:ss")).cast("timestamp")) // 新增一列，获取当前日期
        .withColumn("dwd_modify_user", lit("user1"))
        .withColumn("dwd_modify_time", lit(date_format(current_timestamp(),
"yyyy-MM-dd HH:mm:ss")).cast("timestamp"))
      if (x.trim.toLowerCase().equals("lineitem")) {
        dataFrame.withColumn("shipdate", col("shipdate").cast("timestamp"))
          .withColumn("commentdate", col("commentdate").cast("timestamp"))
          .withColumn("receiptdate", col("receiptdate").cast("timestamp"))
          .write.mode(SaveMode.Overwrite)
          .partitionBy("etldate") // 设置分区字段
          .saveAsTable(s"dwd.fact_${x}") // 保存表

 println(s"======================fact_${x}=================================
===")
        sparkSession.sql(s"show partitions dwd.fact_lineitem").show()
      }
    })
  }
}
```

指标计算

Task1

1、 编写Scala工程代码，根据dwd层表统计每个地区、每个国家、每个月下单的数量和下单的总金额，存入MySQL数据库shtd_store的nationeverymonth表（表结构如下）中，然后在Linux的MySQL命令行中根据订单总数、消费总额、国家表主键三列均逆序排序的方式，查询出前5条，将SQL语句与执行结果截图粘贴至对应报告中;

字段	类型	中文含义	备注
nationkey	int	国家表主键	
nationname	text	国家名称	
regionkey	int	地区表主键	
regionname	text	地区名称	
totalconsumption	double	消费总额	当月消费订单总额
totalorder	int	订单总数	当月订单总额
year	int	年	订单产生的年
month	int	月	订单产生的月

sql

```
  select * from shtd_store.nationeverymonth order by totalorder
desc,totalconsumption desc,nationkey desc limit 5;
```

```
mysql> select * from shtd_store.nationeverymonth order by totalorder desc,totalconsumption desc,nationkey desc limit 5;
+-----------+------------+-----------+-------------+--------------------+------------+------+-------+
| nationkey | nationname | regionkey | regionname  | totalconsumption   | totalorder | year | month |
+-----------+------------+-----------+-------------+--------------------+------------+------+-------+
|        16 | MOZAMBIQUE |         0 | AFRICA      | 131601206.32000011 |        849 | 1998 |     1 |
|        19 | ROMANIA    |         3 | EUROPE      | 123820320.57999992 |        841 | 1997 |    12 |
|         7 | GERMANY    |         3 | EUROPE      | 125878733.22000016 |        835 | 1998 |     6 |
|         4 | EGYPT      |         4 | MIDDLE EAST | 126269785.10000002 |        825 | 1998 |     3 |
|         6 | FRANCE     |         3 | EUROPE      | 122033919.87999985 |        825 | 1998 |     5 |
+-----------+------------+-----------+-------------+--------------------+------------+------+-------+
```

Task2

2、 请根据dwd层表计算出1998每个国家的平均消费额和所有国家平均消费额相比较结果（“高/低/相同”）,存入MySQL数据库shtd_store的nationavgcmp表（表结构如下）中，然后在Linux的MySQL命令行中根据比较结果、所有国家内客单价、该国家内客单价三列均逆序排序的方式，查询出前5条，将SQL语句与执行结果截图粘贴至对应报告中;

字段	类型	中文含义	备注
nationkey	int	国家表主键	
nationname	text	国家名称	

字段	类型	中文含义	备注
nationavgconsumption	double	该国家内客单价	该国家已购买产品的人均消费额
compartmentalization	double	所有国家内客单价	所有国家已购买的产品的人均消费额
comparison	string	比较结果	国家内人均和所有国家人均相比 结果有：高/低/相同

修改mysql数据库字符编码

```
alter database shtd_store character set utf8;
```

sql

```
select * from shtd_store.nationavgcmp order by comparison desc
,allnationavgconsumption desc,nationavgconsumption desc limit 5;
```

输出结果

```
mysql> select * from shtd_store.nationavgcmp order by comparison desc ,allnationavgconsumption desc,nationavgconsumption desc limit 5;
+-----------+---------------+---------------------+-------------------------+------------+
| nationkey | nationname    | nationavgconsumption | allnationavgconsumption | comparison |
+-----------+---------------+---------------------+-------------------------+------------+
|         7 | GERMANY       |   292886.23432019877 |       284457.8564466555 | 高         |
|        23 | UNITED KINGDOM |   289290.58516448503 |       284457.8564466555 | 高         |
|         8 | INDIA         |    289049.6048263768 |       284457.8564466555 | 高         |
|        13 | JORDAN        |    287971.5400935875 |       284457.8564466555 | 高         |
|        11 | IRAQ          |    286946.0117515345 |       284457.8564466555 | 高         |
+-----------+---------------+---------------------+-------------------------+------------+
```

参考代码:

```
package spark

import org.apache.spark.sql.{DataFrame, SaveMode, SparkSession}

import java.util.Properties

object CalculateTask2 {
  def main(args: Array[String]): Unit = {
    val warehouse: String = s"hdfs://master:9000/user/hive/warehouse"
    val sparkSession: SparkSession = SparkSession.builder()
      .config("spark.sql.warehouse.dir", warehouse) //指定hive仓库地址
      .config("hive.metastore.uris", "thrift://master:9083") // 指定
metastore地址
      .config("spark.sql.shuffle.partitions", 1000) //设置shuffle的分区数
      .appName("CalculateTask2") //设置程序名称
      .enableHiveSupport() // 连接hive
      .getOrCreate()
    val MYSQLDBURL: String = "jdbc:mysql://master:3306/shtd_store?
useUnicode=true&characterEncoding=utf-8" // mysql url地址
    val properties: Properties = new Properties()
```

```
    properties.put("user", "root") //用户名
    properties.put("password", "123456") // 密码
    properties.put("driver", "com.mysql.jdbc.Driver") // 驱动名称
    /**
     * 统计1998年每个国家每个用户的消费总额
     */
    val selectData: String =
      """
        |from dwd.dim_nation,
        |     dwd.fact_orders,
        |     dwd.dim_customer
        |select cast(dim_nation.nationkey as INT)           as nationkey,
        |       dim_nation.name                             as nationname,
        |       fact_orders.custkey,
        |       sum(cast(fact_orders.totalprice as DOUBLE)) as
cust_totalprice
        |where dim_nation.nationkey = dim_customer.nationkey
        |  and dim_customer.custkey = fact_orders.custkey
        |and year(cast(fact_orders.orderdate as date)) = 1998
        |group by dim_nation.nationkey, dim_nation.name,fact_orders.custkey
        |""".stripMargin
    sparkSession.sql(selectData).createTempView("selectData")
    /**
     * 统计每个国家的平均消费额,统计每个国家的消费总额和人数
     */
    val margin: String =
      """
        |from selectData
        |select nationkey,nationname,
        |avg(cust_totalprice) as nationavgconsumption,
        |sum(cust_totalprice) as nation_totalprice,
        |count(*)  as population_num
        |group by nationkey,nationname
        |""".stripMargin
    sparkSession.sql(margin).createTempView("margin")
    /**
     * 开窗计算所有国家的平均消费额
     */
    val stripMargin: String =
      s"""
         |from margin
         |select nationkey,nationname,nationavgconsumption,
         |sum(nation_totalprice) over() / sum(population_num) over() as
allnationavgconsumption
         |""".stripMargin
    sparkSession.sql(stripMargin).createTempView("stripMargin")
    /**
     * 根据国家的平均消费值和所有国家平均消费值比较
     */
    val whenSQL: String =
      """
```

```
        |select *,
        |       case
        |           when nationavgconsumption < allnationavgconsumption
then '低'
        |           when nationavgconsumption > allnationavgconsumption
then '高'
        |           else '相同'
        |           end as comparison
        |from stripMargin
        |""".stripMargin
    val result: DataFrame = sparkSession.sql(whenSQL)
    result.show()
    result.write.mode(SaveMode.Overwrite).jdbc(MYSQLDBURL, "nationavgcmp",
properties)
  }

}
```

Task3

3、 编写Scala工程代码，根据dwd层表统计连续两个月下单并且下单金额保持增长的用户，订单发生时间限制为大于等于1997年，存入MySQL数据库shtd_store的usercontinueorder表(表结构如下)中。然后在Linux的MySQL命令行中根据订单总数、消费总额、客户主键三列均逆序排序的方式，查询出前5条，将SQL语句与执行结果截图粘贴至对应报告中。

字段	类型	中文含义	备注
custkey	int	客户主键	
custname	text	客户名称	
month	text	月	记录当前月和下月，用下划线'_'相连 例如：202201_202202 表示2022年1月到2月用户连续下单。
totalconsumption	double	消费总额	连续两月的订单总额

字段	类型	中文含义	备注
totalorder	int	订单总数	连续两月的订单总数

sql

```
select * from shtd_store.usercontinueorder order by totalorder desc
,totalconsumption desc ,custkey desc limit 5;
```

输出结果

```
mysql> select * from shtd_store.usercontinueorder order by totalorder desc ,totalconsumption desc ,custkey desc limit 5;
+---------+-------------------+--------------+--------------------+------------+
| custkey | custname          | month        | totalconsumption   | totalorder |
+---------+-------------------+--------------+--------------------+------------+
|  116107 | Customer#000116107 | 199801_199802 |          576267.5 |          4 |
|    7978 | Customer#000007978 | 199801_199802 | 505157.67000000004 |          4 |
|   53815 | Customer#000053815 | 199803_199804 |          490768.12 |          4 |
|   53815 | Customer#000053815 | 199802_199803 |          418904.14 |          4 |
|  115810 | Customer#000115810 | 199806_199807 |          403621.36 |          4 |
+---------+-------------------+--------------+--------------------+------------+
5 rows in set (0.01 sec)
```

参考代码:

```
package training_one.data_calculation.spark

import org.apache.spark.sql.functions.{col, concat, concat_ws}
import org.apache.spark.sql.{DataFrame, SaveMode, SparkSession}

import java.util.Properties

/**
 * @author smy
 * @see
 * 编写Scala工程代码，根据dwd层表统计连续两个月下单并且下单金额保持增长的用户，
 * 订单发生时间限制为大于等于1997年，
 * 存入MySQL数据库shtd_store的usercontinueorder表(表结构如下)中。
 * 然后在Linux的MySQL命令行中根据订单总数、消费总额、客户主键三列均逆序排序的方式，
 * 查询出前5条，将SQL语句与执行结果截图粘贴至对应报告中。
 * @see
 * |        字段        |  类型   | 中文含义 | 备注
                                |
 * | :--------------: | :----: | :------: | :-----------------------------
------------------------------ |
 * |     custkey      |  int   | 客户主键 |
                                |
```

```
 * |     custname     |  text  | 客户名称 |
                                |
 * |      month       |  text  |    月    | 记录当前月和下月，用下划线‘_’相连
例如： 202201_202202 表示2022年1月到2月用户连续下单。 |
 * | totalconsumption | double | 消费总额 | 连续两月的订单总额
                                |
 * |    totalorder    |  int   | 订单总数 | 连续两月的订单总数
                                |
 * @version spark 2.1.1
 *          scala 2.11.0
 *          mysql 5.1.47
 */
object CalculateTask3 {
  def main(args: Array[String]): Unit = {
    val warehouse: String = s"hdfs://master:9000/user/hive/warehouse"
    val sparkSession: SparkSession = SparkSession.builder()
      .config("spark.sql.warehouse.dir", warehouse) //指定hive仓库地址
      .config("hive.metastore.uris", "thrift://master:9083") // 指定
metastore地址
      .config("spark.sql.shuffle.partitions", 1000) //设置shuffle的分区数
      .appName("CalculateTask3") //设置程序名称
      .enableHiveSupport() // 连接hive
      .getOrCreate()
    // mysql url地址
    val MYSQLDBURL: String = s"jdbc:mysql://master:3306/shtd_store?
useUnicode=true&characterEncoding=utf-8"
    val properties: Properties = new Properties()
    properties.put("user", "root") //用户名
    properties.put("password", "123456") // 密码
    properties.put("driver", "com.mysql.jdbc.Driver") // 驱动名称
    // 获取需要的字段数据
    val selectData: String =
      """
        |from dwd.dim_customer,
        |     dwd.fact_orders
        |select cast(dim_customer.custkey as INT)           as custkey,
        |       dim_customer.name                           as custname,
        |       year(fact_orders.orderdate)                 as `year`,
        |       date_format(fact_orders.orderdate, "MM")    as `month`,
        |       sum(cast(fact_orders.totalprice as DOUBLE)) as
totalpriceSUM
        |where year(fact_orders.orderdate) >= 1997
        |  and fact_orders.custkey = dim_customer.custkey
        |group by dim_customer.custkey,
        |         dim_customer.name,
        |         year(fact_orders.orderdate),
        |         date_format(fact_orders.orderdate, "MM")
        |""".stripMargin
    sparkSession.sql(selectData).createTempView("selectData")
    val selectLag: String =
      s"""
```

```
          |select *,
          |       lead(`year`) over (partition by custkey,custname order by
`year`,`month`)             as `next_year`,
          |       lead(`month`) over (partition by custkey,custname order by
`year`,`month`)           as `next_month`,
          |       lead(`totalpriceSUM`) over (partition by custkey,custname
order by `year`,`month`) as `next_totalpriceSUM`
          |from selectData
          |""".stripMargin
    //过滤null值
    val dataFrame: DataFrame = sparkSession.sql(selectLag).filter(
      col("next_year").isNotNull &&
        col("next_month").isNotNull &&
        col("next_totalpriceSUM").isNotNull)
      .filter( //过滤非连续两个月下单用户
        ((col("next_year").cast("int") - col("year").cast("int")) * 12 +
          col("next_month").cast("int") - col("month").cast("int")) === 1)
      .filter(col("next_totalpriceSUM") > col("totalpriceSUM")) //过滤非月
下单金额保持增长的用户
      .withColumn("month", concat_ws("_", concat(col("year"),
col("month")), concat(col("next_year"), col("next_month"))))
      .withColumn("totalconsumption", col("totalpriceSUM") +
col("next_totalpriceSUM"))
    dataFrame.createTempView("tmpTable")
    val frame: DataFrame = sparkSession.sql("select
`custkey`,`custname`,`month`,`totalconsumption`,count(*) over (partition by
custkey,custname) as totalorder from tmpTable")
    frame.write.mode(SaveMode.Overwrite).jdbc(MYSQLDBURL,
"usercontinueorder", properties)
  }
}
```

Task4

4、 根据dwd层表统计每人每天下单的数量和下单的总金额，存入dws层的customer_consumption_day_aggr表（表结构如下）中，然后在hive cli中按照cust_key，totalconsumption, totalorder三列均逆序排序的方式，查询出前5条，将SQL语句与执行结果截图粘贴至对应报告中;

字段	类型	中文含义	备注
cust_key	int	客户key	
cust_name	string	客户名称	
totalconsumption	double	消费总额	当月消费订单总额
totalorder	int	订单总数	当月订单总额
year	int	年	订单产生的年
month	int	月	订单产生的月

字段	类型	中文含义	备注
day	int	日	订单产生的日

sql

```
select * from dws.customer_consumption_day_aggr order by cust_key desc
,totalconsumption desc ,totalorder desc limit 5;
```

输出结果

```
149999  Customer#000149999      211382.23       1       1992    8       2
149999  Customer#000149999      46217.79        1       1998    7       6
149998  Customer#000149998      380681.85       1       1995    9       15
149998  Customer#000149998      158592.64       1       1998    3       6
149998  Customer#000149998      146650.42       1       1997    12      12
```

参考代码:

```
package spark

import org.apache.spark.sql.{DataFrame, SaveMode, SparkSession}

import java.util.Properties

/**
 * @author smy
 * @see
 * 根据dwd层表统计每人每天下单的数量和下单的总金额，
 * 存入dws层的customer_consumption_day_aggr表（表结构如下）中，
 * 然后在hive cli中按照cust_key，totalconsumption，totalorder三列均逆序排序的
方式，
 * 查询出前5条，将SQL语句与执行结果截图粘贴至对应报告中；
 */
object CalculateTask4 {
  def main(args: Array[String]): Unit = {
    val warehouse: String = "hdfs://master/user/hive/warehouse"
    val sparkSession: SparkSession = SparkSession.builder()
      .config("spark.sql.warehouse.dir", warehouse) //指定hive仓库地址
      .config("hive.metastore.uris", "thrift://master:9083") // 指定
metastore地址
      .config("spark.sql.shuffle.partitions", 1000) //设置shuffle的分区数
      .config("hive.mapred.mode","nonstrict")
      .appName("CalculateTask4") //设置程序名称
      .enableHiveSupport() // 连接hive
      .getOrCreate()
    val MYSQLDBURL: String = "jdbc:mysql://master:3306/shtd_store?
useUnicode=true&characterEncoding=utf-8" // mysql url地址
    val properties: Properties = new Properties()
    properties.put("user", "root") //用户名
    properties.put("password", "123456") // 密码
```

```
    properties.put("driver", "com.mysql.jdbc.Driver") // 驱动名称
    // 获取需要的数据集
    val margin: String =
      """
        |from dwd.dim_customer,
        |     dwd.fact_orders
        |select cast(dim_customer.custkey as INT)           as cust_key,
        |       dim_customer.name                           as cust_name,
        |       sum(cast(fact_orders.totalprice as DOUBLE)) as
totalconsumption,
        |       count(*)                                    as totalorder,
        |       year(fact_orders.orderdate)                 as `year`,
        |       month(fact_orders.orderdate)                as `month`,
        |       day(fact_orders.orderdate)                  as `day`
        |where dim_customer.custkey = fact_orders.custkey
        |group by cast(dim_customer.custkey as INT),
        |         dim_customer.name,
        |         year(fact_orders.orderdate),
        |         month(fact_orders.orderdate),
        |         day(fact_orders.orderdate)
        |""".stripMargin
    val dataFrame: DataFrame = sparkSession.sql(margin)
    dataFrame.createTempView("tmpTable")
    // 创建表
    sparkSession.sql("create table if not exists
dws.customer_consumption_day_aggr like tmpTable")

 dataFrame.write.mode(SaveMode.Overwrite).saveAsTable("dws.customer_consump
tion_day_aggr")
  }
}
```

Task5

5、 根据dws层表customer_consumption_day_aggr表，再联合dwd.dim_region,dwd.dim_nation统计每人每个月下单的数量和下单的总金额，并按照cust_key，totalconsumption，totalorder，month进行分组逆序排序（以cust_key为分组条件），将计算结果存入MySQL数据库shtd_store的customer_consumption_month_aggr表（表结构如下）中，然后在Linux的MySQL命令行中根据订单总数、消费总额、国家表主键三列均逆序排序的方式，查询出前5条，将SQL语句与执行结果截图粘贴至对应报告中;

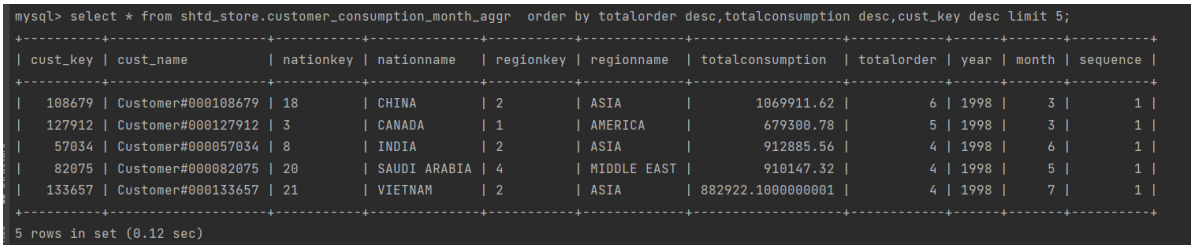
字段	类型	中文含义	备注
cust_key	int	客户key	
cust_name	string	客户名称	
nationkey	int	国家表主键	
nationname	text	国家名称	

字段	类型	中文含义	备注
regionkey	int	地区表主键	
regionname	text	地区名称	
totalconsumption	double	消费总额	当月消费订单总额
totalorder	int	订单总数	当月订单总额
year	int	年	订单产生的年
month	int	月	订单产生的月
sequence	Int	次序	

sql

```
select * from shtd_store.customer_consumption_month_aggr  order by
totalorder desc,totalconsumption desc,cust_key desc limit 5;
```

输出结果



```
mysql> select * from shtd_store.customer_consumption_month_aggr  order by totalorder desc,totalconsumption desc,cust_key desc limit 5;
+----------+--------------------+-----------+-------------+-----------+-------------+-------------------+------------+------+-------+----------+
| cust_key | cust_name          | nationkey | nationname  | regionkey | regionname  | totalconsumption  | totalorder | year | month | sequence |
+----------+--------------------+-----------+-------------+-----------+-------------+-------------------+------------+------+-------+----------+
|   108679 | Customer#000108679 | 18        | CHINA       | 2         | ASIA        |        1069911.62 |          6 | 1998 |     3 |        1 |
|   127912 | Customer#000127912 | 3         | CANADA      | 1         | AMERICA     |         679300.78 |          5 | 1998 |     3 |        1 |
|    57034 | Customer#000057034 | 8         | INDIA       | 2         | ASIA        |         912885.56 |          4 | 1998 |     6 |        1 |
|    82075 | Customer#000082075 | 20        | SAUDI ARABIA | 4         | MIDDLE EAST |         910147.32 |          4 | 1998 |     5 |        1 |
|   133657 | Customer#000133657 | 21        | VIETNAM     | 2         | ASIA        | 882922.1000000001 |          4 | 1998 |     7 |        1 |
+----------+--------------------+-----------+-------------+-----------+-------------+-------------------+------------+------+-------+----------+
5 rows in set (0.12 sec)
```

参考代码:

```
package spark

import org.apache.spark.sql.{DataFrame, SaveMode, SparkSession}
import org.apache.spark.sql.functions.desc

import java.util.Properties

/**
 * @author smy
 * @see
 * 根据dws层表customer_consumption_day_aggr表，
 * 再联合dwd.dim_region,dwd.dim_nation统计每人每个月下单的数量和下单的总金额，
 * 并按照cust_key，totalconsumption，totalorder，month进行分组逆序排序（以
cust_key为分组条件），
 * 将计算结果存入MySQL数据库shtd_store的customer_consumption_month_aggr表（表
结构如下）中，
 * 然后在Linux的MySQL命令行中根据订单总数、消费总额、国家表主键三列均逆序排序的
方式，
 * 查询出前5条，将SQL语句与执行结果截图粘贴至对应报告中；
 */
object CalculateTask5 {
```

```
  def main(args: Array[String]): Unit = {
    val warehouse: String = s"hdfs://master/user/hive/warehouse"
    val sparkSession: SparkSession = SparkSession.builder()
      .config("spark.sql.warehouse.dir", warehouse) //指定hive仓库地址
      .config("hive.metastore.uris", "thrift://master:9083") // 指定
metastore地址
      .config("spark.sql.shuffle.partitions", 1000) //设置shuffle的分区数
      .appName("CalculateTask5") //设置程序名称
      .enableHiveSupport() // 连接hive
      .getOrCreate()
    val MYSQLDBURL: String = s"jdbc:mysql://master:3306/shtd_store?
useUnicode=true&characterEncoding=utf-8" // mysql url地址
    val properties: Properties = new Properties()
    properties.put("user", "root") //用户名
    properties.put("password", "123456") // 密码
    properties.put("driver", "com.mysql.jdbc.Driver") // 驱动名称
    val margin: String =
      """
        |from dws.customer_consumption_day_aggr,
        |     dwd.dim_nation,
        |     dwd.dim_region,
        |     dwd.dim_customer
        |select cust_key,
        |       cust_name,
        |       dim_nation.nationkey,
        |       dim_nation.name       as nationname,
        |       dim_region.regionkey,
        |       dim_region.name       as regionname,
        |       sum(totalconsumption) as totalconsumption,
        |       sum(totalorder)       as totalorder,
        |       `year`,
        |       `month`
        |where dim_nation.regionkey = dim_region.regionkey
        |  and cust_key = dim_customer.custkey
        |  and dim_customer.nationkey = dim_nation.nationkey
        |group by cust_key,
        |         cust_name,
        |         dim_nation.nationkey,
        |         dim_nation.name,
        |         dim_region.regionkey,
        |         dim_region.name,
        |         `year`,
        |         `month`
        |""".stripMargin
        sparkSession.sql(margin).createTempView("margin")
    val stripMargin: String =
      """
        |select *,row_number()
        |over (partition by cust_key order by totalconsumption
desc,totalorder desc,year desc,month desc) as sequence
        |from  margin
```

```
        |""".stripMargin
    val dataFrame: DataFrame = sparkSession.sql(stripMargin)
    dataFrame.sort(desc("sequence")).show()
    dataFrame.write.mode(SaveMode.Overwrite).jdbc(MYSQLDBURL,
"customer_consumption_month_aggr", properties)
  }
}
```

Task6

6、 请根据dws层表fact_orders表，再联合dwd.dim_nation计算出1998年每个国家的消费额中中位数,存入MySQL数据库shtd_store的nationmedian表（表结构如下）中，然后在Linux的MySQL命令行中根据所有国家中位数,该国家中位数、国家表主键三列均逆序排序的方式，查询出前5条，将SQL语句与执行结果截图粘贴至对应报告中;

字段	类型	中文含义	备注
nationkey	int	国家表主键	
nationname	text	国家名称	
nationmedianconsumption	double	该国家内客单价	该国家已购买产品的金额的中位数
allnationmedianconsumption	double	所有国家订单的中位数	所有国家已购买的产品的金额的中位数

sql

```
select * from shtd_store.nationmedian order by allnationmedianconsumption
desc,nationmedianconsumption desc ,nationkey desc limit 5;
```

输出结果

```
mysql> select * from shtd_store.nationmedian order by allnationmedianconsumption desc,nationmedianconsumption desc ,nationkey desc limit 5;
+-----------+---------------+------------------------+---------------------------+
| nationkey | nationname    | nationmedianconsumption | allnationmedianconsumption |
+-----------+---------------+------------------------+---------------------------+
|        19 | ROMANIA       |      147311.22999999998 |                 144647.08 |
|        23 | UNITED KINGDOM |              147209.285 |                 144647.08 |
|        10 | IRAN          |              146196.615 |                 144647.08 |
|        21 | VIETNAM       |      146176.85499999998 |                 144647.08 |
|        20 | SAUDI ARABIA  |      146000.90000000002 |                 144647.08 |
+-----------+---------------+------------------------+---------------------------+
5 rows in set (0.00 sec)
```

自定义UDAF求中位数

```
package training_one.data_calculation.spark

import org.apache.spark.sql.Row
import org.apache.spark.sql.expressions.{MutableAggregationBuffer,
UserDefinedAggregateFunction}
import org.apache.spark.sql.types.{ArrayType, DataType, DoubleType,
LongType, StructField, StructType}
```

```scala
import scala.collection.mutable
import scala.collection.mutable.ArrayBuffer

object CustomMedian extends UserDefinedAggregateFunction {
  // 设置输入列的数据类型
  override def inputSchema: StructType =
StructType(StructField("inputColumn", DoubleType) :: Nil)

  // 设置缓冲区中值的数据类型ArrayType(DoubleType,containsNull = false)) 不包含null值 和数据条数
  override def bufferSchema: StructType = {
    StructType(StructField("arrayBuffer", ArrayType(DoubleType,
containsNull = false)) :: Nil)
  }
  // 设置输出的数据类型
  override def dataType: DataType = DoubleType
  // 是否验证数据类型
  override def deterministic: Boolean = true

  // 设置数据初始化值
  override def initialize(buffer: MutableAggregationBuffer): Unit = {
    buffer(0) = new ArrayBuffer[Double]()
  }
  override def update(buffer: MutableAggregationBuffer, input: Row): Unit =
{
    if (!input.isNullAt(0)) {
      // 将将数组转换长数组缓冲
      val toBuffer: mutable.Buffer[Double] =
buffer.getAs[mutable.WrappedArray[Double]](0).toBuffer
      // 将数据添加到ArrayBuffer中
      toBuffer += input.getAs[Double](0)
      buffer(0) = toBuffer
    }

  }
  //合并两个聚合缓冲区并将更新的缓冲区值存储回 `buffer1`
  override def merge(buffer1: MutableAggregationBuffer, buffer2: Row): Unit
= {
    buffer1(0) = buffer1.getAs[mutable.WrappedArray[Double]](0) ++
buffer2.getAs[mutable.WrappedArray[Double]](0)
  }

  // 计算最后的结果
  override def evaluate(buffer: Row): Any = {
    // 排序
    val sortBuffer: mutable.Buffer[Double] =
buffer.getAs[mutable.WrappedArray[Double]](0).sorted.toBuffer
    // 数据长度
    val sortBufferSize: Int = sortBuffer.size
    if ((sortBufferSize % 2).equals(0)){
      val lastIndex: Int = sortBufferSize / 2
```

```
      (sortBuffer(lastIndex) + sortBuffer(lastIndex - 1)) / 2
    }else{
      sortBuffer((sortBufferSize / 2).toInt)
    }
  }
}
```

参考代码

```
package training_one.data_calculation.spark

import org.apache.spark.sql.functions.lit
import org.apache.spark.sql.{DataFrame, SaveMode, SparkSession}

import java.util.Properties

/**
 * @author smy
 * @see
 * 请根据dwd层表fact_orders表，再联合dwd.dim_nation计算出1998年每个国家的消费额中中位数，
 * 存入MySQL数据库shtd_store的nationmedian表（表结构如下）中，
 * 然后在Linux的MySQL命令行中根据该国家内客单价 、所有国家订单的中位数 、国家表主键三列均逆序排序的方式，
 * 查询出前5条，将SQL语句与执行结果截图粘贴至对应报告中；
 * | 字段                          | 类型    | 中文含义              | 备注                          |
 * | ----------------------------- | ------ | --------------------- | ------------------------------ |
 * | nationkey                     | int    | 国家表主键            |                               |
 * | nationname                    | text   | 国家名称              |                               |
 * | nationmedianconsumption       | double | 该国家内客单价        | 该国家已购买产品的金额的中位数      |
 * | allnationmedianconsumption | double | 所有国家订单的中位数 | 所有国家已购买的产品的金额的中位数 |
 */
object CalculateTask6 {
  def main(args: Array[String]): Unit = {
    val warehouse: String = s"hdfs://master:9000/user/hive/warehouse"
    val sparkSession: SparkSession = SparkSession.builder()
      .config("spark.sql.warehouse.dir", warehouse) //指定hive仓库地址
      .config("hive.metastore.uris", "thrift://master:9083") // 指定metastore地址
      .config("spark.sql.shuffle.partitions", 1000) //设置shuffle的分区数
      .appName("CalculateTask6") //设置程序名称
      .enableHiveSupport() // 连接hive
      .getOrCreate()
    val MYSQLDBURL: String = s"jdbc:mysql://master:3306/shtd_store?useUnicode=true&characterEncoding=utf-8" // mysql url地址
```

```
    val properties: Properties = new Properties()
    properties.put("user", "root") //用户名
    properties.put("password", "123456") // 密码
    properties.put("driver", "com.mysql.jdbc.Driver") // 驱动名称
    // 注册自定义udaf函数
    sparkSession.udf.register("custom_median", CustomMedian)

    /**
     * percentile_approx:对数据类型没有要求，近似中位数,如果第三个参数大于数据
条数可以得到精确值中位数
     * percentile:int类型，精准中位数
     */
    /**
     * 获取1998年,每个国家每个人的消费订单数据
     */
    val margin: String =
      """
        |from dwd.fact_orders,
        |     dwd.dim_nation,
        |     dwd.dim_customer
        |select cast(dim_nation.nationkey as INT)
            as nationkey,
        |       dim_nation.name
            as nationname,
        |       dim_customer.custkey,
        |       custom_median(cast(fact_orders.totalprice as DOUBLE))
        |                     over (partition by
dim_nation.nationkey,dim_nation.name ) as nationmedianconsumption,
        |       cast(fact_orders.totalprice as DOUBLE)
             as totalprice
        |where fact_orders.custkey = dim_customer.custkey
        |  and dim_customer.nationkey = dim_nation.nationkey
        |  and year(fact_orders.orderdate) = 1998
        |""".stripMargin
    val dataFrame: DataFrame = sparkSession.sql(margin)
    dataFrame.show()
    /**
     * 计算所有国家已购买的产品的金额的中位数,
     * 如果直接用over会产生数据倾斜的问题
     */
    val array: Array[Double] = dataFrame.select("totalprice")
      .collect().map(x => x.get(0).toString.toDouble).sorted
    var allnationmedianconsumption: Double = 0.0
    if (array.length % 2 != 0) {
      val medianIndex: Int = (array.length / 2).toInt
      allnationmedianconsumption = array(medianIndex)
    } else {
      val medianIndex: Int = (array.length / 2)
      allnationmedianconsumption = (array(medianIndex) + array(medianIndex
- 1)) / 2
    }
```

```
    println(allnationmedianconsumption)
    val result: DataFrame = dataFrame.select("nationkey", "nationname",
"nationmedianconsumption")
      .withColumn("allnationmedianconsumption",
lit(allnationmedianconsumption))
      .dropDuplicates("nationkey")
    result.show()
    result.write.mode(SaveMode.Overwrite).jdbc(MYSQLDBURL, "nationmedian",
properties)
  }
}
```

Task7

7、 根据MySQL数据库shtd_store的usercontinueorder表结合表dim_nation与表customer_consumption_day_aggr，请计算每个国家连续两个月下单并且下单金额保持增长与该国已下单用户的占比，将结果存入MySQL数据库shtd_store的userrepurchased表(表结构如下)中。然后在Linux的MySQL命令行中根据占比、连续下单人数、国家主键三列均逆序排序的方式，查询出前5条，将SQL语句与执行结果截图粘贴至对应报告中。

字段	类型	中文含义	备注
nationkey	int	国家主键	
nationname	string	国家名称	
purchaseduser	int	下单人数	该国家内已下单人数
repurchaseduser	double	连续下单人数	该国家内连续两个月下单并且下单金额保持增长人数
repurchaserate	int	占比	下单人数/连续下单人数（保留3位小数）

sql

```
select * from shtd_store.userrepurchased order by repurchaserate desc
,repurchaseduser desc ,nationkey desc limit 5;
```

输出结果

```
mysql> select * from shtd_store.userrepurchased order by repurchaserate desc ,repurchaseduser desc ,nationkey desc limit 5;
+-----------+-------------+---------------+-----------------+----------------+
| nationkey | nationname  | purchaseduser | repurchaseduser | repurchaserate |
+-----------+-------------+---------------+-----------------+----------------+
|         8 | INDIA       |          3315 |             460 |          0.139 |
|        16 | MOZAMBIQUE  |          3417 |             454 |          0.133 |
|        20 | SAUDI ARABIA |          3248 |             429 |          0.132 |
|         7 | GERMANY     |          3315 |             435 |          0.131 |
|         2 | BRAZIL      |          3300 |             430 |          0.130 |
+-----------+-------------+---------------+-----------------+----------------+
5 rows in set (0.00 sec)
```

参考代码:

```
package spark
```

```
import org.apache.spark.sql._

import java.util.Properties

/**
 * @author smy
 * @see
 *      使用MySQL数据库shtd_store的usercontinueorder表
 *      和dws的customer_consumption_day_aggr并结合dwd表dim_nation,
 *      请计算每个国家连续两个月下单并且下单金额保持增长与该国已下单用户的占比
 *      ,将结果存入MySQL数据库shtd_store的userrepurchased表(表结构如下)中。
 *      然后在Linux的MySQL命令行中根据订单总数、消费总额、客户主键三列均逆序排序
的方式,
 *      查询出前5条,将SQL语句与执行结果截图粘贴至对应报告中。
 *      | 字段            | 类型   | 中文含义       | 备注
                |
 *      | --------------- | ------ | ------------ | -----------------------
----------------------- |
 *      | nationkey       | int    | 国家主键       |
                 |
 *      | nationname      | string | 国家名称       |
                 |
 *      | purchaseduser   | int    | 下单人数       | 该国家内已下单人数
                |
 *      | repurchaseduser | double | 连续下单人数 | 该国家内连续两个月下单并且
下单金额保持增长人数 |
 *      | repurchaserate  | int    | 占比         | 连续下单人数/下单人数(保留
3位小数)           |
 * @see dws.customer_consumption_day_aggr
 *      | 字段             | 类型   | 中文含义 | 备注             |
 *      | ---------------- | ------ | -------- | ---------------- |
 *      | cust_key         | int    | 客户key  |                  |
 *      | cust_name        | string | 客户名称 |                  |
 *      | totalconsumption | double | 消费总额 | 当日消费订单总额 |
 *      | totalorder       | int    | 订单总数 | 当日订单总数     |
 *      | year             | int    | 年       | 订单产生的年     |
 *      | month            | int    | 月       | 订单产生的月     |
 *      | day              | int    | 日       | 订单产生的日     |
 * @see usercontinueorder
 *      |      字段      |  类型  | 中文含义 | 备注
                        |
 *      | :-------------: | :----: | :------: | :-------------------------
---------------------------------- |
 *      |     custkey     |  int   | 客户主键 |
                          |
 *      |     custname    |  text  | 客户名称 |
                          |
 *      |      month      |  text  |    月    | 记录当前月和下月,用下划
线'_'相连<br /> 例如: 202201_202202 表示2022年1月到2月用户连续下单。  |
```

```
 *      | totalconsumption | double | 消费总额 | 连续两月的订单总额
                                  |
 *      |    totalorder    |  int   | 订单总数 | 连续两月的订单总数
 */
object CalculateTask7 {
  def main(args: Array[String]): Unit = {
    val warehouse: String = s"hdfs://master:9000/user/hive/warehouse"
    val sparkSession: SparkSession = SparkSession.builder()
      .config("spark.sql.warehouse.dir", warehouse) //指定hive仓库地址
      .config("hive.metastore.uris", "thrift://master:9083") // 指定
metastore地址
      .config("spark.sql.shuffle.partitions", 1000) //设置shuffle的分区数
      .appName("CalculateTask7") //设置程序名称
      .enableHiveSupport() // 连接hive
      .getOrCreate()
    val MYSQLDBURL: String = s"jdbc:mysql://master:3306/shtd_store?
useUnicode=true&characterEncoding=utf-8" // mysql url地址
    val properties: Properties = new Properties()
    properties.put("user", "root") //用户名
    properties.put("password", "123456") // 密码
    properties.put("driver", "com.mysql.jdbc.Driver") // 驱动名称
    sparkSession.read.jdbc(MYSQLDBURL, "usercontinueorder",
properties).createTempView("usercontinueorder")
    val margin: String =
      """
        |from dwd.dim_nation,
        |     dwd.dim_customer,
        |     dws.customer_consumption_day_aggr
        |select cast(dim_nation.nationkey as INT)      as nationkey,
        |       dim_nation.name                        as nationname,
        |       count(distinct (dim_customer.custkey)) as purchaseduser
        |where dim_nation.nationkey = dim_customer.nationkey
        |  and dim_customer.custkey =
customer_consumption_day_aggr.cust_key
        |group by dim_nation.nationkey, dim_nation.name
        |""".stripMargin
    sparkSession.sql(margin).createTempView("purchaseduserTable")
    val stripMargin: String =
      """
        |from dwd.dim_nation,
        |     dwd.dim_customer,
        |     usercontinueorder
        |select cast(dim_nation.nationkey as INT) as nationkey,
        |       dim_nation.name                   as nationname,
        |       count(distinct (usercontinueorder.custkey)) as
repurchaseduser
        |where dim_nation.nationkey = dim_customer.nationkey
        |  and dim_customer.custkey = usercontinueorder.custkey
        |group by dim_nation.nationkey, dim_nation.name
        |""".stripMargin
    sparkSession.sql(stripMargin).createTempView("usercontinueorderTable")
```

```
    val result: String =
      """
        |from usercontinueorderTable,
        |     purchaseduserTable
        |select purchaseduserTable.nationkey,
        |       purchaseduserTable.nationname,
        |       purchaseduserTable.purchaseduser,
        |       usercontinueorderTable.repurchaseduser,
        |       cast((repurchaseduser / purchaseduser ) as decimal(18,3))
as repurchaserate
        |where purchaseduserTable.nationkey =
usercontinueorderTable.nationkey
        |  and purchaseduserTable.nationname =
usercontinueorderTable.nationname
        |""".stripMargin

 sparkSession.sql(result).write.mode(SaveMode.Overwrite).jdbc(MYSQLDBURL,
"userrepurchased", properties)
  }
}
```