第四天笔记

Hive 常用函数

关系运算

```
// 等值比较 = == <=>
// 不等值比较 != <>
// 区间比较： select * from default.students where id between 1500100001 and
1500100010;
// 空值/非空值判断：is null、is not null、nvl()、isnull()
// like、rlike、regexp用法
```

数值计算

```
取整函数(四舍五入)：round(目标列,要保留的小数位)
向上取整：ceil
向下取整：floor
```

条件函数

- if： if(表达式,如果表达式成立的返回值,如果表达式不成立的返回值)

```
select if(1>0,1,0);
select if(1>0,if(-1>0,-1,1),0);
select score,if(score>120,'优秀',if(score>100,'良好',if(score>90,'及格','不及格')))
as pingfen from score limit 20;
```

- COALESCE

```
select COALESCE(null,'1','2'); // 1 从左往右 一次匹配 直到非空为止
select COALESCE('1',null,'2'); // 1
```

- case when

```
select  score
        ,case when score>120 then '优秀'
              when score>100 then '良好'
              when score>90 then '及格'
        else '不及格'
        end as pingfen
from score limit 20;

select  score
        ,case when score>120 then id
              when score>100 then '良好'
              when score>90 then '及格'
        else '不及格'
        end as pingfen
from score limit 20;

select  name
        ,case name when "施笑槐" then "槐ge"
                   when "吕金鹏" then "鹏ge"
                   when "单乐蕊" then "蕊jie"
        else "算了不叫了"
        end as nickname
from students limit 10;
```

> 注意条件的顺序

日期函数

```
select from_unixtime(1610611142,'YYYY/MM/dd HH:mm:ss');

select from_unixtime(unix_timestamp(),'YYYY/MM/dd HH:mm:ss');

// '2021年01月14日' -> '2021-01-14'
select from_unixtime(unix_timestamp('2021年01月14日','yyyy年MM月dd日'),'yyyy-MM-
dd');
// "04牛2021数加16逼" -> "2021/04/16"
select from_unixtime(unix_timestamp("04牛2021数加16逼","MM牛yyyy数加dd
逼"),"yyyy/MM/dd");
```

字符串函数

```
concat('123','456'); // 123456
concat('123','456',null); // NULL

select concat_ws('#','a','b','c'); // a#b#c
select concat_ws('#','a','b','c',NULL); // a#b#c 可以指定分隔符，并且会自动忽略NULL
select concat_ws("|",cast(id as string),name,cast(age as string),gender,clazz)
from students limit 10;

select substring("abcdefg",1); // abcdefg HQL中涉及到位置的时候 是从1开始计数
// '2021/01/14' -> '2021-01-14'
select concat_ws("-
",substring('2021/01/14',1,4),substring('2021/01/14',6,2),substring('2021/01/14'
,9,2));
// 建议使用日期函数去做日期
select from_unixtime(unix_timestamp('2021/01/14','yyyy/MM/dd'),'yyyy-MM-dd');

select split("abcde,fgh",","); // ["abcde","fgh"]
select split("a,b,c,d,e,f",",")[2]; // c

select explode(split("abcde,fgh",",")); // abcde
                                        //  fgh

// 解析json格式的数据
select get_json_object('{"name":"zhangsan","age":18,"score":
[{"course_name":"math","score":100},
{"course_name":"english","score":60}]}',"$.score[0].score"); // 100
```

Hive 中的wordCount

```
create table words(
    words string
)row format delimited fields terminated by '|';

// 数据
hello,java,hello,java,scala,python
hbase,hadoop,hadoop,hdfs,hive,hive
hbase,hadoop,hadoop,hdfs,hive,hive

select word,count(*) from (select explode(split(words,',')) word from words) a
group by a.word;

// 结果
hadoop  4
```

```
hbase   2
hdfs    2
hello   2
hive    4
java    2
python  1
scala   1
```

Hive 开窗函数

> 好像给每一份数据 开一扇窗户 所以叫开窗函数
>
> 在sql中有一类函数叫做聚合函数,例如sum()、avg()、max()等等,这类函数可以将多行数据按照规则聚集为一行,一般来讲聚集后的行数是要少于聚集前的行数的.但是有时我们想要既显示聚集前的数据,又要显示聚集后的数据,这时我们便引入了窗口函数.

测试数据

```
111,69,class1,department1
112,80,class1,department1
113,74,class1,department1
114,94,class1,department1
115,93,class1,department1
121,74,class2,department1
122,86,class2,department1
123,78,class2,department1
124,70,class2,department1
211,93,class1,department2
212,83,class1,department2
213,94,class1,department2
214,94,class1,department2
215,82,class1,department2
216,74,class1,department2
221,99,class2,department2
222,78,class2,department2
223,74,class2,department2
224,80,class2,department2
225,85,class2,department2
```

建表语句

```
create table new_score(
    id  int
    ,score int
    ,clazz string
    ,department string
) row format delimited fields terminated by ",";
```

row_number：无并列排名

- 用法： select xxxx, row_number() over(partition by 分组字段 order by 排序字段 desc) as rn from tb group by xxxx

dense_rank：有并列排名，并且依次递增

rank：有并列排名，不依次递增

percent_rank：(rank的结果-1)/(分区内数据的个数-1)

cume_dist：计算某个窗口或分区中某个值的累积分布。

> 假定升序排序，则使用以下公式确定累积分布： 小于等于当前值x的行数 / 窗口或partition分区内的总行数。其中，x 等于 order by 子句中指定的列的当前行中的值。

NTILE(n)：对分区内数据再分成n组，然后打上组号

max、min、avg、count、sum：基于每个partition分区内的数据做对应的计算

窗口帧：用于从分区中选择指定的多条记录，供窗口函数处理

> Hive 提供了两种定义窗口帧的形式：`ROWS` 和 `RANGE`。两种类型都需要配置上界和下界。例如，`ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW` 表示选择分区起始记录到当前记录的所有行；`SUM(close) RANGE BETWEEN 100 PRECEDING AND 200 FOLLOWING` 则通过 字段差值来进行选择。如当前行的 `close` 字段值是 `200`，那么这个窗口帧的定义就会选择分区中 `close` 字段值落在 `100` 至 `400` 区间的记录。以下是所有可能的窗口帧定义组合。如果没有定义窗口帧，则默认为 `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`。
>
> 只能运用在max、min、avg、count、sum、FIRST_VALUE、LAST_VALUE这几个窗口函数上

```
(ROWS | RANGE) BETWEEN (UNBOUNDED | [num]) PRECEDING AND ([num] PRECEDING |
CURRENT ROW | (UNBOUNDED | [num]) FOLLOWING)
(ROWS | RANGE) BETWEEN CURRENT ROW AND (CURRENT ROW | (UNBOUNDED | [num])
FOLLOWING)
(ROWS | RANGE) BETWEEN [num] FOLLOWING AND (UNBOUNDED | [num]) FOLLOWING
range between 3 PRECEDING and 11 FOLLOWING
```

```
SELECT id
     ,score
     ,clazz
     ,SUM(score) OVER w as sum_w
     ,round(avg(score) OVER w,3) as avg_w
     ,count(score) OVER w as cnt_w
FROM new_score
WINDOW w AS (PARTITION BY clazz ORDER BY score rows between 2 PRECEDING and
2 FOLLOWING);
```

```
111 69  class1  217 72.333  3
113 74  class1  297 74.25   4
216 74  class1  379 75.8    5
112 80  class1  393 78.6    5
215 82  class1  412 82.4    5
212 83  class1  431 86.2    5
211 93  class1  445 89.0    5
115 93  class1  457 91.4    5
213 94  class1  468 93.6    5
114 94  class1  375 93.75   4
214 94  class1  282 94.0    3
124 70  class2  218 72.667  3
121 74  class2  296 74.0    4
223 74  class2  374 74.8    5
```

```
222 78  class2  384 76.8    5
123 78  class2  395 79.0    5
224 80  class2  407 81.4    5
225 85  class2  428 85.6    5
122 86  class2  350 87.5    4
221 99  class2  270 90.0    3
```

```
select  id
        ,score
        ,clazz
        ,department
        ,row_number() over (partition by clazz order by score desc) as rn_rk
        ,dense_rank() over (partition by clazz order by score desc) as dense_rk
        ,rank() over (partition by clazz order by score desc) as rk
        ,percent_rank() over (partition by clazz order by score desc) as
percent_rk
        ,round(cume_dist() over (partition by clazz order by score desc),3) as
cume_rk
        ,NTILE(3) over (partition by clazz order by score desc) as ntile_num
        ,max(score) over (partition by clazz order by score desc range between 3
PRECEDING and 11 FOLLOWING) as max_p
from new_score;


id  score   clazz   department  rn_rk  ds_rk  rk  percent_rk  cume_rk ntile_num
max_p
114  94     class1  department1   1      1     1      0.0       0.273      1
94
214  94     class1  department2   2      1     1      0.0       0.273      1
94
213  94     class1  department2   3      1     1      0.0       0.273      1
94
211  93     class1  department2   4      2     4      0.3       0.455      1
94
115  93     class1  department1   5      2     4      0.3       0.455      2
94
212  83     class1  department2   6      3     6      0.5       0.545      2
83
215  82     class1  department2   7      4     7      0.6       0.636      2
83
112  80     class1  department1   8      5     8      0.7       0.727      2
83
113  74     class1  department1   9      6     9      0.8       0.909      3
74
216  74     class1  department2   10     6     9      0.8       0.909      3
74
111  69     class1  department1   11     7     11     1.0       1.0        3
69
221  99     class2  department2   1      1     1      0.0       0.111      1
99
122  86     class2  department1   2      2     2      0.125     0.222      1
86
225  85     class2  department2   3      3     3      0.25      0.333      1
86
```

```
224  80     class2  department2   4       4      4       0.375      0.444       2
80
123  78     class2  department1   5       5      5       0.5        0.667       2
80
222  78     class2  department2   6       5      5       0.5        0.667       2
80
121  74     class2  department1   7       6      7       0.75       0.889       3
74
223  74     class2  department2   8       6      7       0.75       0.889       3
74
124  70     class2  department1   9       7      9       1.0        1.0         3
70
```

LAG(col,n)：往前第n行数据

LEAD(col,n)：往后第n行数据

FIRST_VALUE：取分组内排序后，截止到当前行，第一个值

LAST_VALUE：取分组内排序后，截止到当前行，最后一个值，对于并列的排名，取最后一个

```
select  id
        ,score
        ,clazz
        ,department
        ,lag(id,2) over (partition by clazz order by score desc) as lag_num
        ,LEAD(id,2) over (partition by clazz order by score desc) as lead_num
        ,FIRST_VALUE(id) over (partition by clazz order by score desc) as
first_v_num
        ,LAST_VALUE(id) over (partition by clazz order by score desc) as
last_v_num
        ,NTILE(3) over (partition by clazz order by score desc) as ntile_num
from new_score;


id  score   clazz   department  lag_num lead_num  first_v_num last_v_num
ntile_num
114  94     class1  department1   NULL     213      114           213         1
214  94     class1  department2   NULL     211      114           213         1
213  94     class1  department2   114      115      114           213         1
211  93     class1  department2   214      212      114           115         1
115  93     class1  department1   213      215      114           115         2
212  83     class1  department2   211      112      114           212         2
215  82     class1  department2   115      113      114           215         2
112  80     class1  department1   212      216      114           112         2
113  74     class1  department1   215      111      114           216         3
216  74     class1  department2   112      NULL     114           216         3
111  69     class1  department1   113      NULL     114           111         3
221  99     class2  department2   NULL     225      221           221         1
122  86     class2  department1   NULL     224      221           122         1
225  85     class2  department2   221      123      221           225         1
224  80     class2  department2   122      222      221           224         2
123  78     class2  department1   225      121      221           222         2
222  78     class2  department2   224      223      221           222         2
121  74     class2  department1   123      124      221           223         3
223  74     class2  department2   222      NULL     221           223         3
124  70     class2  department1   121      NULL     221           124         3
```

> https://blog.csdn.net/qq_26937525/article/details/54925827

Hive 行转列

lateral view explode

```
create table testArray2(
    name string,
    weight array<string>
)row format delimited
fields terminated by '\t'
COLLECTION ITEMS terminated by ',';

志凯  "150","170","180"
上单  "150","180","190"



select name,col1  from testarray2 lateral view explode(weight) t1 as col1;

志凯  150
志凯  170
志凯  180
上单  150
上单  180
上单  190

select key from (select explode(map('key1',1,'key2',2,'key3',3)) as (key,value))
t;

key1
key2
key3

select name,col1,col2  from testarray2 lateral view
explode(map('key1',1,'key2',2,'key3',3)) t1 as col1,col2;
志凯  key1    1
志凯  key2    2
志凯  key3    3
上单  key1    1
上单  key2    2
上单  key3    3


select name,pos,col1  from testarray2 lateral view posexplode(weight) t1 as
pos,col1;

志凯  0   150
志凯  1   170
志凯  2   180
上单  0   150
上单  1   180
上单  2   190
```

Hive 列转行

```
// testLieToLine
name col1
志凯  150
志凯  170
志凯  180
上单  150
上单  180
上单  190

create table testLieToLine(
    name string,
    col1 int
)row format delimited
fields terminated by '\t';


select name,collect_list(col1) from testLieToLine group by name;

// 结果
上单  ["150","180","190"]
志凯  ["150","170","180"]

select  t1.name
        ,collect_list(t1.col1)
from (
    select  name
            ,col1
    from testarray2
    lateral view explode(weight) t1 as col1
) t1 group by t1.name;
```

Hive自定义函数UserDefineFunction

UDF：一进一出

- 创建maven项目，并加入依赖

```
        <dependency>
            <groupId>org.apache.hive</groupId>
            <artifactId>hive-exec</artifactId>
            <version>1.2.1</version>
        </dependency>
```

- 编写代码，继承org.apache.hadoop.hive.ql.exec.UDF，实现evaluate方法，在evaluate方法中实现自己的逻辑

```java
import org.apache.hadoop.hive.ql.exec.UDF;

public class HiveUDF extends UDF {
    // hadoop => #hadoop$
    public String evaluate(String col1) {
    // 给传进来的数据 左边加上 # 号 右边加上 $
        String result = "#" + col1 + "$";
        return result;
    }
}
```

- 打成jar包并上传至Linux虚拟机
- 在hive shell中，使用 `add jar 路径` 将jar包作为资源添加到hive环境中

```
add jar /usr/local/soft/jars/HiveUDF2-1.0.jar;
```

- 使用jar包资源注册一个临时函数，fxxx1是你的函数名，'MyUDF'是主类名

```
create temporary function fxxx1 as 'MyUDF';
```

- 使用函数名处理数据

```
select fxx1(name) as fxx_name from students limit 10;

#施笑槐$
#吕金鹏$
#单乐蕊$
#葛德曜$
#宣谷芹$
#边昂雄$
#尚孤风$
#符半双$
#沈德昌$
#羿彦昌$
```

UDTF：一进多出

> "key1:value1,key2:value2,key3:value3"
>
> key1 value1
>
> key2 value2
>
> key3 value3

方法一：使用 explode+split

```
select split(t.col1,":")[0],split(t.col1,":")[1]
from (select explode(split("key1:value1,key2:value2,key3:value3",",")) as col1)
t;
```

方法二：自定UDTF

- 代码

```
import org.apache.hadoop.hive.ql.exec.UDFArgumentException;
import org.apache.hadoop.hive.ql.metadata.HiveException;
import org.apache.hadoop.hive.ql.udf.generic.GenericUDTF;
import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspector;
import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspectorFactory;
import org.apache.hadoop.hive.serde2.objectinspector.StructObjectInspector;
import
org.apache.hadoop.hive.serde2.objectinspector.primitive.PrimitiveObjectInspector
Factory;

import java.util.ArrayList;

public class HiveUDTF extends GenericUDTF {
    // 指定输出的列名 及 类型
    @Override
    public StructObjectInspector initialize(StructObjectInspector argOIs) throws
UDFArgumentException {
        ArrayList<String> filedNames = new ArrayList<String>();
        ArrayList<ObjectInspector> filedObj = new ArrayList<ObjectInspector>();
        filedNames.add("col1");
        filedObj.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);
        filedNames.add("col2");
        filedObj.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);
        return
ObjectInspectorFactory.getStandardStructObjectInspector(filedNames, filedObj);
    }

    // 处理逻辑 my_udtf(col1,col2,col3)
    // "key1:value1,key2:value2,key3:value3"
    // my_udtf("key1:value1,key2:value2,key3:value3")
    public void process(Object[] objects) throws HiveException {
        // objects 表示传入的N列
        String col = objects[0].toString();
        // key1:value1  key2:value2  key3:value3
        String[] splits = col.split(",");
        for (String str : splits) {
            String[] cols = str.split(":");
            // 将数据输出
            forward(cols);
        }

    }

    // 在UDTF结束时调用
    public void close() throws HiveException {

    }
}
```

- SQL

```
select my_udtf("key1:value1,key2:value2,key3:value3");
```

> 字段：id,col1,col2,col3,col4,col5,col6,col7,col8,col9,col10,col11,col12 共13列
>
> 数据：
>
> a,1,2,3,4,5,6,7,8,9,10,11,12
>
> b,11,12,13,14,15,16,17,18,19,20,21,22
>
> c,21,22,23,24,25,26,27,28,29,30,31,32
>
> 转成3列：id,hours,value
>
> 例如：
>
> a,1,2,3,4,5,6,7,8,9,10,11,12
>
> a,0时,1
>
> a,2时,2
>
> a,4时,3
>
> a,6时,4
>
> ……

```
create table udtfData(
    id string
    ,col1 string
    ,col2 string
    ,col3 string
    ,col4 string
    ,col5 string
    ,col6 string
    ,col7 string
    ,col8 string
    ,col9 string
    ,col10 string
    ,col11 string
    ,col12 string
)row format delimited fields terminated by ',';
```

代码：

```
import org.apache.hadoop.hive.ql.exec.UDFArgumentException;
import org.apache.hadoop.hive.ql.metadata.HiveException;
import org.apache.hadoop.hive.ql.udf.generic.GenericUDTF;
import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspector;
import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspectorFactory;
import org.apache.hadoop.hive.serde2.objectinspector.StructObjectInspector;
import
org.apache.hadoop.hive.serde2.objectinspector.primitive.PrimitiveObjectInspector
Factory;

import java.util.ArrayList;

public class HiveUDTF2 extends GenericUDTF {
    @Override
    public StructObjectInspector initialize(StructObjectInspector argOIs) throws
UDFArgumentException {
```

```
        ArrayList<String> filedNames = new ArrayList<String>();
        ArrayList<ObjectInspector> fieldObj = new ArrayList<ObjectInspector>();
        filedNames.add("col1");
        fieldObj.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);
        filedNames.add("col2");
        fieldObj.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);
        return
ObjectInspectorFactory.getStandardStructObjectInspector(filedNames, fieldObj);
    }

    public void process(Object[] objects) throws HiveException {
        int hours = 0;
        for (Object obj : objects) {
            hours = hours + 1;
            String col = obj.toString();
            ArrayList<String> cols = new ArrayList<String>();
            cols.add(hours + "时");
            cols.add(col);
            forward(cols);
        }
    }

    public void close() throws HiveException {

    }
}
```

添加jar资源:

```
add jar /usr/local/soft/HiveUDF2-1.0.jar;
```

注册udtf函数:

```
create temporary function my_udtf as 'MyUDTF';
```

SQL:

```
select id,hours,value from udtfData lateral view
my_udtf(col1,col2,col3,col4,col5,col6,col7,col8,col9,col10,col11,col12) t as
hours,value ;
```

UDAF：多进一出

Hive Shell

第一种:

```
hive -e "select * from test1.students limit 10"
```

第二种:

```
hive -f hql文件路径
```

> 将HQL写在一个文件里，再使用 -f 参数指定该文件

连续登陆问题

> 在电商、物流和银行可能经常会遇到这样的需求：统计用户连续交易的总额、连续登陆天数、连续登陆开始和结束时间、间隔天数等

数据：

> 注意：每个用户每天可能会有多条记录

```
id  datestr   amount
1,2019-02-08,6214.23
1,2019-02-08,6247.32
1,2019-02-09,85.63
1,2019-02-09,967.36
1,2019-02-10,85.69
1,2019-02-12,769.85
1,2019-02-13,943.86
1,2019-02-14,538.42
1,2019-02-15,369.76
1,2019-02-16,369.76
1,2019-02-18,795.15
1,2019-02-19,715.65
1,2019-02-21,537.71
2,2019-02-08,6214.23
2,2019-02-08,6247.32
2,2019-02-09,85.63
2,2019-02-09,967.36
2,2019-02-10,85.69
2,2019-02-12,769.85
2,2019-02-13,943.86
2,2019-02-14,943.18
2,2019-02-15,369.76
2,2019-02-18,795.15
2,2019-02-19,715.65
2,2019-02-21,537.71
3,2019-02-08,6214.23
3,2019-02-08,6247.32
3,2019-02-09,85.63
3,2019-02-09,967.36
3,2019-02-10,85.69
3,2019-02-12,769.85
3,2019-02-13,943.86
3,2019-02-14,276.81
3,2019-02-15,369.76
3,2019-02-16,369.76
3,2019-02-18,795.15
3,2019-02-19,715.65
3,2019-02-21,537.71
```

建表语句

```
create table deal_tb(
    id string
    ,datestr string
    ,amount string
)row format delimited fields terminated by ',';
```

计算逻辑

- 先按用户和日期分组求和，使每个用户每天只有一条数据

```
select  id
        ,datestr
        ,sum(amount) as sum_amount
from deal_tb
group by id,datestr
```

- 根据用户ID分组按日期排序，将日期和分组序号相减得到连续登陆的开始日期，如果开始日期相同说明连续登陆

```
select  tt1.id
        ,tt1.datestr
        ,tt1.sum_amount
        ,date_sub(tt1.datestr,rn) as grp
from(
    select  t1.id
            ,t1.datestr
            ,t1.sum_amount
            ,row_number() over(partition by id order by datestr) as rn
    from(
       select  id
                ,datestr
                ,sum(amount) as sum_amount
       from deal_tb
       group by id,datestr
    ) t1
) tt1
```

- 统计用户连续交易的总额、连续登陆天数、连续登陆开始和结束时间、间隔天数

```
select  ttt1.id
        ,ttt1.grp
        ,round(sum(ttt1.sum_amount),2) as sc_sum_amount
        ,count(1) as sc_days
        ,min(ttt1.datestr) as sc_start_date
        ,max(ttt1.datestr) as sc_end_date
        ,datediff(ttt1.grp,lag(ttt1.grp,1) over(partition by ttt1.id order by
ttt1.grp)) as iv_days
from(
    select  tt1.id
            ,tt1.datestr
            ,tt1.sum_amount
            ,date_sub(tt1.datestr,rn) as grp
    from(
        select  t1.id
                ,t1.datestr
                ,t1.sum_amount
                ,row_number() over(partition by id order by datestr) as rn
        from(
            select  id
                    ,datestr
                    ,sum(amount) as sum_amount
            from deal_tb
```

```
            group by id,datestr
        ) t1
    ) tt1
) ttt1
group by ttt1.id,ttt1.grp;
```

- 结果

```
1   2019-02-07  13600.23    3   2019-02-08  2019-02-10 NULL
1   2019-02-08  2991.650    5   2019-02-12  2019-02-16  1
1   2019-02-09  1510.8      2   2019-02-18  2019-02-19  1
1   2019-02-10  537.71      1   2019-02-21  2019-02-21  1
2   2019-02-07  13600.23    3   2019-02-08  2019-02-10 NULL
2   2019-02-08  3026.649    4   2019-02-12  2019-02-15  1
2   2019-02-10  1510.8      2   2019-02-18  2019-02-19  2
2   2019-02-11  537.71      1   2019-02-21  2019-02-21  1
3   2019-02-07  13600.23    3   2019-02-08  2019-02-10 NULL
3   2019-02-08  2730.04     5   2019-02-12  2019-02-16  1
3   2019-02-09  1510.8      2   2019-02-18  2019-02-19  1
3   2019-02-10  537.71      1   2019-02-21  2019-02-21  1
```