Note05 Hive

1. Hive 表操作

1.1. Hive 表创建语法

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table_name
   [(col_name data_type [COMMENT col_comment], ...)]
   [COMMENT table_comment]
   [PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
   [CLUSTERED BY (col_name, col_name, ...)
   [SORTED BY (col_name [ASC|DESC], ...)] INTO num_buckets BUCKETS]
   [ROW FORMAT row_format]
   [STORED AS file_format]
   [LOCATION hdfs_path]
```

说明：

1. CREATE TABLE 创建一个指定名字的表。如果相同名字的表已经存在，则抛出异常；用户可以用 IF NOT EXISTS 选项来忽略这个异常。
   . EXTERNAL关键字可以让用户创建一个外部表，在建表的同时指定一个指向实际数据的路径（LOCATION），Hive 创建内部表时，会将数据移动到数据仓库指向的路径；若创建外部表，仅记录数据所在的路径，不对数据的位置做任何改变。在删除表的时候，内部表的元数据和数据会被一起删除，而外部表只删除元数据，不删除数据。
   . LIKE 允许用户复制现有的表结构，但是不复制数据。
   . ROW FORMAT DELIMITED [FIELDS TERMINATED BY char] [COLLECTION ITEMS TERMINATED BY char] [MAP KEYS TERMINATED BY char] [LINES TERMINATED BY char] | SERDE serde_name [WITH SERDEPROPERTIES (property_name=property_value, property_name=property_value, ...)]
   用户在建表的时候可以自定义 SerDe 或者使用自带的 SerDe。如果没有指定 ROW FORMAT 或者 ROW FORMAT DELIMITED，将会使用自带的 SerDe。在建表的时候，用户还需要为表指定列，用户在指定表的列的同时也会指定自定义的 SerDe，Hive通过 SerDe 确定表的具体的列的数据。
   . STORED AS
   SEQUENCEFILE|TEXTFILE|RCFILE
   如果文件数据是纯文本，可以使用 STORED AS TEXTFILE。如果数据需要压缩，使用 STORED AS SEQUENCEFILE。
2. CLUSTERED BY
   对于每一个表（table）或者分区， Hive可以进一步组织成桶，也就是说桶是更为细粒度的数据范围划分。Hive也是 针对某一列进行桶的组织。Hive采用对列值哈希，然后除以桶的个数求余的方式决定该条记录存放在哪个桶当中。
   把表（或者分区）组织成桶（Bucket）有两个理由：
   1. 获得更高的查询处理效率。桶为表加上了额外的结构，Hive 在处理有些查询时能利用这个结构。具体而言，连接两个在（包含连接列的）相同列上划分了桶的表，可以使用 Map 端连接 （Map-side join）高效的实现。比如JOIN操作。对于JOIN操作两个表有一个相同的列，如果对这两个表都进行了桶操作。那么将保存相同列值的桶进行JOIN操作就可以，可以大大较少JOIN的数据量。
   2. 使取样（sampling）更高效。在处理大规模数据集时，在开发和修改查询的阶段，如果能在数据集的一小部分数据上试运行查询，会带来很多方便。

1.2. 管理表的操作

建表初体验

```
use myhive;
create table stu(id int,name string);
insert into stu values (1,"zhangsan");
select * from stu;
```

Hive建表时候的字段类型

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Types

分类	类型	描述	字面量示例
原始类型	BOOLEAN	true/false	TRUE
	TINYINT	1字节的有符号整数, -128~127	1Y
	SMALLINT	2个字节的有符号整数，-32768~32767	1S
	INT	4个字节的带符号整数	1
	BIGINT	8字节带符号整数	1L
	FLOAT	4字节单精度浮点数	1.0
	DOUBLE	8字节双精度浮点数	1.0
	DEICIMAL	任意精度的带符号小数	1.0
	STRING	字符串，变长	“a”,’b’
	VARCHAR	变长字符串	“a”,’b’
	CHAR	固定长度字符串	“a”,’b’
	BINARY	字节数组	无法表示
	TIMESTAMP	时间戳，毫秒值精度	122327493795
	DATE	日期	‘2016-03-29’
	INTERVAL	时间频率间隔	
复杂类型	ARRAY	有序的的同类型的集合	array(1,2)
	MAP	key-value,key必须为原始类型, value可以任意类型	map(‘a’,1,’b’,2)
	STRUCT	字段集合,类型可以不同	struct(‘1’,1,1.0), named_stract(‘col1’,’1’,’col2’,1,’clo3’,1.0)
	UNION	在有限取值范围内的一个值	create_union(1,’a’,63)

创建表并指定字段之间的分隔符

```
create  table if not exists stu2(id int ,name string) row format delimited fields terminated by '\t'
```

```
0: jdbc:hive2://node03.hadoop.com:10000> create  table if not exists stu2(id int ,name string) row format deli
mited fields terminated by '\t' stored as textfile location '/user/stu2';
INFO  : Compiling command(queryId=root_20180903114141_f2f65296-f1e7-4550-8cff-17f7f4bd5cfb): create  table if
not exists stu2(id int ,name string) row format delimited fields terminated by '\t' stored as textfile locatio
n '/user/stu2'
INFO  : Semantic Analysis Completed
INFO  : Returning Hive schema: Schema(fieldSchemas:null, properties:null)
INFO  : Completed compiling command(queryId=root_20180903114141_f2f65296-f1e7-4550-8cff-17f7f4bd5cfb); Time ta
ken: 0.011 seconds
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Executing command(queryId=root_20180903114141_f2f65296-f1e7-4550-8cff-17f7f4bd5cfb): create  table if
not exists stu2(id int ,name string) row format delimited fields terminated by '\t' stored as textfile locatio
n '/user/stu2'
INFO  : Starting task [Stage-0:DDL] in serial mode
INFO  : Completed executing command(queryId=root_20180903114141_f2f65296-f1e7-4550-8cff-17f7f4bd5cfb); Time ta
ken: 0.074 seconds
INFO  : OK
No rows affected (0.157 seconds)
```

根据查询结果创建表

```
create table stu3 as select * from stu2; # 通过复制表结构和表内容创建新表
```

根据已经存在的表结构创建表

```
create table stu4 like stu2;
```

```
No rows affected (17.624 seconds)
0: jdbc:hive2://localhost:10000> create table stu4 like stu2;
INFO  : Compiling command(queryId=root_20180611104949_e1d45413-2adf-4ce1-bdbc-2a0eca56a545): create table stu4 like stu2
INFO  : Semantic Analysis Completed
INFO  : Returning Hive schema: Schema(fieldSchemas:null, properties:null)
INFO  : Completed compiling command(queryId=root_20180611104949_e1d45413-2adf-4ce1-bdbc-2a0eca56a545); Time taken: 0.01 secon
ds
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Executing command(queryId=root_20180611104949_e1d45413-2adf-4ce1-bdbc-2a0eca56a545): create table stu4 like stu2
INFO  : Starting task [Stage-0:DDL] in serial mode
INFO  : Completed executing command(queryId=root_20180611104949_e1d45413-2adf-4ce1-bdbc-2a0eca56a545); Time taken: 0.092 seco
nds
```

查询表的类型

```
desc formatted  stu2;
```

```
4 rows selected (0.043 seconds)
0: jdbc:hive2://localhost:10000> desc formatted  stu2;
INFO  : Compiling command(queryId=root_20180611105050_cebbe6c7-134e-42a6-add6-3510250c0aa5): desc formatted  stu2
INFO  : Semantic Analysis Completed
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:col_name, type:string, comment:from deserializer), Field
Schema(name:data_type, type:string, comment:from deserializer), FieldSchema(name:comment, type:string, comment:from deseriali
zer)], properties:null)
INFO  : Completed compiling command(queryId=root_20180611105050_cebbe6c7-134e-42a6-add6-3510250c0aa5); Time taken: 0.064 seco
nds
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Executing command(queryId=root_20180611105050_cebbe6c7-134e-42a6-add6-3510250c0aa5): desc formatted  stu2
INFO  : Starting task [Stage-0:DDL] in serial mode
INFO  : Completed executing command(queryId=root_20180611105050_cebbe6c7-134e-42a6-add6-3510250c0aa5); Time taken: 0.022 seco
nds
INFO  : OK
+-------------------------------+----------------------------------------------------+-----------------------+--+
|           col_name            |                     data_type                      |        comment        |
+-------------------------------+----------------------------------------------------+-----------------------+--+
| # col_name                    | data_type                                          | comment               |
|                               | NULL                                               | NULL                  |
| id                            | int                                                |                       |
| name                          | string                                             |                       |
|                               | NULL                                               | NULL                  |
| # Detailed Table Information  | NULL                                               | NULL                  |
| Database:                     | myhive                                             | NULL                  |
| Owner:                        | root                                               | NULL                  |
| CreateTime:                   | Mon Jun 11 10:47:26 CST 2018                       | NULL                  |
| LastAccessTime:               | UNKNOWN                                            | NULL                  |
| Protect Mode:                 | None                                               | NULL                  |
| Retention:                    | 0                                                  | NULL                  |
| Location:                     | hdfs://192.168.52.100:9000/user/hive/warehouse/myhive/stu2 | NULL          |
| Table Type:                   | MANAGED_TABLE                                      | NULL                  |
| Table Parameters:             | NULL                                               | NULL                  |
|                               | transient_lastDdlTime                              | 1528685246            |
```

1.3. 外部表的操作

外部表说明

外部表因为是指定其他的hdfs路径的数据加载到表当中来，所以hive表会认为自己不完全独占这份数据，所以删除hive表的时候，数据仍然存放在hdfs当中，不会删掉

管理表和外部表的使用场景

每天将收集到的网站日志定期流入HDFS文本文件。在外部表（原始日志表）的基础上做大量的统计分析，用到的中间表、结果表使用内部表存储，数据通过SELECT+INSERT进入内部表。

操作案例

分别创建老师与学生表外部表，并向表中加载数据

创建老师表

```
create external table teacher (t_id string,t_name string) row format delimited fields terminated by
```

创建学生表

```
create external table student (s_id string,s_name string,s_birth string , s_sex string ) row format (
```

加载数据

```
load data local inpath '/export/servers/hivedatas/student.csv' into table student;
```

加载数据并覆盖已有数据

```
load data local inpath '/export/servers/hivedatas/student.csv' overwrite  into table student;
```

从hdfs文件系统向表中加载数据（需要提前将数据上传到hdfs文件系统）

```
cd /export/servers/hivedatas
hdfs dfs -mkdir -p /hivedatas
hdfs dfs -put techer.csv /hivedatas/
load data inpath '/hivedatas/techer.csv' into table teacher;
```

1.4. 分区表

在大数据中，最常用的一种思想就是分治，我们可以把大的文件切割划分成一个个的小的文件，这样每次操作一个小的文件就会很容易了，同样的道理，在hive当中也是支持这种思想的，就是我们可以把大的数据，按照每天，或者每小时进行切分成一个个的小的文件，这样去操作小的文件就会容易得多了

创建分区表语法

```
create table score(s_id string,c_id string, s_score int) partitioned by (month string) row format de
```

创建一个表带多个分区

```
create table score2 (s_id string,c_id string, s_score int) partitioned by (year string,month string,
```

加载数据到分区表中

```
load data local inpath '/export/servers/hivedatas/score.csv' into table score partition (month='2018
```

加载数据到多分区表中

```
load data local inpath '/export/servers/hivedatas/score.csv' into table score2 partition(year='2018'
```

多分区表联合查询(使用 union all)

```
select * from score where month = '201806' union all select * from score where month = '201806';
```

查看分区

```
show  partitions  score;
```

添加一个分区

```
alter table score add partition(month='201805');
```

删除分区

```
alter table score drop partition(month = '201806');
```

1.5. 分桶表

将数据按照指定的字段进行分成多个桶中去，说白了就是将数据按照字段进行划分，可以将数据按照字段划分到多个文件当中去

开启 Hive 的分桶功能

```
set hive.enforce.bucketing=true;
```

设置 Reduce 个数

```
set mapreduce.job.reduces=3;
```

创建桶表

```
create table course (c_id string,c_name string,t_id string) clustered by(c_id) into 3 buckets row fo
```

桶表的数据加载，由于通标的数据加载通过hdfs dfs -put文件或者通过load data均不好使，只能通过insert overwrite

创建普通表，并通过insert overwrite的方式将普通表的数据通过查询的方式加载到桶表当中去

创建普通表

```
create table course_common (c_id string,c_name string,t_id string) row format delimited fields termi
```

普通表中加载数据

```
load data local inpath '/export/servers/hivedatas/course.csv' into table course_common;
```

通过insert overwrite给桶表中加载数据

```
insert overwrite table course select * from course_common cluster by(c_id);
```

1.6. 修改表

重命名

基本语法：

```
alter  table  old_table_name  rename  to  new_table_name;
```

把表score4修改成score5

```
alter table score4 rename to score5;
```

增加/修改列信息

- 查询表结构

```
desc score5;
```

- 添加列

```
alter table score5 add columns (mycol string, mysco string);
```

- 查询表结构

```
desc score5;
```

- 更新列

```
alter table score5 change column mysco mysconew int;
```

1.7. 删除表

```
drop table score5;
```

1.8. hive表中加载数据

直接向分区表中插入数据

```
create table score3 like score;

insert into table score3 partition(month ='201807') values ('001','002','100');
```

通过查询插入数据

通过load方式加载数据

```
load data local inpath '/export/servers/hivedatas/score.csv' overwrite into table score partition(mo
```

通过查询方式加载数据

```
create table score4 like score;
insert overwrite table score4 partition(month = '201806') select s_id,c_id,s_score from score;
```

2. Hive 查询语法

2.1. SELECT

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list [HAVING condition]]
[CLUSTER BY col_list
| [DISTRIBUTE BY col_list] [SORT BY| ORDER BY col_list]
]
[LIMIT number]
```

1. order by 会对输入做全局排序，因此只有一个reducer，会导致当输入规模较大时，需要较长的计算时间。
2. sort by不是全局排序，其在数据进入reducer前完成排序。因此，如果用sort by进行排序，并且设置mapred.reduce.tasks>1，则sort by只保证每个reducer的输出有序，不保证全局有序。
3. distribute by(字段)根据指定的字段将数据分到不同的reducer，且分发算法是hash散列。
4. Cluster by(字段) 除了具有Distribute by的功能外，还会对该字段进行排序。 ---> distribute by + sort by

因此，如果分桶和sort字段是同一个时，此时， `cluster by = distribute by + sort by`

分桶表的作用：最大的作用是用来提高join操作的效率；

思考这个问题： select a.id,a.name,b.addr from a join b on a.id = b.id;
如果a表和b表已经是分桶表，而且分桶的字段是id字段 做这个join操作时，还需要全表做笛卡尔积吗?

2.2. 查询语法

全表查询

```
select * from score;
```

选择特定列

```
select s_id ,c_id from score;
```

列别名

1）重命名一个列。
2）便于计算。
3）紧跟列名，也可以在列名和别名之间加入关键字‘AS’

```
select s_id as myid ,c_id from score;
```

2.3. 常用函数

- 求总行数（count）

```
select count(1) from score;
```

- 求分数的最大值（max）

```
select max(s_score) from score;
```

- 求分数的最小值（min）

```
select min(s_score) from score;
```

- 求分数的总和（sum）

```
select sum(s_score) from score;
```

- 求分数的平均值（avg）

```
select avg(s_score) from score;
```

2.4. LIMIT语句

典型的查询会返回多行数据。LIMIT子句用于限制返回的行数。

```
select * from score limit 3;
```

2.5. WHERE语句

1. 使用WHERE 子句，将不满足条件的行过滤掉。
2. WHERE 子句紧随 FROM 子句。
3. 案例实操

查询出分数大于60的数据

```
select * from score where s_score > 60;
```

比较运算符

操作符	支持的数据类型	描述

操作符	支持的数据类型	描述
A=B	基本数据类型	如果A等于B则返回TRUE，反之返回FALSE
A<=>B	基本数据类型	如果A和B都为NULL，则返回TRUE，其他的和等号（=）操作符的结果一致，如果任一为NULL则结果为NULL
A<>B, A!=B	基本数据类型	A或者B为NULL则返回NULL；如果A不等于B，则返回TRUE，反之返回FALSE
A<B	基本数据类型	A或者B为NULL，则返回NULL；如果A小于B，则返回TRUE，反之返回FALSE
A<=B	基本数据类型	A或者B为NULL，则返回NULL；如果A小于等于B，则返回TRUE，反之返回FALSE
A>B	基本数据类型	A或者B为NULL，则返回NULL；如果A大于B，则返回TRUE，反之返回FALSE
A>=B	基本数据类型	A或者B为NULL，则返回NULL；如果A大于等于B，则返回TRUE，反之返回FALSE
A [NOT] BETWEEN B AND C	基本数据类型	如果A，B或者C任一为NULL，则结果为NULL。 如果A的值大于等于B而且小于或等于C，则结果为TRUE，反之为FALSE。 如果使用NOT关键字则可达到相反的效果。
A IS NULL	所有数据类型	如果A等于NULL，则返回TRUE，反之返回FALSE
A IS NOT NULL	所有数据类型	如果A不等于NULL，则返回TRUE，反之返回FALSE
IN(数值1, 数值2)	所有数据类型	使用 IN运算显示列表中的值
A [NOT] LIKE B	STRING 类型	B是一个SQL下的简单正则表达式，如果A与其匹配的话，则返回TRUE；反之返回FALSE。 B的表达式说明如下：‘x%’表示A必须以字母‘x’开头，‘%x’表示A必须以字母’x’结尾，而‘%x%’表示A包含有字母’x’,可以位于开头，结尾或者字符串中间。 如果使用NOT关键字则可达到相反的效果。
A RLIKE B, A REGEXP B	STRING	类型 B是一个正则表达式，如果A与其匹配，则返回TRUE；反之返回FALSE。匹配使用的是JDK中的正则表达式接口实现的，因为正则也依据其中的规则。例如，正则表达式必须和整个字符串A相匹配，而不是只需与其字符串匹配。

- 查询分数等于80的所有的数据

```
select * from score where s_score = 80;
```

- 查询分数在80到100的所有数据

```
select * from score where s_score between 80 and 100;
```

- 查询成绩为空的所有数据

```
select * from score where s_score is null;
```

- 查询成绩是80和90的数据

```
select * from score where s_score in(80,90);
```

2.6. LIKE 和 RLIKE

1. 使用LIKE运算选择类似的值
2. 选择条件可以包含字符或数字:

```
% 代表零个或多个字符(任意个字符)。
_ 代表一个字符。
```

3. RLIKE子句是Hive中这个功能的一个扩展，其可以通过Java的正则表达式这个更强大的语言来指定匹配条件。
4. 案例实操
    1. 查找以8开头的所有成绩

        ```
        select * from score where s_score like '8%';
        ```

    2. 查找第二个数值为9的所有成绩数据

        ```
        select * from score where s_score like '_9%';
        ```

    3. 查找成绩中含9的所有成绩数据

        ```
        select * from score where s_score rlike '[9]';  #  like '%9%'
        ```

2.7. 逻辑运算符

操作符	含义
AND	逻辑并
OR	逻辑或
NOT	逻辑否

- 查询成绩大于80，并且s_id是01的数据

```
select * from score where s_score >80 and s_id = '01';
```

- 查询成绩大于80，或者s_id 是01的数

```
select * from score where s_score > 80 or s_id = '01';
```

- 查询s_id 不是 01和02的学生

```
select * from score where s_id not in ('01','02');
```

2.8. 分组

GROUP BY 语句

GROUP BY语句通常会和聚合函数一起使用，按照一个或者多个列队结果进行分组，然后对每个组执行聚合操作。
案例实操：

- 计算每个学生的平均分数

```
select s_id ,avg(s_score) from score group by s_id;
```

- 计算每个学生最高成绩

```
select s_id ,max(s_score) from score group by s_id;
```

HAVING 语句

1. having与where不同点
    1. where针对表中的列发挥作用，查询数据；having针对查询结果中的列发挥作用，筛选数据。
    2. where后面不能写分组函数，而having后面可以使用分组函数。
    3. having只用于group by分组统计语句。
2. 案例实操：
    - 求每个学生的平均分数

    ```
    select s_id ,avg(s_score) from score group by s_id;
    ```

    - 求每个学生平均分数大于85的人

    ```
    select s_id ,avg(s_score) avgscore from score group by s_id having avgscore > 85;
    ```

2.9. JOIN 语句

2.9.1. 等值 JOIN

Hive支持通常的SQL JOIN语句，但是只支持等值连接，不支持非等值连接。

案例操作: 查询分数对应的姓名

```
SELECT s.s_id,s.s_score,stu.s_name,stu.s_birth  FROM score s LEFT JOIN student stu ON s.s_id = stu.s
```

2.9.2. 表的别名

- 好处
    - 使用别名可以简化查询。
    - 使用表名前缀可以提高执行效率。
- 案例实操
    - 合并老师与课程表

    ```
    select * from techer t join course c on t.t_id = c.t_id;
    ```

2.9.3. 内连接

内连接：只有进行连接的两个表中都存在与连接条件相匹配的数据才会被保留下来。

```
select * from techer t inner join course c on t.t_id = c.t_id;
```

2.9.4. 左外连接

左外连接：JOIN操作符左边表中符合WHERE子句的所有记录将会被返回。
查询老师对应的课程

```
select * from techer t left join course c on t.t_id = c.t_id;
```

2.9.5. 右外连接

右外连接：JOIN操作符右边表中符合WHERE子句的所有记录将会被返回。

```
select * from teacher t right join course c on t.t_id = c.t_id;
```

2.9.6. 多表连接

注意：连接 n个表，至少需要n-1个连接条件。例如：连接三个表，至少需要两个连接条件。

多表连接查询，查询老师对应的课程，以及对应的分数，对应的学生

```
select * from teacher t
left join course c
on t.t_id = c.t_id
left join score s
on s.c_id = c.c_id
left join student stu
on s.s_id = stu.s_id;
```

大多数情况下，Hive会对每对JOIN连接对象启动一个MapReduce任务。本例中会首先启动一个MapReduce job对表techer和表course进行连接操作，然后会再启动一个MapReduce job将第一个MapReduce job的输出和表score;进行连接操作。

2.9. 排序

2.9.1. 全局排序

Order By：全局排序，一个reduce

1. 使用 ORDER BY 子句排序
   ASC（ascend）: 升序（默认）
   DESC（descend）: 降序
2. ORDER BY 子句在SELECT语句的结尾。
3. 案例实操
   1. 查询学生的成绩，并按照分数降序排列

```
SELECT * FROM student s LEFT JOIN score sco ON s.s_id = sco.s_id ORDER BY sco.s_score DESC;
```

   2. 查询学生的成绩，并按照分数升序排列

```
SELECT * FROM student s LEFT JOIN score sco ON s.s_id = sco.s_id ORDER BY sco.s_score asc;
```

2.9.2. 按照别名排序

按照分数的平均值排序

```
select s_id ,avg(s_score) avg from score group by s_id order by avg;
```

2.9.3. 多个列排序

按照学生id和平均成绩进行排序

```
select s_id ,avg(s_score) avg from score group by s_id order by s_id,avg;
```

2.9.4. 每个MapReduce内部排序（Sort By）局部排序

Sort By：每个MapReduce内部进行排序，对全局结果集来说不是排序。

1. 设置reduce个数

```
set mapreduce.job.reduces=3;
```

2. 查看设置reduce个数

```
set mapreduce.job.reduces;
```

3. 查询成绩按照成绩降序排列

```
select * from score sort by s_score;
```

4. 将查询结果导入到文件中（按照成绩降序排列）

```
insert overwrite local directory '/export/servers/hivedatas/sort' select * from score sort by s_score
```

2.9.5. 分区排序（DISTRIBUTE BY）

Distribute By：类似MR中partition，进行分区，结合sort by使用。

注意，Hive要求DISTRIBUTE BY语句要写在SORT BY语句之前。

对于distribute by进行测试，一定要分配多reduce进行处理，否则无法看到distribute by的效果。

案例实操：先按照学生id进行分区，再按照学生成绩进行排序。

1. 设置reduce的个数，将我们对应的s_id划分到对应的reduce当中去

```
set mapreduce.job.reduces=7;
```

2. 通过distribute by 进行数据的分区

```
insert overwrite local directory '/export/servers/hivedatas/sort' select * from score distribute by s
```

2.9.6. CLUSTER BY

当distribute by和sort by字段相同时，可以使用cluster by方式。

cluster by除了具有distribute by的功能外还兼具sort by的功能。但是排序只能是倒序排序，不能指定排序规则为ASC或者DESC。

以下两种写法等价

```
select * from score cluster by s_id;
select * from score distribute by s_id sort by s_id;
```

3. Hive 函数

3.1. 内置函数

内容较多，见《Hive官方文档》

```
https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF
```

1. 查看系统自带的函数

```
hive> show functions;
```

2. 显示自带的函数的用法

```
hive> desc function upper;
```

3. 详细显示自带的函数的用法

```
hive> desc function extended upper;
```

4:常用内置函数

```
# 字符串连接函数： concat
<p class="mume-header " id="字符串连接函数-concat"></p>

select concat('abc','def','gh');
# 带分隔符字符串连接函数： concat_ws
<p class="mume-header " id="带分隔符字符串连接函数-concat_ws"></p>

select concat_ws(',','abc','def','gh');
# cast类型转换
<p class="mume-header " id="cast类型转换"></p>

select cast(1.5 as int);
# get_json_object(json 解析函数，用来处理json，必须是json格式)
<p class="mume-header " id="get_json_objectjson-解析函数用来处理json必须是json格式"></p>

 select get_json_object('{"name":"jack","age":"20"}','$.name');
# URL解析函数
<p class="mume-header " id="url解析函数"></p>

 select parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'HOST');
# explode：把map集合中每个键值对或数组中的每个元素都单独生成一行的形式
<p class="mume-header " id="explode把map集合中每个键值对或数组中的每个元素都单独生成一行的形式"></p>
```

3.2. 自定义函数

1. Hive 自带了一些函数，比如：max/min等，但是数量有限，自己可以通过自定义UDF来方便的扩展。
2. 当Hive提供的内置函数无法满足你的业务处理需要时，此时就可以考虑使用用户自定义函数（UDF：user-defined function）。
3. 根据用户自定义函数类别分为以下三种：upper -->my_upper
    1. UDF（User-Defined-Function）
        - 一进一出
    2. UDAF（User-Defined Aggregation Function）
        - 聚集函数，多进一出
        - 类似于： `count` / `max` / `min`
    3. UDTF（User-Defined Table-Generating Functions）
        - 一进多出
        - 如 `lateral view explore()`
4. 官方文档地址 `https://cwiki.apache.org/confluence/display/Hive/HivePlugins`
5. 编程步骤：
    1. 继承org.apache.hadoop.hive.ql.UDF
    2. 需要实现evaluate函数；evaluate函数支持重载；
6. 注意事项
    1. UDF必须要有返回类型，可以返回null，但是返回类型不能为void；
    2. UDF中常用Text/LongWritable等类型，不推荐使用java类型；

3.3. UDF 开发实例

3.3.1. Step 1 创建 Maven 工程

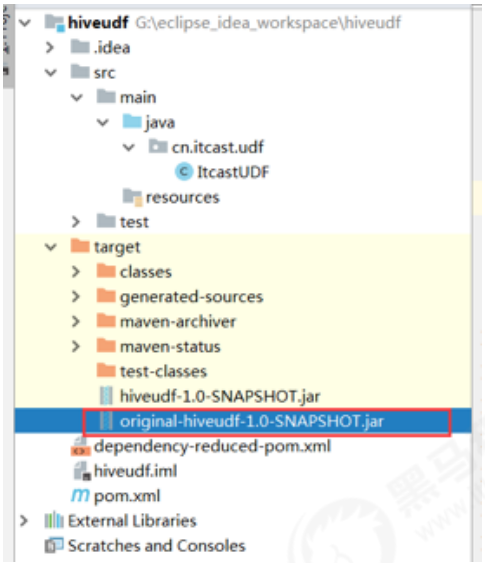
```
<dependencies>
    <!-- https://mvnrepository.com/artifact/org.apache.hive/hive-exec -->
    <dependency>
        <groupId>org.apache.hive</groupId>
        <artifactId>hive-exec</artifactId>
        <version>3.1.1</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-common -->
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-common</artifactId>
        <version>3.1.1</version>
    </dependency>
</dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.0</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                    <encoding>UTF-8</encoding>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

3.3.2. Step 2 开发 Java 类集成 UDF

```
public class MyUDF  extends UDF{
    public Text evaluate(final Text str){
        String tmp_str = str.toString();
        if(str != null && !tmp_str.equals("")){
          String str_ret =   tmp_str.substring(0, 1).toUpperCase() + tmp_str.substring(1);
          return  new Text(str_ret);
        }
        return  new Text("");
    }
}
```

3.3.3. Step 3 项目打包，并上传到hive的lib目录下



3.3.4. Step 4 添加jar包

重命名我们的jar包名称

```
cd /export/servers/apache-hive-3.1.1-bin/lib
mv original-day_05_hive_udf-1.0-SNAPSHOT.jar myudf.jar
```

hive的客户端添加我们的jar包

```
add jar /export/servers/apache-hive-3.1.1-bin/lib/udf.jar;
```

3.3.5. Step 5 设置函数与我们的自定义函数关联

```
create temporary function my_upper as 'cn.itcast.udf.ItcastUDF';
```

```
1 row selected (0.439 seconds)
0: jdbc:hive2://localhost:10000> create temporary function tolowercase as 'cn.itcast.udf.ItcastUDF';
```

3.3.6. Step 6 使用自定义函数

```
select my_upper('abc');
```