**环境说明：**

|  |
| --- |
| 集群启动命令为在Master上执行allstart.sh；  各主机可通过SSH客户端进行SSH访问（需通过22映射的宿主机端口号访问）；  Master节点MySQL数据库用户名/密码：root/123456（已配置远程连接，需通过3306映射的宿主机端口号连接）；  Hive的配置文件位于/opt/module/hive-3.1.2/conf/  Spark任务在Yarn上用Client运行，方便观察日志。 |

## 离线数据处理

### 任务一：数据抽取

使用Scala编写spark工程代码，将MySQL的ds\_db01库中表customer\_inf、customer\_inf、order\_detail、order\_master、product\_info的数据增量抽取到Hive的ods库（需自建）中对应表customer\_inf、order\_detail、order\_master、product\_info中。

1. 抽取ds\_db01库中customer\_inf的增量数据进入Hive的ods库中表customer\_inf。根据ods.user\_info表中modified\_time作为增量字段，只将新增的数据抽入，字段名称、类型不变，同时添加静态分区，分区字段为etl\_date，类型为String，且值为当前日期的前一天日期（分区字段格式为yyyyMMdd）。使用hive cli执行show partitions ods.customer\_inf命令；

答题区域

**答题时使用的Jar程序包**

**答题时在Linux中使用的命令**

* hive建表：

create table customer\_inf  
(  
 customer\_inf\_id int,  
 customer\_id int,  
 customer\_name varchar(20),  
 identity\_card\_type tinyint,  
 identity\_card\_no varchar(20),  
 mobile\_phone varchar(50),  
 customer\_email varchar(50),  
 gender varchar(1),  
 customer\_point int,  
  
 register\_time date,  
 birthday date,  
 customer\_level int,  
 customer\_money int,  
 modified\_time date  
)  
 partitioned by (etl\_date String)  
 row format delimited fields terminated by ',';

* Linux中的任务提交命令：

/opt/software/spark-3.0.0-bin-hadoop3.2/bin/spark-submit ./test.jar ds\_db01.customer\_inf hdfs://liming141/res1

* Hive中的数据加载命令

load data inpath "hdfs://liming141/res1/part-00000-66fff6d5-8f76-44c8-a0c2-76f783aa4246-c000.csv" into table customer\_inf;

1. 抽取ds\_db01库中product\_info的增量数据进入Hive的ods库中表product\_info。根据ods.product\_info表中modified\_time作为增量字段，只将新增的数据抽入，字段名称、类型不变，同时添加静态分区，分区字段为etl\_date，类型为String，且值为当前日期的前一天日期（分区字段格式为yyyyMMdd）。使用hive cli执行show partitions ods.product\_info命令；
2. 抽取ds\_db01库中order\_master的增量数据进入Hive的ods库中表order\_master，根据ods.order\_master表中modified\_time作为增量字段，只将新增的数据抽入，字段名称、类型不变，同时添加静态分区，分区字段为etl\_date，类型为String，且值为当前日期的前一天日期（分区字段格式为yyyyMMdd）。使用hive cli执行show partitions ods.order\_master命令；
3. 抽取ds\_db01库中order\_detail的增量数据进入Hive的ods库中表order\_detail，根据ods.order\_detail表中modified\_time作为增量字段，只将新增的数据抽入，字段名称、类型不变，同时添加静态分区，分区字段为etl\_date，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）。使用hive cli执行show partitions ods.order\_detail命令。

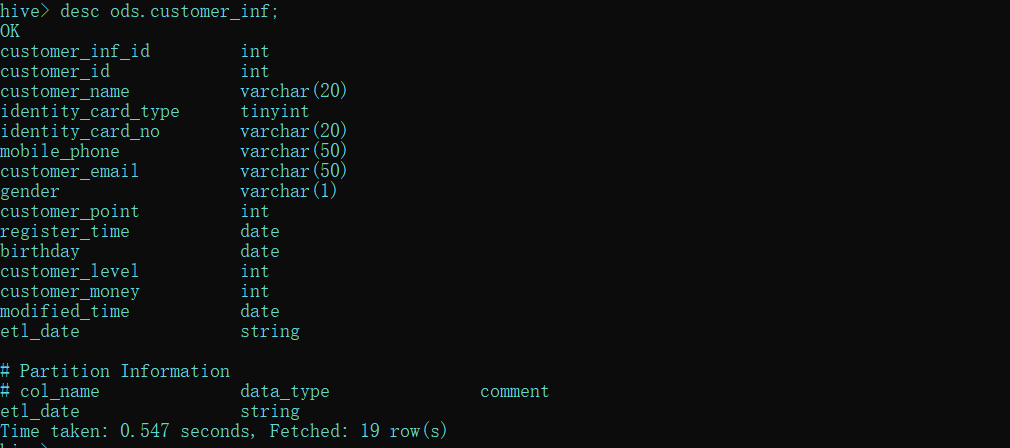
### 任务二：数据清洗

**提示**：若完成了实时计算的任务二中的第3小题后，可以将HBase中备份的数据与ods中的离线数据进行合并，抽取到dwd（需自建）的对应表中。

编写Hive SQL或者Spark Sql代码，将ods库中相应表数据（经过数据抽取得数据）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，若与日期有关的数据，必须转为timestamp，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。

1. 抽取ods库中customer\_inf表中昨天的分区（任务一生成的分区）数据，并结合dim\_customer\_inf最新分区现有的数据，根据customer\_id合并数据到dwd库中dim\_customer\_inf的分区表（合并是指对dwd层数据进行插入或修改，需修改的数据以customer\_id为合并字段，根据modified\_time排序取最新的一条），分区字段为etl\_date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”。若该条记录第一次进入数仓dwd层则dwd\_insert\_time、dwd\_modify\_time均存当前操作时间，并进行数据类型转换。若该数据在进入dwd层时发生了合并修改，则dwd\_insert\_time时间不变，dwd\_modify\_time存当前操作时间，其余列存最新的值。使用hive cli执行show partitions dwd.dim\_customer\_inf命令；

* 首先查看表结构



* 根据表结构将新库与表新建 注意 需要指定默认值

create database dwd;

use dwd;  
create table customer\_inf  
(  
 customer\_inf\_id int,  
 customer\_id int,  
 customer\_name varchar(20),  
 identity\_card\_type tinyint,  
 identity\_card\_no varchar(20),  
 mobile\_phone varchar(50),  
 customer\_email varchar(50),  
 gender varchar(1),  
 customer\_point int,  
  
 register\_time date,  
 birthday date,  
 customer\_level int,  
 customer\_money int,  
 modified\_time date,  
  
 dwd\_insert\_user string default 'user1',  
 dwd\_insert\_time date default `current\_date`(),  
 dwd\_modify\_user string default 'user1',  
 dwd\_modify\_time date default `current\_date`()  
)  
 partitioned by (etl\_date String)  
 row format delimited fields terminated by ',';

* 使用HQL将时间格式转换成为 yyyy-MM-dd HH:mm:ss 格式，并输出到新表中

insert into dwd.customer\_inf (customer\_inf\_id, customer\_id,

customer\_name, identity\_card\_type,  
 identity\_card\_no, mobile\_phone,  
 customer\_email, gender, customer\_point,  
 register\_time, birthday, customer\_level,  
 customer\_money, modified\_time, etl\_date)  
 -- 先插入 ods 库中的表中不存在于 dwd 层的行  
select o.customer\_inf\_id,  
 o.customer\_id,  
 o.customer\_name,  
 o.identity\_card\_type,  
 o.identity\_card\_no,  
 o.mobile\_phone,  
 o.customer\_email,  
 o.gender,  
 o.customer\_point,  
 date\_format(o.register\_time, 'yyyy-MM-dd HH:mm:ss'),  
 date\_format(o.birthday, 'yyyy-MM-dd HH:mm:ss'),  
 o.customer\_level,  
 o.customer\_money,  
 date\_format(o.modified\_time, 'yyyy-MM-dd HH:mm:ss'),  
 date\_format(o.etl\_date, 'yyyy-MM-dd HH:mm:ss')  
from ods.customer\_inf as o  
 left join dwd.customer\_inf as d on o.customer\_id == d.customer\_id and o.customer\_id IS NULL;  
  
insert into dwd.customer\_inf (customer\_inf\_id, customer\_id,  
 customer\_name, identity\_card\_type,  
 identity\_card\_no, mobile\_phone,  
 customer\_email, gender, customer\_point,  
 register\_time, birthday, customer\_level,  
 customer\_money, modified\_time, etl\_date,  
 dwd\_insert\_time)  
 -- 然后插入 ods 库中的表中存在于 dwd 层的行 这时候，其中的插入时间不再是当前时间了，而是 ods 中的插入时间。  
select o.customer\_inf\_id,  
 o.customer\_id,  
 o.customer\_name,  
 o.identity\_card\_type,  
 o.identity\_card\_no,  
 o.mobile\_phone,  
 o.customer\_email,  
 o.gender,  
 o.customer\_point,  
 date\_format(o.register\_time, 'yyyy-MM-dd HH:mm:ss'),  
 date\_format(o.birthday, 'yyyy-MM-dd HH:mm:ss'),  
 o.customer\_level,  
 o.customer\_money,  
 date\_format(o.modified\_time, 'yyyy-MM-dd HH:mm:ss'),  
 date\_format(o.etl\_date, 'yyyy-MM-dd HH:mm:ss'),  
 d.etl\_date  
from ods.customer\_inf as o  
 left join dwd.customer\_inf as d on o.customer\_id == d.customer\_id and o.customer\_id is not null;

1. 抽取ods库中product\_info表中昨天的分区（任务一生成的分区）数据，并结合dim\_product\_info最新分区现有的数据，根据product\_id合并数据到dwd库中dim\_product\_info的分区表（合并是指对dwd层数据进行插入或修改，需修改的数据以product\_id为合并字段，根据modified\_time排序取最新的一条），分区字段为etl\_date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”。若该条记录第一次进入数仓dwd层则dwd\_insert\_time、dwd\_modify\_time均存当前操作时间，并进行数据类型转换。若该数据在进入dwd层时发生了合并修改，则dwd\_insert\_time时间不变，dwd\_modify\_time存当前操作时间，其余列存最新的值。使用hive cli执行show partitions dwd.dim\_product\_info命令；
2. 将ods库中order\_master表昨天的分区（任务一生成的分区）数据抽取到dwd库中fact\_order\_master的动态分区表，分区字段为etl\_date，类型为String，取create\_time值并将格式转换为yyyyMMdd，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，需要过滤掉city字段长度大于8。使用hive cli执行show partitions dwd.fact\_order\_info命令；

**提示**：若完成实时任务二中的第3小题后，可将hbase中的数据合并后在进行清洗，清洗后数据添加至DWD中

1. 将ods库中order\_detail表昨天的分区（任务一生成的分区）数据抽取到dwd库中fact\_order\_detail的动态分区表，分区字段为etl\_date，类型为String，取create\_time值并将格式转换为yyyyMMdd，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。使用hive cli执行show partitions dwd.fact\_order\_detail命令；

**提示**：若完成实时任务二中的第3小题后，可将hbase中的数据合并后在进行清洗，清洗后数据添加至DWD中

### 任务三：指标计算

**注：**与订单金额计算相关使用order\_money字段，同一个订单无需多次重复计算，需要考虑退款或者取消的订单

1. 根据dwd或者dws层表统计每人每天下单的数量和下单的总金额，存入dws层（需自建）的user\_consumption\_day\_aggr表中（表结构如下），然后使用hive cli按照客户主键、订单总金额均为降序排序，查询出前5条；

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 中文含义 | 备注 |
| customer\_id | int | 客户主键 | customer\_id |
| customer\_name | string | 客户名称 | customer\_name |
| total\_amount | double | 订单总金额 | 当天订单总金额 |
| total\_count | int | 订单总数 | 当天订单总数 |
| year | int | 年 | 订单产生的年,为动态分区字段 |
| month | int | 月 | 订单产生的月,为动态分区字段 |
| day | int | 日 | 订单产生的日,为动态分区字段 |

* 创建出 dws 层

-- 创建出 dws 数据库

create database dws;  
use dws;  
-- 创建出 user\_consumption\_day\_aggr 数据表  
create table user\_consumption\_day\_aggr  
(  
 customer\_id int,  
 customer\_name string,  
 total\_amount double,  
 total\_count int,  
 year int,  
 month int,  
 day int  
);

* 使用HQL开始进行数据的处理

-- 统计 ods.customer\_inf 每人每天下单的数量和下单的总金额 存储进入

user\_consumption\_day\_aggr  
  
insert into user\_consumption\_day\_aggr  
select customer\_id,  
 customer\_name,  
 sum(customer\_money),  
 count(\*),  
 year(register\_time),  
 month(register\_time),  
 day(register\_time)  
from ods.customer\_inf  
group by customer\_id, customer\_name, register\_time

1. 根据dwd或者dws层表统计每个城市每月下单的数量和下单的总金额（以order\_master中的地址为判断依据），并按照province\_name，year，month进行分组,按照total\_amount逆序排序，形成sequence值，将计算结果存入Hive的dws数据库city\_consumption\_day\_aggr表中（表结构如下），然后使用hive cli根据订单总数、订单总金额均为降序排序，查询出前5条，在查询时对于订单总金额字段将其转为bigint类型（避免用科学计数法展示）；

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 中文含义 | 备注 |
| city\_name | string | 城市名称 |  |
| province\_name | string | 省份名称 |  |
| total\_amount | double | 订单总金额 | 当月订单总金额 |
| total\_count | int | 订单总数 | 当月订单总数 |
| sequence | int | 次序 | 即当月中该城市消费额在该省中的排名 |
| year | int | 年 | 订单产生的年,为动态分区字段 |
| month | int | 月 | 订单产生的月,为动态分区字段 |

1. 请根据dwd或者dws层表计算出每个城市每个月平均订单金额和该城市所在省份平均订单金额相比较结果（“高/低/相同”）,存入ClickHouse数据库shtd\_result的cityavgcmpprovince表中（表结构如下），然后在Linux的ClickHouse命令行中根据城市平均订单金额、省份平均订单金额均为降序排序，查询出前5条；

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 中文含义 | 备注 |
| cityname | text | 城市份名称 |  |
| cityavgconsumption | double | 该城市平均订单金额 |  |
| provincename | text | 省份名称 |  |
| provinceavgconsumption | double | 该省平均订单金额 |  |
| comparison | text | 比较结果 | 城市平均订单金额和该省平均订单金额比较结果，值为：高/低/相同 |

1. 请根据dwd或者dws层表计算出每个城市每个月平均订单金额和该城市所在省份订单金额中位数相比较结果（“高/低/相同”）,存入ClickHouse数据库shtd\_result的citymidcmpprovince表中（表结构如下），然后在Linux的ClickHouse命令行中根据城市平均订单金额、省份平均订单金额均为降序排序，查询出前5条；

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 中文含义 | 备注 |
| cityname | text | 城市份名称 |  |
| citymidconsumption | double | 该城市订单金额中位数 |  |
| provincename | text | 省份名称 |  |
| provincemidconsumption | double | 该省订单金额中位数 |  |
| comparison | text | 比较结果 | 城市订单金额中位数和该省订单金额中位数比较结果，值为：高/低/相同 |

1. 请根据dwd或者dws层表来计算每个省份2022年订单金额前3省份，依次存入ClickHouse数据库shtd\_result的regiontopthree表中（表结构如下），然后在Linux的ClickHouse命令行中根据省份升序排序，查询出前5条；

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 中文含义 | 备注 |
| provincename | text | 省份名称 |  |
| citynames | text | 城市名称 | 用,分割显示前三城市的name |
| cityamount | text | 省份名称 | 用,分割显示前三城市的订单金额（需要去除小数部分，使用四舍五入） |

例如：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | 华东地区 | 21,27,11 | 上海市,江苏省,浙江省 | 100000,100,10 |

1. 请根据dwd或者dws层的相关表，计算销售量前10的商品，销售额前10的商品，存入ClickHouse数据库shtd\_result的topten表中（表结构如下），然后在Linux的ClickHouse命令行中根据排名升序排序，查询出前5条;

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 中文含义 | 备注 |
| topquantityid | int | 商品id | 销售量前10的商品 |
| topquantityname | text | 商品名称 | 销售量前10的商品 |
| topquantity | int | 该商品销售量 | 销售量前10的商品 |
| toppriceid | text | 商品id | 销售额前10的商品 |
| toppricename | text | 商品名称 | 销售额前10的商品 |
| topprice | decimal | 该商品销售额 | 销售额前10的商品 |
| sequence | int | 排名 | 所属排名 |

1. 请根据dwd或者dws层的数据，请计算连续两天下单的用户与已下单用户的占比，将结果存入ClickHouse数据库shtd\_result的userrepurchasedrate表中(表结构如下)，然后在Linux的ClickHouse命令行中查询结果数据；

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 中文含义 | 备注 |
| purchaseduser | int | 下单人数 | 已下单人数 |
| repurchaseduser | int | 连续下单人数 | 连续两天下单的人数 |
| repurchaserate | text | 百占比 | 连续两天下单人数/已下单人数百分比（保留1位小数，四舍五入，不足的补0）例如21.1%，或者32.0% |

1. 根据dwd或者dws层的数据，请计算每个省份累计订单量，然后根据每个省份订单量从高到低排列，将结果打印到控制台（使用spark中的show算子，同时需要显示列名）；

例如：可以考虑首先生成类似的临时表A：

|  |  |
| --- | --- |
| province\_name | Amount（订单量） |
| A省 | 10122 |
| B省 | 301 |
| C省 | 2333333 |

然后生成结果类似如下：其中C省销量最高，排在第一列，A省次之，以此类推。

|  |  |  |
| --- | --- | --- |
| C省 | A省 | B省 |
| 23333331 | 10122 | 301 |

1. 根据dwd或者dws层的相关表，请计算2022年4月26日凌晨0点0分0秒到早上9点59分59秒为止，该时间段每小时的新增订单金额与当天订单总金额累加值，存入ClickHouse数据库shtd\_result的accumulateconsumption表中，然后在Linux的ClickHouse命令行中根据订单时间段升序排序，查询出前5条;

假如数据为：

|  |  |  |
| --- | --- | --- |
| 用户 | 订单时间 | 订单金额 |
| 张三1号 | 2020-04-26 00:00:10 | 10 |

|  |  |  |
| --- | --- | --- |
| 李四1号 | 2020-04-26 00:20:10 | 5 |
| 李四2号 | 2020-04-26 01:21:10 | 10 |
| 王五1号 | 2020-04-26 03:20:10 | 50 |

计算结果则为：

|  |  |  |
| --- | --- | --- |
| 订单时间段 | 订单新增金额 | 累加总金额 |
| 2020-04-26 00 | 15 | 15 |
| 2020-04-26 01 | 10 | 25 |

|  |  |  |
| --- | --- | --- |
| 2020-04-26 02 | 0 | 25 |

|  |  |  |
| --- | --- | --- |
| 2020-04-26 03 | 50 | 75 |

accumulateconsumption表结构如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 中文含义 | 备注 |
| consumptiontime | varchar | 消费时间段 |  |
| consumptionadd | double | 订单新增金额 |  |
| consumptionacc | double | 累加总金额 |  |

1. 根据dwd层或dws层的相关表，请计算2022年4月26日凌晨0点0分0秒到早上9点59分59秒为止的数据，以5个小时为时间窗口，滑动的步长为1小时，做滑动窗口计算该窗口内订单总金额和订单总量，时间不满5小时不触发计算（即从凌晨5点0分0秒开始触发计算），存入ClickHouse数据库shtd\_result的slidewindowconsumption表中，然后在Linux的ClickHouse命令行中根据订单时间段升序排序，查询出前5条，将核心业务代码中的开窗相关代码与MySQL查询结果展示出来。

假如数据为：

|  |  |  |
| --- | --- | --- |
| 用户 | 订单时间 | 订单金额 |
| 张三1号 | 2020-04-26 00:00:10 | 10 |

|  |  |  |
| --- | --- | --- |
| 李四1号 | 2020-04-26 00:20:10 | 25 |
| 李四2号 | 2020-04-26 01:21:10 | 10 |
| 李四2号 | 2020-04-26 02:21:10 | 5 |
| 王五1号 | 2020-04-26 03:20:10 | 20 |
| 李四2号 | 2020-04-26 04:20:10 | 10 |
| 王五2号 | 2020-04-26 05:10:10 | 10 |
| 李四2号 | 2020-04-26 06:20:10 | 10 |
| 赵六2号 | 2020-04-26 07:10:10 | 10 |
| 赵六2号 | 2020-04-26 08:10:10 | 10 |
| 王五2号 | 2020-04-26 09:11:10 | 10 |
| 王五4号 | 2020-04-26 09:32:10 | 30 |

计算结果则为：

|  |  |  |  |
| --- | --- | --- | --- |
| 订单时间段 | 该窗口内订单金额 | 订单总量 | 平均每单价格 |
| 2020-04-26 04 | 80 | 6 | 13.33 |
| 2020-04-26 05 | 55 | 5 | 11 |
| 2020-04-26 06 | 55 | 5 | 11 |
| 2020-04-26 07 | 60 | 5 | 12 |
| 2020-04-26 08 | 50 | 5 | 10 |
| 2020-04-26 09 | 80 | 6 | 13.33 |

slidewindowconsumption表结构如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 中文含义 | 备注 |
| consumptiontime | varchar | 订单时间段 |  |
| consumptionsum | double | 该窗口内的订单总金额 |  |
| consumptioncount | double | 订单总数量 |  |
| consumptionavg | double | 平均每单价格 | 上面两个字段相除，四舍五入保留两位小数 |

## 数据挖掘

**环境说明：**

|  |
| --- |
| 该模块均使用Scala编写，利用Spark与Flink相关库完成。   * Spark可用在离线数据的推荐 * Flink可用在实时数据的推荐 |

### 任务一：特征工程（一）

剔除订单主表与订单详情表中用户id与商品id不存在现有的维表中的记录，同时建议多利用缓存并充分考虑并行度来优化代码，达到更快的计算效果。

1. 根据Hive的dwd库中相关表或MySQL中ds\_db01中相关表，计算出与用户customer\_id为5811的用户所购买相同商品种类最多的前10位用户（只考虑他俩购买过多少种相同的商品，不考虑相同的商品买了多少次），将10位用户customer\_id进行输出，输出格式如下；

结果格式如下：

-------------------相同种类前10的id结果展示为：--------------------

1,2,901,4,5,21,32,91,14,52

1. 根据Hive的dwd库中相关表或MySQL中ds\_db01中相关商品表（sku\_info），获取id、brand\_id、price、weight、height、length、width、three\_category\_id 字段并进行数据预处理，对数值类型进行规范化(StandardScaler)处理，对类别类型进行one-hot编码处理（若该商品属于该品牌则置为1，否则置为0）,并按照id进行升序排序，在集群中输出第一条数据前10列（无需展示字段名）。

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 中文含义 | 备注 |
| id | double | 主键 |  |
| price | double | 价格 |  |
| weight | double | 重量 |  |
| 。。。 | 。。。 | 。。。 |  |
| brand\_id#1 | double | brand\_id 1 | 若属于该品牌，则内容为1否则为0 |
| brand\_id#2 | double | brand\_id 2 | 若属于该品牌，则内容为1否则为0 |
| ..... | double |  |  |

结果格式如下：

--------------------第一条数据前10列结果展示为：---------------------

1.0,0.892346,1.72568,0.0,0.0,0.0,0.0,1.0,0.0,0.0

### 任务二：推荐系统（一）

1. 根据上述任务的结果，计算出与用户customer\_id为5811的用户所购买相同商品种类最多的前10位用户id（只考虑他俩购买过多少种相同的商品，不考虑相同的商品买了多少次），并根据Hive的dwd库中相关表或MySQL数据库shtd\_store中相关表，获取到这10位用户已购买过的商品，并剔除用户5811已购买的商品，通过计算这10位用户已购买的商品（剔除用户5811已购买的商品）与用户5811已购买的商品数据集中商品的余弦相似度累加再求均值，输出均值前5商品id作为推荐使用。

结果格式如下：

------------------------推荐Top5结果如下------------------------

相似度top1(商品id：1，平均相似度：0.983456)

相似度top2(商品id：71，平均相似度：0.782672)

相似度top3(商品id：22，平均相似度：0.7635246)

相似度top4(商品id：351，平均相似度：0.7335748)

相似度top5(商品id：14，平均相似度：0.522356)

### 任务三：特征工程（二）

1. 据Hive的dwd库中相关表或MySQL数据库ds\_db01中订单相关表，对用户购买过的商品进行去重，将其转换为以下格式：第一列为用户id mapping，第二列为用户购买过的商品id mapping，按照customer\_id与product\_id进行升序排序，输出前5行；

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 中文含义 | 备注 |
| customer\_id | int | 用户id的mapping对应键 |  |
| product\_id | int | 商品id的mapping对应键 |  |

提示：

Mapping操作：例如用户id：1、4、7、8、9，则做完mapping操作转为字典类型，键0对应用户id 1，键1对应用户id 4，以此类推

结果格式如下：

------- customer\_id\_mapping与product\_id\_mapping数据前5条如下：-------

0:0

0:89

1:1

1:2

1:3

1. 根据第1小题的结果，对其进行聚合，其中对product\_id进行one-hot转换，将其转换为以下格式矩阵：第一列为用户id，其余列名为商品id，按照用户id进行升序排序，展示矩阵第一行前5列数据。

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 中文含义 | 备注 |
| customer\_id | double | 客户id |  |
| product\_id0 | double | 用户是否购买过商品1 | 若用户购买过该商品，则值为1，否则为0 |
| product\_id1 | double | 用户是否购买过商品2 | 若用户购买过该商品，则值为1，否则为0 |
| product\_id2 | double | 用户是否购买过商品3 | 若用户购买过该商品，则值为1，否则为0 |
| ..... |  |  |  |

结果格式如下：

---------------第一行前5列结果展示为---------------

0.0,1.0,0.0,0.0,0.0

### 任务四：推荐系统（二）

1. 根据上述任务的结果，对其进行SVD分解，对数据进行降维保留至少保留前95%的信息量，根据该用户customer\_id为5811的用户已购买的商品分别与未购买的商品计算余弦相似度再进行累加求均值，将均值最大的5件商品id进行输出作为推荐使用。将结果输出。

结果格式如下：

------------------------推荐Top5结果如下------------------------

相似度top1(商品id：1，平均相似度：0.983456)

相似度top2(商品id：71，平均相似度：0.782672)

相似度top3(商品id：22，平均相似度：0.7635246)

相似度top4(商品id：351，平均相似度：0.7335748)

相似度top5(商品id：14，平均相似度：0.522356)

## 数据采集与实时计算

**环境说明：**

|  |
| --- |
| Flink任务在Yarn上用per job模式（即Job分离模式，不采用Session模式），方便Yarn回收资源。 |

### 任务一：实时数据采集

1. 在Master节点使用Flume采集实时数据生成器25001端口的socket数据（实时数据生成器脚本为Master节点/data\_log目录下的gen\_ds\_data\_to\_socket脚本，该脚本为Master节点本地部署且使用socket传输），将数据存入到Kafka的Topic中（Topic名称为ods\_mall\_log，分区数为4），使用Kafka自带的消费者消费ods\_mall（Topic）中的数据，查看前2条数据的结果；

**注：需先启动已配置好的Flume再启动脚本，否则脚本将无法成功启动，启动方式为进入/data\_log目录执行./gen\_ds\_data\_to\_socket (如果没有权限，请执行授权命令chmod 777 /data\_log/gen\_ds\_data\_to\_socket)**

1. 在Master节点进入到maxwell-1.29.0的解压后目录下（在/opt/module），配置相关文件并启动，读取MySQL数据的binlog日志（mysql的binlog相关配置已完毕）到Kafka的Topic中（Topic名称为ods\_mall\_data，分区数为4）。使用Kafka自带的消费者消费ods\_mall\_data（Topic）中的数据，查看前2条数据的结果；

### 任务二：使用Flink处理Kafka中的数据

**注：**与订单金额计算相关使用order\_money字段，同一个订单无需多次重复计算，需要考虑退款或者取消的订单。

编写Scala代码，使用Flink消费Kafka中Topic为ods\_mall\_log和ods\_mall\_data的数据并进行相应的数据统计计算(使用ProcessTime）。

1. 使用Flink消费Kafka中topic为ods\_mall\_data的数据，根据数据中不同的表将数据分别分发至kafka的DWD层的fact\_order\_master、fact\_order\_detail的Topic中（只获取data的内容，具体的内容格式考生请自查），其他的表则无需处理，其中请对进入到fact\_order\_master的内容进行排序，同时迟到容忍度设置为2分钟；
2. 使用Flink消费Kafka中topic为ods\_mall\_log的数据，根据数据中不同的表前缀区分，将数据分别分发至kafka的DWD层的dim\_customer\_login\_log的Topic中，其他的表则无需处理；
3. 在任务1，2进行的同时，需要将order\_master、order\_detail、customer\_login\_log备份至Hbase中，同时建立Hive外表(表结构与离线数据ods层一致，timestamp格式可用string或者long代替，rowkey使用随机数（0-9）+yyyyMMddHHmmssSSS，其中对于customer\_login\_log缺失主键，请用随机数（0-9）+用户id+登陆时间代替)，同时在Hive中查询每个表前5条数据；
4. 使用Flink消费Kafka的dwd层数据，统计商城实时订单数量，将key设置成totalcount存入Redis中。使用redis cli以get key方式获取totalcount值；
5. 使用Flink消费Kafka的dwd层数据，实时统计商城中销售量前3的商品，将key设置成top3itemamount存入Redis中（value使用String数据格式，value为前3的商品信息并且外层用[]包裹，其中按排序依次存放商品id:销售量，并用逗号分割）。使用redis cli以get key方式获取top3itemamount值；

示例如下：

top3itemamount：[1:700,42:500,41:100]

1. 使用Flink消费Kafka的dwd层数据，实时统计商城中销售额前3的商品（销售额为单价\*数量），将key设置成top3itemconsumption存入Redis中（value使用String数据格式，value为前3的商品信息并且外层用[]包裹，其中按排序依次存放商品id:销售额，并用逗号分割）。使用redis cli以get key方式获取top3itemconsumption值。

示例如下：

top3itemconsumption：[1:10020.2,42:4540.0,12:540]

1. 使用Flink消费Kafka的dwd层数据，监控order\_status字段为已退款的数据,将数据存入ClickHouse数据库shtd\_result的order\_master表中，然后在Linux的ClickHouse命令行中根查询出前5条。
2. 使用Flink消费Kafka中的数据，实时统计商城中消费额前2的用户（，将key设置成top2userconsumption存入Redis中（value使用String数据格式，value为前2的用户信息并且外层用[]包裹，其中按排序依次存放为该用户id:用户名称:消费总额，用逗号分割，其中用户名称可从HBase或者Hive或者Kafka中的客户表中获取）。使用redis cli以get key方式获取top2userconsumption值；

示例如下：

top2userconsumption：[1:张三:10020,42:李四:4540]

1. 采用双流JOIN的方式（本系统稳定，无需担心数据迟到与丢失的问题,建议使用滚动窗口），结合订单信息和订单详细信息，拼接成如下表所示格式，其中包含订单id、订单总金额、商品数，将数据存入ClickHouse数据库shtd\_result的orderpostiveaggr表中（表结构如下），然后在Linux的ClickHouse命令行中根据sn降序排序，查询出前5条。

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 中文含义 | 备注 |
| sn | int | 订单sn |  |
| orderprice | double | 订单总金额 | 统计订单主表中 order\_money字段 |
| orderdetailcount | int | 商品数量总计 | 累加订单详细信息中 product\_cnt字段 |