1-Hive数据库和表操作

1-1 数据【库】操作

1-1-1 创建数据库

```
create database if not exists 数据库名称;
```

```
0: jdbc:hive2://node3:10000> create database if not exists myhive2;
No rows affected (0.049 seconds)
0: jdbc:hive2://node3:10000>
```

- 设置数据库默认存储位置

hive的表存放位置模式是由hive-site.xml当中的一个属性指定的

```
<property>
    <name>hive.metastore.warehouse.dir</name>
    <value>/user/hive/warehouse</value>
</property>
```

1-1-2 创建数据库并指定hdfs存储位置location

```
create database 数据库名称 location 'hdfs路径';
```

```
0: jdbc:hive2://node3:10000> create database if not exists myhive3 location
'/hive/databases';
No rows affected (0.034 seconds)
0: jdbc:hive2://node3:10000>
```

1-1-3 查看数据库详细信息

```
desc database 数据库名称;
```

主要是可以查看到 这个数据库的 hdfs 路径： hdfs://node1:8020/user/hive/warehouse/myhive.db

```
0: jdbc:hive2://node3:10000> desc database myhive;
+----------+----------+-----------------------------------------------------+-------
------+------------+------------+--+
| db_name  | comment  |                      location                       |
owner_name  | owner_type  | parameters  |
+----------+----------+-----------------------------------------------------+-------
------+------------+------------+--+
| myhive   |          | hdfs://node1:8020/user/hive/warehouse/myhive.db  | root
       | USER        |            |
+----------+----------+-----------------------------------------------------+-------
------+------------+------------+--+
1 row selected (0.03 seconds)
0: jdbc:hive2://node3:10000>
```

1-1-4 删除数据库 drop

- 删除一个空数据库，如果数据库下面有数据表，那么就会报错

```
0: jdbc:hive2://node3:10000> create database mytest2;
No rows affected (0.057 seconds)
0: jdbc:hive2://node3:10000> drop database mytest2;
No rows affected (0.049 seconds)
0: jdbc:hive2://node3:10000>
```

- 删除一个有数据的数据库报错

```
0: jdbc:hive2://node3:10000> drop database mytest3;
Error: Error while processing statement: FAILED: Execution Error, return code 1
from org.apache.hadoop.hive.ql.exec.DDLTask.
InvalidOperationException(message:Database mytest3 is not empty. One or more
tables exist.) (state=08S01,code=1)
0: jdbc:hive2://node3:10000>
```

说明mytest3 里面有数据无法正常删除；

- 强制删除数据库，包含数据库下面的表一起删除

```
0: jdbc:hive2://node3:10000> drop database mytest3 cascade;
No rows affected (1.268 seconds)
0: jdbc:hive2://node3:10000>
```

1-2 数据库【表】操作

1-2-1 创建数据库表语法

```
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。

2、**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**）；

- 有external： 外部表；
- 没有 external： 内部表；
- 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。

3、**LIKE**允许用户复制现有的表结构，但是不复制数据。

4、ROW FORMAT DELIMITED 可用来指定行分隔符

- row format delimited fields terminated by '\t'; # 分隔符是 \t

5、STORED AS SEQUENCEFILE|TEXTFILE|RCFILE 来指定该表数据的存储格式，hive中，表的默认存储格式为TextFile。

6、partitioned by（分区）

- 分区就是分文件夹

7、CLUSTERED BY (分桶)

- 对于每一个表（table）进行分桶(MapReuce中的分区），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。

8、**LOCATION**: 指定表在HDFS上的存储位置。

1-2-2 Hive建表时候的字段类型

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 (-2147483648~2147483647)	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’
	Time	时分秒	‘12:35:46’
	DateTime	年月日 时分秒	
复杂类型	ARRAY	有序的的同类型的集合	["beijing","shanghai","tianjin","hangzhou"]
MAP	key-value,key必须为原始类型，value可以任意类型	{"数学":80,"语文":89,"英语":95}	
STRUCT	字段集合,类型可以不同	struct(‘1’,1,1.0)	

1-2-3 内部表操作

1、hive建表初体验 没有指定分隔符默认为‘\0001’;

- 案例:

```
create database myhive2;
use myhive2;
create table stu(id int,name string);
insert into stu values (1,"Tom");
select * from stu;
```

```
0: jdbc:hive2://node3:10000> drop database myhive2;
No rows affected (0.069 seconds)
0: jdbc:hive2://node3:10000>
0: jdbc:hive2://node3:10000>
0: jdbc:hive2://node3:10000>
0: jdbc:hive2://node3:10000> create database myhive2;
No rows affected (0.068 seconds)
0: jdbc:hive2://node3:10000> use myhive2;
No rows affected (0.037 seconds)
```

```
0: jdbc:hive2://node3:10000> create table stu(id int,name string);
No rows affected (0.148 seconds)
0: jdbc:hive2://node3:10000> insert into stu values(1,"Tom");
No rows affected (13.378 seconds)
0: jdbc:hive2://node3:10000> select * from stu;
+---------+-----------+--+
| stu.id  | stu.name  |
+---------+-----------+--+
| 1       | Tom       |
+---------+-----------+--+
1 row selected (0.141 seconds)
0: jdbc:hive2://node3:10000>
```

- 查看hdfs存储信息

```
[root@node3 expect]# hadoop fs -cat /user/hive/warehouse/myhive2.db/stu/000000_0
1Tom
[root@node3 expect]#
```

2、创建表并指定字段之间的分隔符

- 案例

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

```
0: jdbc:hive2://node3:10000> create table if not exists stu2(id int , name
string) row format delimited fields terminated by '\t';
No rows affected (0.117 seconds)
0: jdbc:hive2://node3:10000> insert into stu2 values (2,"Jerry");
No rows affected (9.98 seconds)
0: jdbc:hive2://node3:10000> select * from stu2;
+----------+------------+--+
| stu2.id  | stu2.name  |
+----------+------------+--+
| 2        | Jerry      |
+----------+------------+--+
1 row selected (0.141 seconds)
0: jdbc:hive2://node3:10000>
```

- 查看hdfs存储信息；可以对比没有指定分隔符；

```
[root@node3 expect]# hadoop fs -cat /user/hive/warehouse/myhive2.db/stu/000000_0
1Tom
[root@node3 expect]# hadoop fs -cat
/user/hive/warehouse/myhive2.db/stu2/000000_0
2   Jerry
[root@node3 expect]#
```

3、根据查询结果创建表 as

- 特点
  - 复制表的结构;
  - 复制表中的数据;
- 案例;

```
create table stu2_2 as select * from stu2;
```

```
0: jdbc:hive2://node3:10000> create table stu2_2 as select * from stu2;
No rows affected (13.421 seconds)
0: jdbc:hive2://node3:10000>
0: jdbc:hive2://node3:10000> select * from stu2_2;
+------------+--------------+--+
| stu2_2.id  | stu2_2.name  |
+------------+--------------+--+
| 2          | Jerry        |
+------------+--------------+--+
1 row selected (0.131 seconds)
0: jdbc:hive2://node3:10000>
```

4、根据已经存在的表结构创建表 like

- 特点
  - 复制表的结构;
- 案例:

```
create table stu2_3 like stu2;
```

```
0: jdbc:hive2://node3:10000> create table stu2_3 like stu2;
No rows affected (0.115 seconds)
0: jdbc:hive2://node3:10000> select * from stu2_3;
+------------+--------------+--+
| stu2_3.id  | stu2_3.name  |
+------------+--------------+--+
+------------+--------------+--+
No rows selected (0.089 seconds)
0: jdbc:hive2://node3:10000>
```

5、查询表的类型

- 案例:

```
desc stu2;
desc formatted  stu2;     # 可以看到更详细的信息;
```

```
0: jdbc:hive2://node3:10000> desc stu2;
+-----------+------------+----------+--+
| col_name  | data_type  | comment  |
+-----------+------------+----------+--+
| id        | int        |          |
| name      | string     |          |
+-----------+------------+----------+--+
2 rows selected (0.076 seconds)
0: jdbc:hive2://node3:10000>
```

```
0: jdbc:hive2://node3:10000> desc formatted stu2;
+-------------------------------+----------------------------------------------------
------------+-----------------------------+--+
|           col_name            |                           data_type
             |           comment           |
+-------------------------------+----------------------------------------------------
------------+-----------------------------+--+
| # col_name                    | data_type
            | comment                     |
|                               | NULL
             | NULL                        |
| id                            | int
              |                             |
| name                          | string
             |                             |
|                               | NULL
             | NULL                        |
| # Detailed Table Information  | NULL
             | NULL                        |
| Database:                     | myhive2
            | NULL                        |
| Owner:                        | root
             | NULL                        |
| CreateTime:                   | Wed Jan 20 16:04:07 CST 2021
             | NULL                        |
| LastAccessTime:               | UNKNOWN
              | NULL                        |
| Retention:                    | 0
              | NULL                        |
| Location:                     |
hdfs://node1:8020/user/hive/warehouse/myhive2.db/stu2       | NULL
         |
| Table Type:                   | MANAGED_TABLE
              | NULL                        |
| Table Parameters:             | NULL
             | NULL                        |
|                               | COLUMN_STATS_ACCURATE
            | {\"BASIC_STATS\":\"true\"}  |
```

```
|                               | numFiles
             | 1                           |
|                               | numRows
             | 1                           |
|                               | rawDataSize
             | 7                           |
|                               | totalSize
             | 8                           |
|                               | transient_lastDdlTime
             | 1611129890                  |
|                               | NULL
             | NULL                        |
| # Storage Information         | NULL
             | NULL                        |
| SerDe Library:                |
org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe          | NULL
         |
| InputFormat:                  | org.apache.hadoop.mapred.TextInputFormat
             | NULL                        |
| OutputFormat:                 |
org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat  | NULL
         |
| Compressed:                   | No
             | NULL                        |
| Num Buckets:                  | -1
             | NULL                        |
| Bucket Columns:               | []
             | NULL                        |
| Sort Columns:                 | []
             | NULL                        |
| Storage Desc Params:          | NULL
             | NULL                        |
|                               | field.delim
            | \t                          |
|                               | serialization.format
             | \t                          |
+-------------------------------+-----------------------------------------------
-------------+-----------------------------+--+
32 rows selected (0.074 seconds)
0: jdbc:hive2://node3:10000>
```

6、删除表

- 案例：

```
drop table stu2;
```

注意： 发现删除内部表之后，所有的内容全部删除；

1-2-4 外部表操作

1、数据装载载命令Load

- Load命令用于将外部数据加载到Hive表中
- 语法

```
load data [local] inpath '/export/data/student.txt' [overwrite] | into table
student [partition (partcol1=val1,…)];
```

- 参数

  1、load data:表示加载数据

  2、local:表示从本地加载数据到hive表；否则从HDFS加载数据到hive表

  3、inpath:表示加载数据的路径

  4、overwrite:表示 **覆盖** 表中已有数据，否则表示 **追加**

  5、into table:表示加载到哪张表

  6、student:表示具体的表名

  7、partition:表示上传到指定分区

2、操作案例

- 数据准备
  - student.txt

```
01  赵雷  1990-01-01  男
02  钱电  1990-12-21  男
03  孙风  1990-05-20  男
04  李云  1990-08-06  男
05  周梅  1991-12-01  女
06  吴兰  1992-03-01  女
07  郑竹  1989-07-01  女
08  王菊  1990-01-20  女
```

  - teacher.txt

```
01 张三
02 李四
03 王五
```

- 1- 创建老师表 teacher

```
create external table teacher (t_id string ,t_name string) row format
delimited fields terminated by '\t';
```

- 2- 创建学生表 student

```
create external table student (s_id string ,s_name string , s_birth
string,s_sex string) row format delimited fields terminated by '\t';
```

- 3- 从本地文件系统向表中加载数据 (追加) (local)

```
load data local inpath '/export/data/student.txt' into table student;
```

```
load data local inpath '/export/data/teacher.txt' into table teacher;
```

- 4- 查看数据

```
0: jdbc:hive2://node3:10000> select * from student;
+---------------+-----------------+------------------+----------------+--+
| student.s_id  | student.s_name  | student.s_birth  | student.s_sex  |
+---------------+-----------------+------------------+----------------+--+
| 01            | 赵雷              | 1990-01-01       | 男              |
| 02            | 钱电              | 1990-12-21       | 男              |
| 03            | 孙风              | 1990-05-20       | 男              |
| 04            | 李云              | 1990-08-06       | 男              |
| 05            | 周梅              | 1991-12-01       | 女              |
| 06            | 吴兰              | 1992-03-01       | 女              |
| 07            | 郑竹              | 1989-07-01       | 女              |
| 08            | 王菊              | 1990-01-20       | 女              |
+---------------+-----------------+------------------+----------------+--+
8 rows selected (0.13 seconds)
0: jdbc:hive2://node3:10000>
```

- 5- 再次从本地文件系统向表中加载数据 并查看数据

```
load data local inpath '/export/data/student.txt' into table student;
```

  - 追加结果

```
0: jdbc:hive2://node3:10000> select * from student;
+---------------+-----------------+------------------+----------------+--+
| student.s_id  | student.s_name  | student.s_birth  | student.s_sex  |
+---------------+-----------------+------------------+----------------+--+
| 01            | 赵雷              | 1990-01-01       | 男              |
| 02            | 钱电              | 1990-12-21       | 男              |
| 03            | 孙风              | 1990-05-20       | 男              |
| 04            | 李云              | 1990-08-06       | 男              |
| 05            | 周梅              | 1991-12-01       | 女              |
| 06            | 吴兰              | 1992-03-01       | 女              |
| 07            | 郑竹              | 1989-07-01       | 女              |
| 08            | 王菊              | 1990-01-20       | 女              |
| 01            | 赵雷              | 1990-01-01       | 男              |
| 02            | 钱电              | 1990-12-21       | 男              |
| 03            | 孙风              | 1990-05-20       | 男              |
```

```
| 04            | 李云              | 1990-08-06       | 男              |
| 05            | 周梅              | 1991-12-01       | 女              |
| 06            | 吴兰              | 1992-03-01       | 女              |
| 07            | 郑竹              | 1989-07-01       | 女              |
| 08            | 王菊              | 1990-01-20       | 女              |
+---------------+-----------------+------------------+----------------+--+
16 rows selected (0.161 seconds)
0: jdbc:hive2://node3:10000>
```

- 6- 加载数据并覆盖已有数据 overwrite

```
load data local inpath '/export/data/student.txt' overwrite into table
student;
```

  - 查看数据

```
0: jdbc:hive2://node3:10000> load data local inpath
'/export/data/student.txt' overwrite into table student;
No rows affected (0.222 seconds)
0: jdbc:hive2://node3:10000> select * from student;
+---------------+-----------------+------------------+----------------+--+
| student.s_id  | student.s_name  | student.s_birth  | student.s_sex  |
+---------------+-----------------+------------------+----------------+--+
| 01            | 赵雷              | 1990-01-01       | 男              |
| 02            | 钱电              | 1990-12-21       | 男              |
| 03            | 孙风              | 1990-05-20       | 男              |
| 04            | 李云              | 1990-08-06       | 男              |
| 05            | 周梅              | 1991-12-01       | 女              |
| 06            | 吴兰              | 1992-03-01       | 女              |
| 07            | 郑竹              | 1989-07-01       | 女              |
| 08            | 王菊              | 1990-01-20       | 女              |
+---------------+-----------------+------------------+----------------+--+
8 rows selected (0.11 seconds)
0: jdbc:hive2://node3:10000>
```

- 7- 从hdfs文件系统向表中加载数据 (去掉local)
  - 其实就是一个移动文件的操作;
  - 需要提前将数据上传到hdfs文件系统;

```
hadoop fs -mkdir -p /hivedatas
cd /export/data/hivedatas
hadoop fs -put teacher.txt /hivedatas/
```

追加加载

```
0: jdbc:hive2://node3:10000> load data inpath '/hivedatas/teacher.txt'
into table teacher;
No rows affected (0.247 seconds)
0: jdbc:hive2://node3:10000> select * from teacher;
```

```
+---------------+-----------------+--+
| teacher.t_id  | teacher.t_name  |
+---------------+-----------------+--+
| 01            | 张三              |
| 02            | 李四              |
| 03            | 王五              |
| 01            | 张三              |
| 02            | 李四              |
| 03            | 王五              |
+---------------+-----------------+--+
6 rows selected (0.132 seconds)
0: jdbc:hive2://node3:10000>
```

覆盖加载

```
0: jdbc:hive2://node3:10000> load data inpath '/hivedatas/teacher.txt'
overwrite into table teacher;
No rows affected (0.259 seconds)
0: jdbc:hive2://node3:10000> select * from teacher;
+---------------+-----------------+--+
| teacher.t_id  | teacher.t_name  |
+---------------+-----------------+--+
| 01            | 张三              |
| 02            | 李四              |
| 03            | 王五              |
+---------------+-----------------+--+
3 rows selected (0.119 seconds)
0: jdbc:hive2://node3:10000>
```

1-2-5 复杂类型操作

1、Array类型

Array是数组类型，Array中存放相同类型的数据

- 源数据

  说明:name与locations之间制表符分隔 '\t'，locations中元素之间逗号分隔','

  ```
  zhangsan    beijing,shanghai,tianjin,hangzhou
  wangwu  changchun,chengdu,wuhan,beijin
  ```

- 建表

  ```
  create external table hive_array(name string,work_locations array<string>)
  row format delimited fields terminated by '\t' collection items terminated
  by ',';
  ```

- 导入数据

```
load data local inpath '/export/data/work_locations.txt' overwrite into
table hive_array;
```

```
0: jdbc:hive2://node3:10000> create external table hive_array(name
string,work_locations array<string>) row format delimited fields terminated
by '\t' collection items terminated by ',';
No rows affected (0.084 seconds)
0: jdbc:hive2://node3:10000> load data local inpath
'/export/data/work_locations.txt' overwrite into table hive_array;
No rows affected (0.27 seconds)
0: jdbc:hive2://node3:10000> select * from hive_array;
+------------------+-------------------------------------------------+--+
| hive_array.name  |          hive_array.work_locations              |
+------------------+-------------------------------------------------+--+
| zhangsan         | ["beijing","shanghai","tianjin","hangzhou"]  |
| wangwu           | ["changchun","chengdu","wuhan","beijin"]     |
+------------------+-------------------------------------------------+--+
2 rows selected (0.119 seconds)
0: jdbc:hive2://node3:10000>
```

- 常用查询
  - 1- 查询所有数据

```
0: jdbc:hive2://node3:10000> select * from hive_array;
+------------------+-------------------------------------------------+--+
| hive_array.name  |          hive_array.work_locations              |
+------------------+-------------------------------------------------+--+
| zhangsan         | ["beijing","shanghai","tianjin","hangzhou"]  |
| wangwu           | ["changchun","chengdu","wuhan","beijin"]     |
+------------------+-------------------------------------------------+--+
2 rows selected (0.119 seconds)
0: jdbc:hive2://node3:10000>
```

  - 2- 查询loction数组中第一个元素 字段名[n]

```
0: jdbc:hive2://node3:10000> select name,work_locations[0] location from
hive_array;
+-----------+------------+--+
|   name    |  location  |
+-----------+------------+--+
| zhangsan  | beijing    |
| wangwu    | changchun  |
+-----------+------------+--+
2 rows selected (0.393 seconds)
0: jdbc:hive2://node3:10000>
```

- 3- 查询location数组中元素的个数 size()

```
0: jdbc:hive2://node3:10000> select name , size(work_locations)
location_num  from hive_array;
+-----------+---------------+--+
|   name    | location_num  |
+-----------+---------------+--+
| zhangsan  | 4             |
| wangwu    | 4             |
+-----------+---------------+--+
2 rows selected (0.101 seconds)
0: jdbc:hive2://node3:10000>
```

- 4- 查询location数组中包含tianjin的信息 array_contains(字段名，值)

```
0: jdbc:hive2://node3:10000> select * from hive_array where
array_contains(work_locations,'tianjin');
+-----------------+-------------------------------------------------+--+
| hive_array.name  |          hive_array.work_locations           |
+-----------------+-------------------------------------------------+--+
| zhangsan         | ["beijing","shanghai","tianjin","hangzhou"]  |
+-----------------+-------------------------------------------------+--+
1 row selected (0.087 seconds)
0: jdbc:hive2://node3:10000>
```

2、Map类型

- 数据准备

  说明：字段与字段分隔符: “,”；需要map字段之间的分隔符："#"；map内部k-v分隔符：":"

```
1,zhangsan,father:xiaoming#mother:xiaohuang#brother:xiaoxu,28
2,lisi,father:mayun#mother:huangyi#brother:guanyu,22
3,wangwu,father:wangjianlin#mother:ruhua#sister:jingtian,29
4,mayun,father:mayongzhen#mother:angelababy,26
```

- 创建表

```
create table if not exists hive_map(id int , name string, members
map<string,string>,age int) row format delimited fields terminated by ','
collection items terminated by '#' map keys terminated by ':';


0: jdbc:hive2://node3:10000> create table if not exists hive_map(id int ,
name string, members map<string,string>,age int) row format delimited fields
terminated by ',' collection items terminated by '#' map keys terminated by
':';

No rows affected (0.108 seconds)
0: jdbc:hive2://node3:10000> desc hive_map;
+-----------+---------------------+----------+--+
| col_name  |      data_type      | comment  |
+-----------+---------------------+----------+--+
| id        | int                 |          |
| name      | string              |          |
| members   | map<string,string>  |          |
| age       | int                 |          |
+-----------+---------------------+----------+--+
4 rows selected (0.055 seconds)
0: jdbc:hive2://node3:10000>
```

- 导入数据

```
load data local inpath '/export/data/hive_map.csv' overwrite into table
hive_map;

0: jdbc:hive2://node3:10000> load data local inpath
'/export/data/hive_map.csv' overwrite into table hive_map;
No rows affected (0.208 seconds)
0: jdbc:hive2://node3:10000>
```

- 常用查询

1. 查询所有信息

```
 select * from hive_map;

0: jdbc:hive2://node3:10000> select * from hive_map;
+--------------+----------------+-------------------------------------
-------------------------+---------------+--+
| hive_map.id  | hive_map.name  |
hive_map.members                        | hive_map.age  |
+--------------+----------------+-------------------------------------
-------------------------+---------------+--+
| 1            | zhangsan       |
{"father":"xiaoming","mother":"xiaohuang","brother":"xiaoxu"}  | 28
       |
```

```
| 2            | lisi           |
{"father":"mayun","mother":"huangyi","brother":"guanyu"}       | 22
       |
| 3            | wangwu         |
{"father":"wangjianlin","mother":"ruhua","sister":"jingtian"}  | 29
       |
| 4            | mayun          |
{"father":"mayongzhen","mother":"angelababy"}                  | 26
       |
+--------------+----------------+-------------------------------------
-------------------------+--------------+--+
4 rows selected (0.124 seconds)
0: jdbc:hive2://node3:10000>
```

2. 查询 member['father'] member['mother']

```
select id,name,age,members['father'] father,members['mother'] mother
from hive_map;

0: jdbc:hive2://node3:10000> select id,name,age,members['father']
father,members['mother'] mother from hive_map;
+-----+-----------+------+--------------+-------------+--+
| id  |   name    | age  |    father    |   mother    |
+-----+-----------+------+--------------+-------------+--+
| 1   | zhangsan  | 28   | xiaoming     | xiaohuang   |
| 2   | lisi      | 22   | mayun        | huangyi     |
| 3   | wangwu    | 29   | wangjianlin  | ruhua       |
| 4   | mayun     | 26   | mayongzhen   | angelababy  |
+-----+-----------+------+--------------+-------------+--+
4 rows selected (0.135 seconds)
0: jdbc:hive2://node3:10000>
```

3. 查询所有key map_keys()

```
select id,name,age,map_keys(members) relations from hive_map;

0: jdbc:hive2://node3:10000> select id,name,age,map_keys(members)
relations from hive_map;
+-----+-----------+------+--------------------------------+--+
| id  |   name    | age  |           relations            |
+-----+-----------+------+--------------------------------+--+
| 1   | zhangsan  | 28   | ["father","mother","brother"]  |
| 2   | lisi      | 22   | ["father","mother","brother"]  |
| 3   | wangwu    | 29   | ["father","mother","sister"]   |
| 4   | mayun     | 26   | ["father","mother"]            |
+-----+-----------+------+--------------------------------+--+
4 rows selected (0.04 seconds)
0: jdbc:hive2://node3:10000>
```

4. 查询所有values map_values()

```
select id ,name,age,map_values(members) relation_names from hive_map;

0: jdbc:hive2://node3:10000> select id ,name,age,map_values(members)
relation_names from hive_map;
+-----+-----------+------+-------------------------------------+--+
| id  |   name    | age  |           relation_names            |
+-----+-----------+------+-------------------------------------+--+
| 1   | zhangsan  | 28   | ["xiaoming","xiaohuang","xiaoxu"]   |
| 2   | lisi      | 22   | ["mayun","huangyi","guanyu"]        |
| 3   | wangwu    | 29   | ["wangjianlin","ruhua","jingtian"]  |
| 4   | mayun     | 26   | ["mayongzhen","angelababy"]         |
+-----+-----------+------+-------------------------------------+--+
4 rows selected (0.041 seconds)
0: jdbc:hive2://node3:10000>
```

5. 查询map中元素个数 size();

```
select id,name,age , size(members) num from hive_map;

0: jdbc:hive2://node3:10000> select id,name,age , size(members) num from
hive_map;
+-----+-----------+------+------+--+
| id  |   name    | age  | num  |
+-----+-----------+------+------+--+
| 1   | zhangsan  | 28   | 3    |
| 2   | lisi      | 22   | 3    |
| 3   | wangwu    | 29   | 3    |
| 4   | mayun     | 26   | 2    |
+-----+-----------+------+------+--+
4 rows selected (0.046 seconds)
0: jdbc:hive2://node3:10000>
```

6. 根据条件查询 array_contains(map_keys(),)

- 根据条件查询所有信息

```
select * from hive_map where
array_contains(map_keys(members),'brother');

0: jdbc:hive2://node3:10000> select * from hive_map where
array_contains(map_keys(members),'brother');
+--------------+----------------+--------------------------------------
-------------------------+---------------+--+
| hive_map.id  | hive_map.name  |
hive_map.members                        | hive_map.age  |
+--------------+----------------+--------------------------------------
-------------------------+---------------+--+
| 1            | zhangsan       |
{"father":"xiaoming","mother":"xiaohuang","brother":"xiaoxu"}  | 28
       |
| 2            | lisi           |
{"father":"mayun","mother":"huangyi","brother":"guanyu"}       | 22
       |
+--------------+----------------+--------------------------------------
-------------------------+---------------+--+
2 rows selected (0.044 seconds)
0: jdbc:hive2://node3:10000>
```

- 根据条件查询指定的亲属信息

```
select id,name,age,members['brother'] brother from hive_map where
array_contains(map_keys(members),'brother');

0: jdbc:hive2://node3:10000> select id,name,age,members['brother']
brother from hive_map where array_contains(map_keys(members),'brother');
+-----+-----------+------+----------+--+
| id  |   name    | age  | brother  |
+-----+-----------+------+----------+--+
| 1   | zhangsan  | 28   | xiaoxu   |
| 2   | lisi      | 22   | guanyu   |
+-----+-----------+------+----------+--+
2 rows selected (0.062 seconds)
0: jdbc:hive2://node3:10000>
```

3、Struct类型

- 数据准备

```
192.168.1.1#zhangsan:40
192.168.1.2#lisi:50
192.168.1.3#wangwu:60
192.168.1.4#zhaoliu:70
```

- 建表

```
```

```
create table if not exists hive_struct(ip string,info
struct<name:string,age:int>) row format delimited fields terminated by '#'
collection items terminated by ':';


0: jdbc:hive2://node3:10000> create table if not exists hive_struct(ip
string,info struct<name:string,age:int>) row format delimited fields
terminated by '#' collection items terminated by ':';
No rows affected (0.034 seconds)
0: jdbc:hive2://node3:10000> desc hive_struct;
+-----------+-----------------------------+----------+--+
| col_name  |          data_type          | comment  |
+-----------+-----------------------------+----------+--+
| ip        | string                      |          |
| info      | struct<name:string,age:int> |          |
+-----------+-----------------------------+----------+--+
2 rows selected (0.024 seconds)
0: jdbc:hive2://node3:10000>
```

- 导入数据

```
load data local inpath '/export/data/hive_struct.csv' overwrite into table
hive_struct;

0: jdbc:hive2://node3:10000> load data local inpath
'/export/data/hive_struct.csv' overwrite into table hive_struct;
No rows affected (0.094 seconds)
0: jdbc:hive2://node3:10000>
```

- 常用查询
  - 查询所有信息

```
select * from hive_struct;

0: jdbc:hive2://node3:10000> select * from hive_struct;
+-----------------+-------------------------------+--+
| hive_struct.ip  |       hive_struct.info        |
+-----------------+-------------------------------+--+
| 192.168.1.1     | {"name":"zhangsan","age":40}  |
| 192.168.1.2     | {"name":"lisi","age":50}      |
| 192.168.1.3     | {"name":"wangwu","age":60}    |
| 192.168.1.4     | {"name":"zhaoliu","age":70}   |
+-----------------+-------------------------------+--+
4 rows selected (0.059 seconds)
0: jdbc:hive2://node3:10000>
```

  - 分字段查询详细信息

```
select ip,info.name name,info.age age from hive_struct;
```

```
0: jdbc:hive2://node3:10000> select ip,info.name name,info.age age from
hive_struct;
+--------------+-----------+------+--+
|      ip      |   name    | age  |
+--------------+-----------+------+--+
| 192.168.1.1  | zhangsan  | 40   |
| 192.168.1.2  | lisi      | 50   |
| 192.168.1.3  | wangwu    | 60   |
| 192.168.1.4  | zhaoliu   | 70   |
+--------------+-----------+------+--+
4 rows selected (0.039 seconds)
0: jdbc:hive2://node3:10000>
```

- 条件查询

```
select ip,info.name name,info.age age from hive_struct where info.age>55;

0: jdbc:hive2://node3:10000> select ip,info.name name,info.age age from
hive_struct where info.age>55;
+--------------+----------+------+--+
|      ip      |   name   | age  |
+--------------+----------+------+--+
| 192.168.1.3  | wangwu   | 60   |
| 192.168.1.4  | zhaoliu  | 70   |
+--------------+----------+------+--+
2 rows selected (0.037 seconds)
0: jdbc:hive2://node3:10000>
```

1-2-6 分区表

- 在大数据中，最常用的一种思想就是分治，分区表实际就是对应hdfs文件系统上的的独立的文件夹，该文件夹下是该分区所有数据文件。
- 分区可以理解为分类，通过分类把不同类型的数据放到不同的目录下。
- 分类的标准就是分区字段，可以一个，也可以多个。
- 分区表的意义在于优化查询；查询时尽量利用分区字段；如果不使用分区字段，就会全部扫描。
- 在查询是通过where子句查询来指定所需的分区。

1、单级分区- 创建分区表语法 partitioned by

```
create table score(sid string ,cid string ,sscore int) partitioned by (分区名 分区
类型) row format delimited fields terminated by '\t';
```

```
create table score(sid string ,cid string ,sscore int) partitioned by (month
string) row format delimited fields terminated by '\t';
```

2、单级分区- 加载数据到分区表中

- 加载一月的成绩

```
load data local inpath '/export/data/score-01.csv'  into table score partition
(month='202001');
```

- 加载二月的成绩

```
load data local inpath '/export/data/score-02.csv' into table score partition
(month='202002');
```

- 加载三月的成绩

```
load data local inpath '/export/data/score-03.csv' into table score partition
(month='202003');
```

- 加载四月的成绩

```
load data local inpath '/export/data/score-04.csv' into table score partition
(month='202004');
```

- 加载五月的成绩

```
load data local inpath '/export/data/score-05.csv' into table score partition
(month='202005');
```

3、单级分区- 查看表结构

- 可以看到 partition information

```
0: jdbc:hive2://node3:10000> desc score;
+--------------------------+----------------------+------------------------+--+
|         col_name         |       data_type      |        comment         |
+--------------------------+----------------------+------------------------+--+
| sid                      | string               |                        |
| cid                      | string               |                        |
| sscore                   | int                  |                        |
| month                    | string               |                        |
|                          | NULL                 | NULL                   |
| # Partition Information  | NULL                 | NULL                   |
| # col_name               | data_type            | comment                |
|                          | NULL                 | NULL                   |
| month                    | string               |                        |
+--------------------------+----------------------+------------------------+--+
9 rows selected (0.048 seconds)
0: jdbc:hive2://node3:10000>
```

4、单级分区- 查看分区

```
show partitions 表名;
```

```
0: jdbc:hive2://node3:10000> show partitions score;
+---------------+--+
|   partition   |
+---------------+--+
| month=202001  |
| month=202002  |
| month=202003  |
| month=202004  |
| month=202005  |
+---------------+--+
5 rows selected (0.085 seconds)
0: jdbc:hive2://node3:10000>
```

5、单级分区- 根据分区条件查询

```
select * from where 分区名称=分区值;
```

```
0: jdbc:hive2://node3:10000> select * from score where month like '%01';  # 查询一月的成绩
+------------+------------+---------------+--------------+--+
| score.sid  | score.cid  | score.sscore  | score.month  |
+------------+------------+---------------+--------------+--+
| 01         | 01         | 81            | 202001       |
| 01         | 02         | 91            | 202001       |
| 01         | 03         | 91            | 202001       |
| 02         | 01         | 71            | 202001       |
| 02         | 02         | 61            | 202001       |
| 02         | 03         | 81            | 202001       |
| 03         | 01         | 81            | 202001       |
| 03         | 02         | 81            | 202001       |
| 03         | 03         | 81            | 202001       |
| 04         | 01         | 51            | 202001       |
0: jdbc:hive2://node3:10000> select * from score where month = '202002';    # 查询二月的成绩
+------------+------------+---------------+--------------+--+
| score.sid  | score.cid  | score.sscore  | score.month  |
+------------+------------+---------------+--------------+--+
| 01         | 01         | 82            | 202002       |
| 01         | 02         | 92            | 202002       |
| 01         | 03         | 92            | 202002       |
| 02         | 01         | 72            | 202002       |
| 02         | 02         | 62            | 202002       |
| 02         | 03         | 82            | 202002       |
| 03         | 01         | 82            | 202002       |
| 03         | 02         | 82            | 202002       |
| 03         | 03         | 82            | 202002       |
| 04         | 01         | 52            | 202002       |
```

6、单级分区- 添加一个分区

```
alter table 表名 add partition(分区名=分区值);
```

```
0: jdbc:hive2://node3:10000> alter table score add partition(month='202006');
No rows affected (0.06 seconds)
0: jdbc:hive2://node3:10000> show partitions score;
+---------------+--+
|   partition   |
+---------------+--+
| month=202001  |
| month=202002  |
| month=202003  |
| month=202004  |
| month=202005  |
| month=202006  |
+---------------+--+
```

7、单级分区- 同时添加多个分区

```
alter table 表名 add partition(分区名=分区值) partition(分区名=分区值);
```

```
0: jdbc:hive2://node3:10000> alter table score add partition(month='202005')
partition(month='202006');
No rows affected (0.061 seconds)
0: jdbc:hive2://node3:10000> show partitions score;
+---------------+--+
|   partition   |
+---------------+--+
| month=202001  |
| month=202002  |
| month=202003  |
| month=202004  |
| month=202005  |
| month=202006  |
+---------------+--+
6 rows selected (0.066 seconds)
```

8、单级分区- 删除分区

```
alter table 表名 drop patition(分区名=分区值);
```

```
0: jdbc:hive2://node3:10000> alter table score drop partition(month='202005');
No rows affected (0.124 seconds)
0: jdbc:hive2://node3:10000> alter table score drop partition(month='202006');
No rows affected (0.119 seconds)
```

```
0: jdbc:hive2://node3:10000> show partitions score;
+---------------+--+
|   partition   |
+---------------+--+
| month=202001  |
| month=202002  |
| month=202003  |
| month=202004  |
+---------------+--+
4 rows selected (0.065 seconds)
```

9、多分区联合查询 union all

把两次查询结果拼接在一起；

```
select
    *
from
    表名
where
    month = '202006'

union all

select
    *
from
    表名
where
    month = '202007';
```

```
select * from score where month = '202002' union all select * from score where
month = '202001'
```

1、多级分区- 创建分区表语法 partitioned by

```
create table score2 (sid string,cid string , sscore int) partitioned by (分区名1=
分区值1，分区名2=分区值2，分区名3=分区值3) row format delimited fields terminated by
'\t';
```

```
create table score2 (sid string,cid string , sscore int) partitioned by (year
string,month string,day string) row format delimited fields terminated by '\t';
```

2、多级分区- 加载数据到分区表中

- 加载2020-01-01的成绩

```
load data local inpath '/export/data/score2020_01_01.csv' into table score2
partition(year='2020',month='01',day='01');
```

- 加载2020-02-02的成绩

```
load data local inpath '/export/data/score2020_02_02.csv' into table score2
partition(year='2020',month='02',day='02');
```

- 加载2021-03-03的成绩

```
load data loca inpath '/export/data/score2021_03_03.csv' into table score2
partition(year='2021',month='03',day='03');
```

- 加载2021-04-04的成绩

```
load data local inpath '/export/data/score2021_04_04.csv' into table score2
partition(year='2021',month='04',day='04');
```

3、多级分区- 查看表结构

- 可以看到 partition information

```
desc 表名;
```

```
0: jdbc:hive2://node3:10000> desc score2;
+--------------------------+----------------------+------------------------+--+
|         col_name         |       data_type      |        comment         |
+--------------------------+----------------------+------------------------+--+
| sid                      | string               |                        |
| cid                      | string               |                        |
| sscore                   | int                  |                        |
| year                     | string               |                        |
| month                    | string               |                        |
| day                      | string               |                        |
|                          | NULL                 | NULL                   |
| # Partition Information  | NULL                 | NULL                   |
| # col_name               | data_type            | comment                |
|                          | NULL                 | NULL                   |
| year                     | string               |                        |
| month                    | string               |                        |
| day                      | string               |                        |
+--------------------------+----------------------+------------------------+--+
13 rows selected (0.041 seconds)
```

4、多级分区- 查看分区

```
show partitions 表名;
```

```
0: jdbc:hive2://node3:10000> show partitions score2;
+----------------------------+--+
|         partition          |
+----------------------------+--+
| year=2020/month=01/day=01  |
| year=2020/month=02/day=02  |
| year=2021/month=03/day=03  |
| year=2021/month=04/day=04  |
| year=2022/month=05/day=05  |
+----------------------------+--+
5 rows selected (0.066 seconds)
0: jdbc:hive2://node3:10000>
```

5、多级分区- 根据分区条件查询

```
select * from where 分区名称=分区值 and 分区名=分区值;
```

```
select * from score2 where year='2020' and month='02' and day='02';
```

6、多级分区- 添加分区

```
alter table 表名 add partition(分区名1=分区值1，分区名2=分区值2，分区名3=分区值3);
```

```
alter table score2 add partition(year='2022',month='05',day='01');
```

7、多级分区- 同时添加多个分区

```
alter table 表名 add partition(分区名1=分区值1，分区名2=分区值2，分区名3=分区值3)
partition(分区名1=分区值1，分区名2=分区值2，分区名3=分区值3);
```

```
0: jdbc:hive2://node3:10000> alter table score2 add
partition(year='2023',month='06',day='02')
partition(year='2024',month='07',day='01');
No rows affected (0.068 seconds)
0: jdbc:hive2://node3:10000> show partitions score2;
+----------------------------+--+
|         partition          |
+----------------------------+--+
| year=2020/month=01/day=01  |
| year=2020/month=02/day=02  |
| year=2021/month=03/day=03  |
| year=2021/month=04/day=04  |
| year=2022/month=05/day=01  |
```

```
| year=2022/month=05/day=05  |
| year=2023/month=06/day=02  |
| year=2024/month=07/day=01  |
+----------------------------+--+
8 rows selected (0.048 seconds)
0: jdbc:hive2://node3:10000>
```

8、多级分区- 删除分区

```
alter table 表名 drop patition(分区名1=分区值1，分区名2=分区值2，分区名3=分区值3);
```

```
0: jdbc:hive2://node3:10000> alter table score2 drop
partition(year='2024',month='07',day='01');
No rows affected (0.084 seconds)
0: jdbc:hive2://node3:10000> alter table score2 drop
partition(year='2023',month='06',day='02');
No rows affected (0.08 seconds)
0: jdbc:hive2://node3:10000> alter table score2 drop
partition(year='2022',month='05',day='05');
No rows affected (0.069 seconds)
0: jdbc:hive2://node3:10000> show partitions score2;
+----------------------------+--+
|         partition          |
+----------------------------+--+
| year=2020/month=01/day=01  |
| year=2020/month=02/day=02  |
| year=2021/month=03/day=03  |
| year=2021/month=04/day=04  |
| year=2022/month=05/day=01  |
+----------------------------+--+
5 rows selected (0.042 seconds)
0: jdbc:hive2://node3:10000>
```

多分区联合查询使用union all来实现

1-2-7 分桶表

- 分桶就是将数据划分到不同的文件，其实就是MapReduce的分区;
- 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;
- 分区规则：

  字段.hash%分区数量=?

  某个字段的hash值 与 分区的数量取模；
- 桶表的数据加载，由于桶表的数据加载通过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

0、数据准备

```
01  语文  02
02  数学  01
03  英语  03
```

1、开启hive的桶表功能

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

注意：开启hive的桶表功能(如果执行该命令报错，表示这个版本的Hive已经自动开启了分桶功能，则直接进行下一步)

2、设置reduce的个数

```
set mapreduce.job.reduces=分桶个数
```

3、创建分桶表 clustered by (字段名) into 分区数量 buckets;

```
 create table course_cluster(c_id string,c_name string , t_id string) clustered
by (根据哪个字段分区) into 分区数量 buckets row format delimited fields terminated by
'\t';
```

```
0: jdbc:hive2://node3:10000> create table course_cluster(c_id string,c_name
string , t_id string) clustered by (c_id) into 3 buckets row format delimited
fields terminated by '\t';
No rows affected (0.059 seconds)
0: jdbc:hive2://node3:10000>
```

4、创建普通表

```
0: jdbc:hive2://node3:10000> create table course_common (c_id string,c_name
string , t_id string) row format delimited fields terminated by '\t';
No rows affected (0.042 seconds)
```

5、普通表中加载数据

```
0: jdbc:hive2://node3:10000> load data local inpath '/export/data/course.csv'
into table course_common;
No rows affected (0.141 seconds)
0: jdbc:hive2://node3:10000> select * from course_common;
+--------------------+----------------------+--------------------+--+
| course_common.c_id  | course_common.c_name  | course_common.t_id  |
+--------------------+----------------------+--------------------+--+
| 01                 | 语文                   | 02                 |
| 02                 | 数学                   | 01                 |
| 03                 | 英语                   | 03                 |
+--------------------+----------------------+--------------------+--+
3 rows selected (0.064 seconds)
```

6、通过insert overwrite给桶表中加载数据

注意： 桶表的数据加载，由于桶表的数据加载通过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

```
0: jdbc:hive2://node3:10000> insert overwrite table course_cluster select * from
course_common cluster by (c_id);
No rows affected (36.431 seconds)
0: jdbc:hive2://node3:10000>
```

1-2-8 修改表

1、表重命名

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

```
0: jdbc:hive2://node3:10000> alter table stu rename to stu1;
No rows affected (0.098 seconds)
0: jdbc:hive2://node3:10000> show tables;
+-------------+--+
|  tab_name   |
+-------------+--+
| hive_array  |
| stu1        |
| stu2        |
| student     |
| teacher     |
+-------------+--+
5 rows selected (0.038 seconds)
0: jdbc:hive2://node3:10000>
```

2、增加/修改列信息

- 添加列 add

```
alter table 表名 add columns (新列名,新列类型);
```

```
0: jdbc:hive2://node3:10000> alter table stu add columns (score int);
No rows affected (0.094 seconds)
0: jdbc:hive2://node3:10000>
0: jdbc:hive2://node3:10000> desc stu;
+-----------+------------+----------+--+
| col_name  | data_type  | comment  |
+-----------+------------+----------+--+
| id        | int        |          |
| name      | string     |          |
| age       | int        |          |
| address   | string     |          |
| score     | int        |          |
+-----------+------------+----------+--+
5 rows selected (0.07 seconds)
0: jdbc:hive2://node3:10000>
```

- 更新列 （更新列时不能更新类型）

```
alter table 表名 change column old列名  new列名 类型;
```

```
0: jdbc:hive2://node3:10000> alter table stu change column score
English_score int;
No rows affected (0.11 seconds)
0: jdbc:hive2://node3:10000> desc stu;
+----------------+------------+----------+--+
|    col_name    | data_type  | comment  |
+----------------+------------+----------+--+
| id             | int        |          |
| name           | string     |          |
| age            | int        |          |
| address        | string     |          |
| english_score  | int        |          |
+----------------+------------+----------+--+
5 rows selected (0.054 seconds)
0: jdbc:hive2://node3:10000>
```

3、删除表

```
drop  table 表名
```

```
0: jdbc:hive2://node3:10000> drop table stu;
No rows affected (0.128 seconds)
0: jdbc:hive2://node3:10000> show tables;
+-------------+--+
|  tab_name   |
+-------------+--+
| hive_array  |
| stu2        |
| student     |
| teacher     |
+-------------+--+
4 rows selected (0.04 seconds)
0: jdbc:hive2://node3:10000>
```

4、清空表数据

```
truncate table 表名;
```

```
0: jdbc:hive2://node3:10000> truncate table stu;
No rows affected (0.12 seconds)
0: jdbc:hive2://node3:10000> select * from stu;
+---------+-----------+----------+-------------+--------------------+--+
| stu.id  | stu.name  | stu.age  | stu.address  | stu.english_score  |
+---------+-----------+----------+-------------+--------------------+--+
+---------+-----------+----------+-------------+--------------------+--+
No rows selected (0.104 seconds)
0: jdbc:hive2://node3:10000>
```

1-2-9 hive表中加载数据

1、直接向分区表中插入数据

- 通过insert into方式加载数据

  注意： 如果是分区表， insert into 时必须要指定分区；

  ```
  create table score3 like score;
  insert into table score3 partition(month ='202007') values
  ('001','002',100);
  ```

- ***通过查询方式加载数据 insert overwrite table xxx partition(xxxx) select xxx**

```
create table score4 like score;
insert overwrite table score4 partition(month = '202006') select
sid,cid,sscore from score;
```

注意： 后面的select 语句不能使用select * from score；因为这个score 里面有个分区month，这样差相当于有四个字段。

```
0: jdbc:hive2://node3:10000> insert overwrite table score4
partition(month='202008') select * from score;
Error: Error while compiling statement: FAILED: SemanticException [Error
10044]: Line 1:23 Cannot insert into target table because column
number/types are different ''202008'': Table insclause-0 has 3 columns, but
query has 4 columns. (state=42000,code=10044)
0: jdbc:hive2://node3:10000>
```

- ***通过load方式加载数据**

```
load data local inpath '/export/data/score-04.csv' overwrite into table
```

2、多插入模式（一个表分为多个表）

语法

```
from 总表
    insert overwrite table 表1 partition(分区1)  select xx,xx,...
    insert overwrite table 表2 partition(分区2)  select xx,xx,...;
```

- 使用场景：
  - 常用于实际生产环境当中，将一张表拆开成两部分或者多部分

案例：

- 给score表加载数据

```
0: jdbc:hive2://node3:10000> load data local inpath '/export/data/score-
04.csv' overwrite into table score5 partition(month='202004');
No rows affected (0.409 seconds)
0: jdbc:hive2://node3:10000> select * from score5;
+-------------+-------------+----------------+---------------+--+
| score5.sid  | score5.cid  | score5.sscore  | score5.month  |
+-------------+-------------+----------------+---------------+--+
| 01          | 01          | 84             | 202004        |
| 01          | 02          | 94             | 202004        |
| 01          | 03          | 94             | 202004        |
| 02          | 01          | 74             | 202004        |
| 02          | 02          | 64             | 202004        |
| 02          | 03          | 84             | 202004        |
| 03          | 01          | 84             | 202004        |
| 03          | 02          | 84             | 202004        |
```

```
| 03          | 03          | 84              | 202004        |
| 04          | 01          | 54              | 202004        |
| 04          | 02          | 34              | 202004        |
| 04          | 03          | 24              | 202004        |
| 05          | 01          | 74              | 202004        |
| 05          | 02          | 84              | 202004        |
| 06          | 01          | 34              | 202004        |
| 06          | 03          | 34              | 202004        |
| 07          | 02          | 84              | 202004        |
| 07          | 03          | 94              | 202004        |
+-------------+-------------+-----------------+---------------+--+
18 rows selected (0.115 seconds)
```

- 创建第一部分表

```
0: jdbc:hive2://node3:10000> create table score_first(sid string ,cid
string)partitioned by (month string) row format delimited fields terminated
by '\t';
No rows affected (0.085 seconds)
```

- 创建第二部分表

```
0: jdbc:hive2://node3:10000> create table score_sencond(cid string,sscore
int) partitioned by (month string) row format delimited fields terminated by
'\t';
```

- 分别给第一部分与第二部分表加载数据

```
0: jdbc:hive2://node3:10000> from score5 insert overwrite table score_first
partition(month='202004') select sid,cid insert overwrite table
score_sencond partition(month='202004') select cid,sscore;
No rows affected (11.417 seconds)
0: jdbc:hive2://node3:10000> select * from score_first;
+------------------+------------------+--------------------+--+
| score_first.sid  | score_first.cid  | score_first.month  |
+------------------+------------------+--------------------+--+
| 01               | 01               | 202004             |
| 01               | 02               | 202004             |
| 01               | 03               | 202004             |
| 02               | 01               | 202004             |
| 02               | 02               | 202004             |
| 02               | 03               | 202004             |
| 03               | 01               | 202004             |
| 03               | 02               | 202004             |
| 03               | 03               | 202004             |
| 04               | 01               | 202004             |
| 04               | 02               | 202004             |
| 04               | 03               | 202004             |
```

```
| 05                | 01                | 202004              |
| 05                | 02                | 202004              |
| 06                | 01                | 202004              |
| 06                | 03                | 202004              |
| 07                | 02                | 202004              |
| 07                | 03                | 202004              |
+-------------------+-------------------+---------------------+--+
18 rows selected (0.13 seconds)
0: jdbc:hive2://node3:10000> select * from score_sencond;
+---------------------+-----------------------+-----------------------+--+
| score_sencond.cid   | score_sencond.sscore  | score_sencond.month   |
+---------------------+-----------------------+-----------------------+--+
| 01                  | 84                    | 202004                |
| 02                  | 94                    | 202004                |
| 03                  | 94                    | 202004                |
| 01                  | 74                    | 202004                |
| 02                  | 64                    | 202004                |
| 03                  | 84                    | 202004                |
| 01                  | 84                    | 202004                |
| 02                  | 84                    | 202004                |
| 03                  | 84                    | 202004                |
| 01                  | 54                    | 202004                |
| 02                  | 34                    | 202004                |
| 03                  | 24                    | 202004                |
| 01                  | 74                    | 202004                |
| 02                  | 84                    | 202004                |
| 01                  | 34                    | 202004                |
| 03                  | 34                    | 202004                |
| 02                  | 84                    | 202004                |
| 03                  | 94                    | 202004                |
+---------------------+-----------------------+-----------------------+--+
18 rows selected (0.112 seconds)
0: jdbc:hive2://node3:10000>
```

1-2-10 hive表中的数据导出

1、insert导出

- 将查询的结果导出到本地
  - 会自动创建目录
  - 不能指定文件名

```
insert overwrite local directory '/export/data/exporthive' select * from
score;
```

```
[root@node3 exporthive]# pwd
/export/data/exporthive
[root@node3 exporthive]# ls
000000_0
[root@node3 exporthive]# cat 000000_0
010181202001
010291202001
010391202001
020171202001
020261202001
```

- 将查询的结果 **格式化** 导出到本地 row format xxx
  - row format delimited fields terminated by ''

```
insert overwrite local directory '/export/data/exporthive' row format
delimited fields terminated by '\t' select * from score;
```

```
[root@node3 exporthive]# pwd
/export/data/exporthive
[root@node3 exporthive]# cat 000000_0
01  01  81  202001
01  02  91  202001
01  03  91  202001
02  01  71  202001
02  02  61  202001
02  03  81  202001
03  01  81  202001
03  02  81  202001
```

  - row format delimited fields terminated by + collection items terminated by

```
0: jdbc:hive2://node3:10000> select * from hive_array;
+-----------------+-----------------------------------------+--+
| hive_array.name  |          hive_array.work_locations          |
+-----------------+-----------------------------------------+--+
| zhangsan         | ["beijing","shanghai","tianjin","hangzhou"]  |
| wangwu           | ["changchun","chengdu","wuhan","beijin"]     |
+-----------------+-----------------------------------------+--+
2 rows selected (0.085 seconds)
0: jdbc:hive2://node3:10000> insert overwrite local directory
'/export/data/exporthive/hivearray' row format delimited fields terminated
by
'\t' collection items terminated by '#' select * from hive_array;
No rows affected (9.649 seconds)
0: jdbc:hive2://node3:10000>
```

```
[root@node3 hivearray]# cat 000000_0
zhangsan    beijing#shanghai#tianjin#hangzhou
wangwu  changchun#chengdu#wuhan#beijin
[root@node3 hivearray]#
```

- 将查询的结果导出到HDFS上(没有local)

```
0: jdbc:hive2://node3:10000> insert overwrite directory '/exporthive' row
format delimited fields terminated by '\t' select * from score;
No rows affected (11.512 seconds)
0: jdbc:hive2://node3:10000>
```

```
[root@node3 exporthive]# hadoop fs -cat /exporthive/000000_0
01  01  81  202001
01  02  91  202001
01  03  91  202001
02  01  71  202001
02  02  61  202001
02  03  81  202001
03  01  81  202001
03  02  81  202001
```

2、hive shell 命令导出

- 基本语法：（hive -f/-e 执行语句或者脚本 > file）

```
hive -e 'select * from 数据库名.表名' >  file 路径
```

```
[root@node3 exporthive]# hive -e "select * from myhive2.score" >
/export/data/exporthive/score.txt
which: no hbase in (:/export/server/hive-2.1.0/bin::/export/server/hadoop-
2.7.5/bin:/export/server/hadoop-
2.7.5/sbin::/export/shell/bin::/export/server/zookeeper-
3.4.6/bin::/export/server/jdk1.8.0_241/bin:/usr/local/sbin:/usr/local/bin:/u
sr/sbin:/usr/bin:/export/server/mysql-5.7.29/bin:/root/bin)
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/export/server/hive-2.1.0/lib/hive-jdbc-
2.1.0-standalone.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/export/server/hive-2.1.0/lib/log4j-slf4j-
impl-2.4.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/export/server/hadoop-
2.7.5/share/hadoop/common/lib/slf4j-log4j12-
1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an
explanation.
```

```
SLF4J: Actual binding is of type
[org.apache.logging.slf4j.Log4jLoggerFactory]

Logging initialized using configuration in jar:file:/export/server/hive-
2.1.0/lib/hive-common-2.1.0.jar!/hive-log4j2.properties Async: true
OK
Time taken: 3.036 seconds, Fetched: 72 row(s)
[root@node3 exporthive]# ls
000000_0  score.txt
[root@node3 exporthive]# cat score.txt
01  01  81  202001
01  02  91  202001
01  03  91  202001
02  01  71  202001
02  02  61  202001
02  03  81  202001
03  01  81  202001
```

3、export导出到HDFS上

- export 命令 这个相当于 hadoop fs -cp 命令；

```
export table 表名 to 'hdfs目录';
```

```
0: jdbc:hive2://node3:10000> export table score to '/exporthive/score';
No rows affected (0.505 seconds)
0: jdbc:hive2://node3:10000>
```

4、sqoop导出

2- hive查询语法

2-1 SELECT语句

2-1-1 语句结构

- 语法

```
SELECT [ALL | DISTINCT]select_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[HAVING where_condition]   // 对分组后的条件筛选
[ORDER BY col_list]
[CLUSTER BY col_list
  | [DISTRIBUTE BY col_list] [SORT BY col_list]
]
[LIMIT number]
```

- 解释

```
1、ORDER BY用于全局排序，就是对指定的所有排序键进行全局排序，使用ORDER BY的查询语句，最后会用一个Reduce Task来完成全局排序。
2、HAVING 对分组后的条件筛选
3、sort by用于分区内排序，即每个Reduce任务内排序。，则sort by只保证每个reducer的输出有序，不保证全局有序。
4、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。
5、cluster by(字段) 除了具有Distribute by的功能外，还兼具sort by的排序功能。。
因此，如果distribute by和sort by字段是同一个时，此时，cluster by = distribute by + sort by
6、LIMIT ， 显示多少行数据； limit n,m ： 显示从n开始到m
```

2-1-2 全表查询

```
select * from score;
```

2-1-3 选择特定列查询

```
select sid , cid ,sscore from score ;
```

2-1-4 列别名 as (可以省略)

```
select sid as myid ,cid from score;
```

2-1-5 常用聚合函数

2-1-5-1 求总行数（count）

```
1）求总行数（count）
 select count(1) from score;
```

2-1-5-2 求分数的最大值（max）

```
2）求分数的最大值（max）
 select max(sscore) from score;
```

2-1-5-3 求分数的最小值（min）

```
3）求分数的最小值（min）
 select min(sscore) from score;
```

2-1-5-4 求分数的总和（sum）

```
4）求分数的总和（sum）
 select sum(sscore) from score;
```

2-1-5-5 求分数的平均值（avg）

```
5）求分数的平均值（avg）
 select avg(sscore) from score;
```

2-1-6 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 表名 limit n,m  -- 从n开始，显示m条数据 --
```

2-1-7 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 sscore > 60;
```

2-2 运算符

2-2-1 比较运算符

- 操作符

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。

操作符	支持的数据类型	描述
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。

- 案例操作

```
（1）查询分数等于80的所有的数据
select * from score where sscore = 80;
（2）查询分数在80到100的所有数据
select * from score where sscore between 80 and 100;
（3）查询成绩为空的所有数据
select * from score where sscore is null;
（4）查询成绩是80或 90的数据
select * from score where sscore in(80,90);
```

- like 和 rlike
- 使用LIKE运算选择类似的值
- 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。
- 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。
- 案例实操

```
（1）查找以8开头的所有成绩
select * from score where sscore like '8%';
（2）查找第二个数值为9的所有成绩数据
select * from score where sscore like '_9%';
（3）查找id中含1的所有成绩信息
select * from score where sid rlike '[1]';
```

2-2-2 逻辑运算符

操作符	含义
AND	逻辑并
OR	逻辑或
NOT	逻辑否

- 案例操作

```
（1）查询成绩大于80，并且sid是01的数据
select * from score where sscore >80 and sid = '01';
（2）查询成绩大于80，或者sid  是01的数
select * from score where sscore > 80 or sid = '01';
（3）查询sid  不是 01和02的学生
select * from score where sid not in ('01','02');
```

2-3 分组

2-3-1 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。

注意：使用group by分组之后，select后面的字段**只能是 分组字段 聚合函数 。**

- 案例实操：

```
（1）计算每个学生的平均分数
select sid ,avg(sscore) from score group by sid;

（2）计算每个学生最高成绩
select sid ,max(sscore) from score group by sid;
```

2-3-2 HAVING语句 只用于 group by 后面

- 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。**

- 案例实操：

```
-- 求每个学生的平均分数
select sid ,avg(sscore) from score group by sid;

-- 求每个学生平均分数大于85的人
select sid ,avg(sscore) avgscore from score group by sid having avgscore > 85;
```

2-4 JOIN语句

注意： **Hive的join操作只支持等值连接**

2-4-1 内连接（INNER JOIN）

- 内连接：（求交集）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 teacher t, course c where t.tid = c.tid; #隐式内连接
select * from teacher t inner join course c on t.tid = c.tid; #显式内连接
select * from teacher t join course c on t.tid = c.tid;
```

2-4-2 左外连接（LEFT OUTER JOIN on）

- 左外连接：(显示左表所有信息)JOIN操作符左边表中符合WHERE子句的所有记录将会被返回。

```
select * from teacher t left join course c on t.tid = c.tid;
```

2-4-3 右外连接（RIGHT OUTER JOIN）

- 右外连接：(显示右表所有信息)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.tid = c.tid;
```

2-4-4 满外连接（FULL OUTER JOIN）

- 满外连接：(两个表都全部显示)将会返回所有表中符合WHERE语句条件的所有记录。如果任一表的指定字段没有符合条件的值的话，那么就使用NULL值替代。

```
SELECT * FROM teacher t FULL JOIN course c ON t.tid = c.tid ;
```

2-4-5 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.tid = c.tid
left join score s
    on s.cid = c.cid
left join student stu
    on s.sid = stu.sid;
```

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teacher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。

2-5 排序

2-5-1 Order By-全局排序

- Order By：全局排序，一个reduce
- 使用 ORDER BY 子句排序
  - ASC（ascend）: 升序（默认）
  - DESC（descend）: 降序
- order by 后面可以跟多个字段排序，先根据前面的字段排序，如果前面的字段相等再根据后面的字段排序；
- ORDER BY 子句在SELECT语句的结尾。
- 案例操作

```
（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.sid = sco.sid ORDER BY
sco.sscore DESC;
```

```
(2)按照分数的平均值排序
select sid ,avg(sscore) avg from score group by sid order by avg;
```

```
(3)按照学生id和平均成绩进行排序
select sid ,avg(sscore) avg from score group by sid order by sid,avg;
```

2-5-2 Sort By-每个MapReduce内部局部排序

- Sort By：每个MapReduce内部进行排序，对全局结果集来说不是排序。
- 将表文件分成多份，并对每个reduce的输出结果进行排序；

```
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 sscore;
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/data/exporthive/sort' select * from
score sort by sscore;
```

2-5-3 Distribute By-分区排序 （分组 + 排序）

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。

distribute by + sort by：distribute by 实现分区，sort by 对每个分区的数据进行排序；

- 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sid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/data/exporthive/distribute' select *
from score distribute by sid sort by sscore;
```

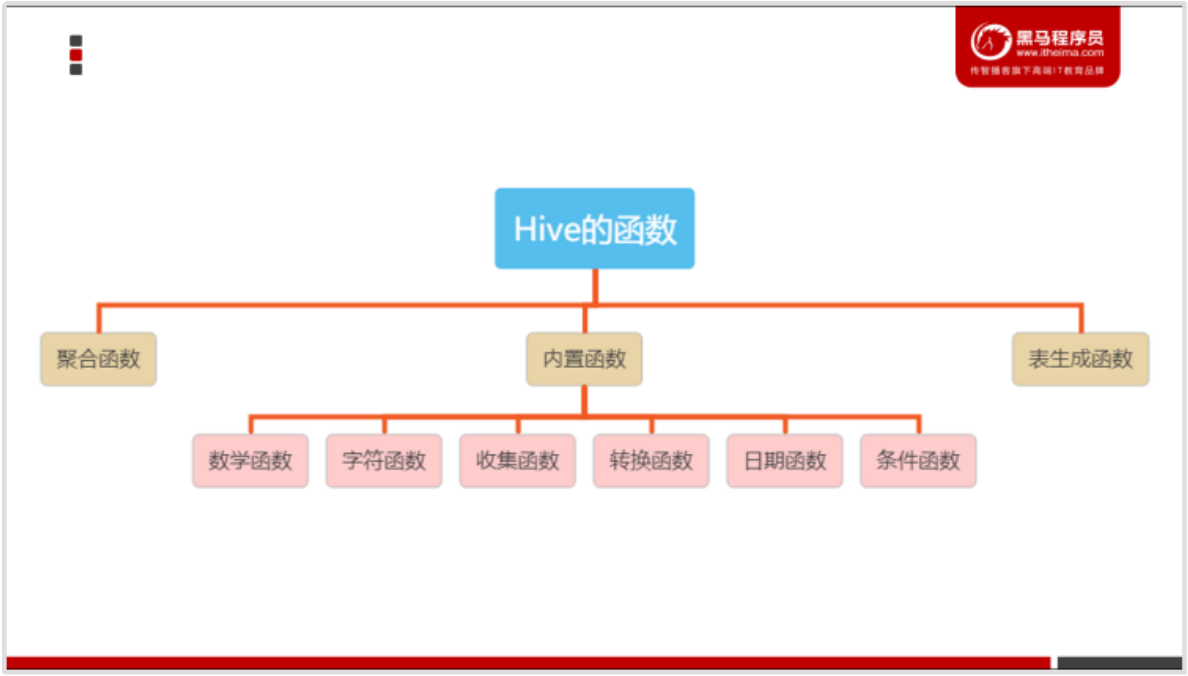
2-5-4 Cluster By

- 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 **只能是升序** 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 sid;
select * from score distribute by sid sort by sid;  # 根据sid 分区 sid 排序
```

3- Hive函数



- Hive的函数分为三类： 聚合函数、内置函数，表生成函数;

3-1 Hive的内置函数

3-1-1 聚合函数

####

3-1-1-1 求总行数（count）

```
1）求总行数（count）
 select count(1) from score;
```

3-1-1-2 求分数的最大值（max）

```
2）求分数的最大值（max）
 select max(sscore) from score;
```

3-1-1-3 求分数的最小值（min）

```
3）求分数的最小值（min）
 select min(sscore) from score;
```

3-1-1-4 求分数的总和（sum）

```
4）求分数的总和（sum）
 select sum(sscore) from score;
```

3-1-1-5 求分数的平均值（avg）

```
5）求分数的平均值（avg）
 select avg(sscore) from score;
```

3-1-2 数学函数

3-1-2-1 取整函数: round

- 语法: round(double a)
- 返回值: BIGINT
- 说明:返回double类型的整数值部分（遵循四舍五入）
- 举例:

```
hive> select round(3.1415926);
3
```

3-1-2-2 指定精度取整函数: round

- 语法: round(double a, int d)
- 返回值: DOUBLE
- 说明:返回指定精度d的double类型
- 举例:

```
hive> select round(3.1415926,4);
3.1416
```

3-1-2-3 向下取整函数: floor

- 语法: floor(double a)
- 返回值: BIGINT
- 说明:返回等于或者小于该double变量的最大的整数
- 举例:

```
hive> select floor(3.1415926);
3
```

3-1-2-4 向上取整函数: ceil

- 语法: ceil(double a)
- 返回值: BIGINT
- 说明:返回等于或者大于该double变量的最小的整数
- 举例:

```
hive> select ceil(3.1415926) ;
4
```

3-1-2-5 取随机数函数: rand 0到1随机数

- 语法: rand(),rand(int seed)
- 返回值: double
- 说明:返回一个0到1范围内的随机数。如果指定种子seed，则会返回固定的随机数
- 举例

```
hive> select rand();
0.5577432776034763

hive> select rand();
0.6638336467363424

hive> select rand(100);
0.7220096548596434

hive> select rand(100);
0.7220096548596434
```

3-1-2-6 幂运算函数: pow

- 语法: pow(double a, double p)
- 返回值: double
- 说明:返回a的p次幂
- 举例:

```
hive> select pow(2,4) ;
16.0
```

3-1-2-7 绝对值函数: abs

- 语法: abs(double a) abs(int a)
- 返回值: double　int
- 说明:返回数值a的绝对值
- 举例:

```
hive> select abs(-3.9);
3.9
hive> select abs(10.9);
10.9
```

3-1-3 字符串函数

3-1-3-1 字符串长度函数：length

- 语法: length(string A)
- 返回值: int
- 说明：返回字符串A的长度
- 举例：

```
hive> select length('abcedfg');
7
```

3-1-3-2 字符串反转函数：reverse

- 语法: reverse(string A)
- 返回值: string
- 说明：返回字符串A的反转结果
- 举例：

```
hive> select reverse("hello");
olleh
```

3-1-3-3 字符串连接函数：concat

- 语法: concat(string A, string B...)
- 返回值: string
- 说明：返回输入字符串连接后的结果，支持任意个输入字符串
- 举例

```
hive> select concat('hello' , '#' , 'world');
hello#world
```

3-1-3-4 字符串连接函数-带分隔符：concat_ws

- 语法: concat_ws(string SEP, string A, string B...)
- 返回值: string
- 说明：返回输入字符串连接后的结果，SEP表示各个字符串间的分隔符
- 举例：

```
hive> select concat_ws(',','abc','def','gh');
abc,def,gh
```

3-1-3-5 字符串截取函数：substr,substring

- 语法: substr(string A, int start),substring(string A, int start)
- 返回值: string
- 说明：返回字符串A从start位置到结尾的字符串
- 举例：

```
hive> select substr('abcde',3);
cde
hive> select substring('abcde',3);
cde
hive>select substr('abcde',-1);
e
```

3-1-3-6 字符串截取函数：substr,substring

- 语法: substr(string A, int start, int len),substring(string A, intstart, int len)
- 返回值: string
- 说明：返回字符串A从start位置开始，长度为len的字符串
- 举例：

```
hive> select substr('abcde',3,2);
cd
hive> select substring('abcde',3,2);
cd
hive>select substring('abcde',-2,2);
de
```

3-1-3-7 字符串转大写函数：lower,ucase

- 语法: lower(string A) ucase(string A)
- 返回值: string
- 说明：返回字符串A的大写格式
- 举例：

```
hive> select lower('abSEd');
ABSED
hive> select ucase('abSEd');
ABSED
```

3-1-3-8 字符串转小写函数：lower,lcase

- 语法: lower(string A) lcase(string A)
- 返回值: string
- 说明：返回字符串A的小写格式
- 举例：

```
hive> select lower('abSEd');
absed
hive> select lcase('abSEd');
absed
```

3-1-3-9 去空格函数：trim

- 语法: trim(string A)
- 返回值: string
- 说明:去除字符串两边的空格
- 举例：

```
hive> select trim(' abc ');
abc
```

3-1-3-10 左边去空格函数：ltrim

- 语法: ltrim(string A)
- 返回值: string
- 说明：去除字符串左边的空格
- 举例：

```
hive> select ltrim('  abc ');
abc
```

3-1-3-11 右边去空格函数：rtrim

- 语法: rtrim(string A)
- 返回值: string
- 说明：去除字符串右边的空格
- 举例：

```
hive> select rtrim(' abc   ');
abc
```

3-1-3-12 正则表达式替换函数：regexp_replace

- 语法: regexp_replace(string A, string B, string C)
- A: 源字符串； B： 正则表达式； C: 替换的字符串;
- 返回值: string
- 说明：将字符串A中的符合java正则表达式B的部分替换为C。注意，在有些情况下要使用转义字符,类似oracle中的regexp_replace函数。
- 举例：

```
hive> select regexp_replace('foobar', 'oo|ar', '');
fb
```

3-1-3-13 URL解析函数：parse_url

- 语法: parse_url(string urlString, string partToExtract [, stringkeyToExtract])
- 返回值: string
- 说明：返回URL中指定的部分。partToExtract的有效值为：HOST, PATH, QUERY, REF, PROTOCOL, AUTHORITY, FILE, and USERINFO.
- 举例：

```
hive> select parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1',
'HOST');
facebook.com

hive> select parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1',
'PATH');
/path1/p.php

hive> select parse_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1',
'QUERY','k1');
v1

select parse_url ('http://facebook.com/path1/p.php?k11=v1&k2=v2#Ref1','FILE');
/path1/p.php?k11=v1&k2=v2

select parse_url ('http://facebook.com/path1/p.php?k11=v1&k2=v2#Ref1','QUERY');
k11=v1&k2=v2
```

3-1-3-14 分割字符串函数: split

- 语法: split(string str, stringpat)
- 返回值: array
- 说明:按照pat字符串分割str，会返回分割后的字符串数组
- 举例：

```
hive> select split('abtcdtef','t');
["ab","cd","ef"]
```

3-1-4 日期函数

3-1-4-1 获取当前UNIX时间戳函数:unix_timestamp

- 语法: unix_timestamp()
- 返回值: bigint
- 说明:获得当前时区的UNIX时间戳
- 举例:

```
0: jdbc:hive2://node3:10000> select unix_timestamp();
+------------+--+
|    _c0     |
+------------+--+
| 1611565954 |
+------------+--+
1 row selected (0.085 seconds)
0: jdbc:hive2://node3:10000>
```

3-1-4-2 UNIX时间戳转日期函数:from_unixtime

- 语法: from_unixtime(bigint unixtime[, string format])
- 返回值: string
- 说明:转化UNIX时间戳（从1970-01-01 00:00:00 UTC到指定时间的秒数）到当前时区的时间格式
- 举例:

```
0: jdbc:hive2://node3:10000> select from_unixtime(1611565954);
+----------------------+--+
|         _c0          |
+----------------------+--+
| 2021-01-25 17:12:34  |
+----------------------+--+
1 row selected (0.027 seconds)
0: jdbc:hive2://node3:10000>
0: jdbc:hive2://node3:10000> select from_unixtime(1611565954,'yyyy-MM-dd');
+-------------+--+
|     _c0     |
+-------------+--+
| 2021-01-25  |
+-------------+--+
1 row selected (0.038 seconds)
0: jdbc:hive2://node3:10000>
0: jdbc:hive2://node3:10000> select from_unixtime(1611565954,'yyyy-MM-dd
HH:mm');
+------------------+--+
|        _c0       |
+------------------+--+
| 2021-01-25 17:12 |
+------------------+--+
1 row selected (0.039 seconds)
0: jdbc:hive2://node3:10000>
```

3-1-4-3 日期转UNIX时间戳函数:unix_timestamp

- 语法: unix_timestamp(string date)
- 返回值: bigint
- 说明:转换格式为"yyyy-MM-ddHH:mm:ss"的日期到UNIX时间戳。如果转化失败，则返回0。
- 举例:

```
0: jdbc:hive2://node3:10000> select unix_timestamp('2021-01-25 17:12:34');
+-------------+--+
|     _c0     |
+-------------+--+
| 1611565954  |
+-------------+--+
1 row selected (0.026 seconds)
0: jdbc:hive2://node3:10000>
```

3-1-4-4 指定格式日期转时间戳函数:unix_timestamp

- 语法: unix_timestamp(string date, string pattern)
- 返回值: bigint
- 说明:转换pattern格式的日期到UNIX时间戳。如果转化失败，则返回0。
- 举例:

```
0: jdbc:hive2://node3:10000> select unix_timestamp('20210125 17:12:34','yyyyMMdd
HH:mm:ss');
+-------------+--+
|     _c0     |
+-------------+--+
| 1611565954  |
+-------------+--+
1 row selected (0.025 seconds)
0: jdbc:hive2://node3:10000>
```

3-1-4-5 日期时间转日期函数:to_date

- 语法: to_date(string timestamp)
- 注意： 参数必须是 yyyy-MM-dd xxxxxxx格式
- 返回值: string
- 说明:返回日期时间字段中的日期部分。
- 举例:

```
0: jdbc:hive2://node3:10000> select to_date('2021-01-25 17:12:34');
+-------------+--+
|     _c0     |
+-------------+--+
| 2021-01-25  |
+-------------+--+
1 row selected (0.039 seconds)
0: jdbc:hive2://node3:10000>
```

3-1-4-6 日期转年函数: year

- 语法: year(string date)
- 注意： 参数必须是 yyyy-MM-dd xxxxxxx格式
- 返回值: int
- 说明:返回日期中的年。
- 举例:

```
0: jdbc:hive2://node3:10000> select year('2021-01-25 17:13:34');
+-------+--+
|  _c0  |
+-------+--+
| 2021  |
+-------+--+
1 row selected (0.028 seconds)
0: jdbc:hive2://node3:10000>
```

3-1-4-7 日期转月函数: month

- 语法: month (string date)
- 注意： 参数必须是 yyyy-MM-dd xxxxxxx格式
- 返回值: int
- 说明:返回日期中的月份。
- 举例:

```
0: jdbc:hive2://node3:10000> select month('2021-01-25 19:34:29');
+------+--+
| _c0  |
+------+--+
| 1    |
+------+--+
```

3-1-4-8 日期转天函数: day

- 语法: day (string date)
- 注意： 参数必须是 yyyy-MM-dd xxxxxxx格式
- 返回值: int
- 说明:返回日期中的天。
- 举例:

```
0: jdbc:hive2://node3:10000> select day('2021-01-25 20:20:30' );
+------+--+
| _c0  |
+------+--+
| 25   |
+------+--+
1 row selected (0.048 seconds)
0: jdbc:hive2://node3:10000>
```

3-1-4-9 日期转小时函数：hour

- 语法: hour(string date)
- 注意： 参数必须是 yyyy-MM-dd xxxxxxx格式
- 返回值: int
- 说明:返回日期中的小时。
- 举例:

```
0: jdbc:hive2://node3:10000> select hour('2021-01-25 20:20:30' );
+------+--+
| _c0  |
+------+--+
| 20   |
+------+--+
1 row selected (0.053 seconds)
0: jdbc:hive2://node3:10000>
```

3-1-4- 10 日期转分钟函数：minute

- 语法: minute(string date)
- 注意： 参数必须是 yyyy-MM-dd xxxxxxx格式
- 返回值: int
- 说明:返回日期中的分钟。
- 举例:

```
0: jdbc:hive2://node3:10000> select minute('2021-01-25 20:21:30' );
+------+--+
| _c0  |
+------+--+
| 21   |
+------+--+
1 row selected (0.041 seconds)
0: jdbc:hive2://node3:10000>
```

3-1-4-11 日期转秒函数：second

- 语法: second(string date)
- 注意： 参数必须是 yyyy-MM-dd xxxxxxx格式
- 返回值: int
- 说明:返回日期中的秒。
- 举例:

```
0: jdbc:hive2://node3:10000> select second('2021-01-25 20:21:30' );
+------+--+
| _c0  |
+------+--+
| 30   |
+------+--+
1 row selected (0.05 seconds)
0: jdbc:hive2://node3:10000>
```

3-1-4-12 日期转周函数:weekofyear

- 语法: weekofyear (string date)
- 注意： 参数必须是 yyyy-MM-dd xxxxxxx格式
- 返回值: int
- 说明:返回日期在当前的周数。
- 举例:

```
0: jdbc:hive2://node3:10000> select weekofyear('2020-1-25');
+------+--+
| _c0  |
+------+--+
| 4    |
+------+--+
1 row selected (0.065 seconds)
0: jdbc:hive2://node3:10000>
```

3-1-4-13 日期比较函数: datediff

- 语法: datediff(string enddate, string startdate)
- 注意： 参数必须是 yyyy-MM-dd xxxxxxx格式
- 返回值: int
- 说明:返回结束日期减去开始日期的天数。
- 举例:

```
0: jdbc:hive2://node3:10000> select datediff('2021-1-25','2018-3-24');
+-------+--+
|  _c0  |
+-------+--+
| 1038  |
+-------+--+
1 row selected (0.053 seconds)
0: jdbc:hive2://node3:10000>
```

3-1-4-14 日期增加函数: date_add(后几天)

- 语法: date_add(string startdate, int days)
- 注意： 参数必须是 yyyy-MM-dd xxxxxxx格式
- 返回值: string
- 说明:返回开始日期startdate增加days天后的日期。
- 举例:

```
0: jdbc:hive2://node3:10000> select date_add('2021-01-25',10);
+-------------+--+
|     _c0     |
+-------------+--+
| 2021-02-04  |
+-------------+--+
```

3-1-4-15 日期减少函数: date_sub (前几天)

- 语法: date_sub(string startdate, int days)
- 注意： 参数必须是 yyyy-MM-dd xxxxxxx格式
- 返回值: string
- 说明:返回开始日期startdate减少days天后的日期。
- 举例:

```
0: jdbc:hive2://node3:10000> select date_sub('2021-01-25',10);
+-------------+--+
|     _c0     |
+-------------+--+
| 2021-01-15  |
+-------------+--+
1 row selected (0.044 seconds)
0: jdbc:hive2://node3:10000>
```

3-1-4-16 date_format函数 (长度固定为8)

- 语法: date_format(string date, string dataformat)
- 注意： 参数必须是 yyyy-MM-dd xxxxxxx格式
- 返回值: string
- 说明:返回开始日期startdate减少days天后的日期。
- 举例:

```
0: jdbc:hive2://node3:10000> select date_format('2021-1-5','yyyy-MM-dd');
+-------------+--+
|     _c0     |
+-------------+--+
| 2021-01-05  |
+-------------+--+
1 row selected (0.057 seconds)
0: jdbc:hive2://node3:10000>
```

3-1-5 条件函数

3-1-5-1 if函数: if (类似于三目运算符)

- 语法: if(boolean testCondition, T valueTrue, T valueFalseOrNull)
- 返回值: T
- 说明: 当条件testCondition为TRUE时，返回valueTrue；否则返回valueFalseOrNull
- 举例:

```
hive> select if(1=2,100,200) ;
200
hive> select if(1=1,100,200) ;
100
```

3-1-5-2 条件判断函数：CASE xxx

- 语法: CASE a WHEN b THEN c [WHEN d THEN e]* [ELSE f] END
- 返回值: T
- 说明：如果a等于b，那么返回c；如果a等于d，那么返回e；否则返回f
- 举例:

```
select
    case 100
        when 10 then 'tom'
        when 100 then 'mary'
        else 'tony'
    end;

mary
```

3-1-5-3 条件判断函数：CASE

- 语法: CASE WHEN a THEN b [WHEN c THEN d]* [ELSE e] END
- 返回值: T
- 说明：如果a为TRUE,则返回b；如果c为TRUE，则返回d；否则返回e
- 举例：

```
select
    case
        when 1>2 then 'tom'
        when 1<2 then 'mary'
        else 'tony'
    end;

select
    sid,
    case
        when sscore >=60 then 'A'
        when sscore < 60 then 'B'
        else 'O'
    end
from
    score
where
    month = '202004';
```

3-1-6 转换函数

3-1-6-1 cast 类似于java中的强转

- 类似于java中的强转
- 公式：
  - cast(表达式 as 数据类型)
  - cast函数，可以将"20190607"这样类型的时间数据转化成int类型数据。

```
0: jdbc:hive2://node3:10000> select cast(12.34 as int);
+------+--+
| _c0  |
+------+--+
| 12   |
+------+--+

0: jdbc:hive2://node3:10000> select cast('2021-1-25' as date);
+-------------+--+
|     _c0     |
+-------------+--+
| 2021-01-25  |
+-------------+--+
```

3-1-7 Hive的（行转列）

3-1-7-1 介绍

- 行转列是指多行数据转换为一个列的字段。
- 行转列必须有group by
- Hive行转列用到的函数：
    - concat(str1,str2,...) --字段或字符串拼接
    - concat_ws(sep, str1,str2) --以分隔符拼接每个字符串
    - collect_set(col) --将某字段的值进行去重汇总，产生array类型字段

3-1-7-2 测试数据

```
字段: depno   depname
20   SMITH
30  ALLEN
30  WARD
20   JONES
30  MARTIN
30  BLAKE
10  CLARK
20   SCOTT
10  KING
30   TURNER
20   ADAMS
30   JAMES
20  FORD
```

3-1-7-3 操作步骤

- 建表

```
create table emp(
depno int,
depname string
) row format delimited fields terminated by '\t';
```

- 插入数据

```
load data local inpath "/export/data/hivedatas/emp.txt" into table emp;
```

- 转换

```
select depno , concat_ws('|',collect_set(depname)) depnames from emp group by
depno;
+--------+---------------------------------------+--+
| depno  |               depnames                |
+--------+---------------------------------------+--+
| 10     | CLARK|KING|MILLER                     |
| 20     | SMITH|JONES|SCOTT|ADAMS|FORD          |
| 30     | ALLEN|WARD|MARTIN|BLAKE|TURNER|JAMES  |
+--------+---------------------------------------+--+
3 rows selected (9.555 seconds)
```

3-2 Hive的表生成函数

3-2-1 explode函数

- explode(col)：将hive一列中复杂的array或者map结构拆分成多行。
- explode(ARRAY) 数组的每个元素生成一行
- explode(MAP) map中每个key-value对，生成一行，key为一列，value为一列

3-2-1-1 案例操作

- 数据

```
10   CLARK|KING|MILLER
20   SMITH|JONES|SCOTT|ADAMS|FORD
30  ALLEN|WARD|MARTIN|BLAKE|TURNER|JAMES
```

- 创建表

```
create table emp2(
    depno int,
    depnames array<string>
) row format delimited fields terminated by '\t'
collection items terminated by '|';
```

- 导入数据

```
load data local inpath '/export/data/hivedatas/emp2.txt' into table emp2;
```

- 使用explode查询 查询结果就是一张表；

```
select explode(depnames) from emp2;
+---------+--+
|   col   |
+---------+--+
| CLARK   |
| KING    |
| MILLER  |
```

```
| SMITH    |
| JONES    |
| SCOTT    |
| ADAMS    |
| FORD     |
| ALLEN    |
| WARD     |
| MARTIN   |
| BLAKE    |
| TURNER   |
| JAMES    |
+---------+--+
```

3-2-2（列转行） LATERAL VIEW侧视图

- 用法：LATERAL VIEW udtf(col) 临时表名 AS 列名
- 解释：用于和split, explode等UDTF一起使用，它能够将一列数据拆成多行数据，在此基础上可以对拆分后的数据进行聚合。
- 相当于两张表的笛卡尔积

```
0: jdbc:hive2://node3:10000> select * from emp2 lateral view explode(depnames)
tmp_tb as depname;
+------------+-----------------------------------------------------+------------
-----+--+
| emp2.empno  |                    emp2.depnames                    |
tmp_tb.depname  |
+------------+-----------------------------------------------------+------------
-----+--+
| 10          | ["CLARK","KING","MILLER"]                           | CLARK
      |
| 10          | ["CLARK","KING","MILLER"]                           | KING
     |
| 10          | ["CLARK","KING","MILLER"]                           | MILLER
     |
| 20          | ["SMITH","JONES","SCOTT","ADAMS","FORD"]            | SMITH
      |
| 20          | ["SMITH","JONES","SCOTT","ADAMS","FORD"]            | JONES
      |
| 20          | ["SMITH","JONES","SCOTT","ADAMS","FORD"]            | SCOTT
      |
| 20          | ["SMITH","JONES","SCOTT","ADAMS","FORD"]            | ADAMS
      |
| 20          | ["SMITH","JONES","SCOTT","ADAMS","FORD"]            | FORD
     |
| 30          | ["ALLEN","WARD","MARTIN","BLAKE","TURNER","JAMES"]  | ALLEN
      |
| 30          | ["ALLEN","WARD","MARTIN","BLAKE","TURNER","JAMES"]  | WARD
     |
| 30          | ["ALLEN","WARD","MARTIN","BLAKE","TURNER","JAMES"]  | MARTIN
     |
```

```
| 30          | ["ALLEN","WARD","MARTIN","BLAKE","TURNER","JAMES"]  | BLAKE
      |
| 30          | ["ALLEN","WARD","MARTIN","BLAKE","TURNER","JAMES"]  | TURNER
     |
| 30          | ["ALLEN","WARD","MARTIN","BLAKE","TURNER","JAMES"]  | JAMES
      |
+-------------+-----------------------------------------------------+-----------
-----+--+
14 rows selected (0.057 seconds)
0: jdbc:hive2://node3:10000>
```

- 抽取需要的字段 empno 和depname

```
0: jdbc:hive2://node3:10000> select empno , depname from emp2 lateral view
explode(depnames) tmp_tb as depname;
+--------+----------+--+
| empno  | depname  |
+--------+----------+--+
| 10     | CLARK    |
| 10     | KING     |
| 10     | MILLER   |
| 20     | SMITH    |
| 20     | JONES    |
| 20     | SCOTT    |
| 20     | ADAMS    |
| 20     | FORD     |
| 30     | ALLEN    |
| 30     | WARD     |
| 30     | MARTIN   |
| 30     | BLAKE    |
| 30     | TURNER   |
| 30     | JAMES    |
+--------+----------+--+
14 rows selected (0.053 seconds)
0: jdbc:hive2://node3:10000>
```

3-2-3 Reflect函数

- **reflect函数可以支持在sql中调用java中的自带函数**

3-2-3-1 使用java.lang.Math当中的Max求两列中最大值

```
--创建hive表
create table test_reflect(col1 int,col2 int) row format delimited fields
terminated by ',';

--准备数据 test_udf.txt
```

```
1,2
4,3
6,4
7,5
5,6

--加载数据

load data local inpath '/export/data/hivedatas/test_udf.txt'  into table
test_reflect;

--使用java.lang.Math当中的Max求两列当中的最大值
select reflect("java.lang.Math","max",col1,col2) from test_reflect;
```

- 求两个数的最大值;

```
select reflect('java.lang.Math','max',2,4);

| 4    |
```

- 获取uuid

```
0: jdbc:hive2://node3:10000> select  reflect('java.util.UUID','randomUUID');
+--------------------------------------+--+
|                 _c0                  |
+--------------------------------------+--+
| 262e0763-c75e-44e8-ab28-4694663dc945  |
+--------------------------------------+--+
```

- 获取uuid 去掉'-'

```
0: jdbc:hive2://node3:10000> select
regexp_replace(reflect('java.util.UUID','randomUUID'),'-','');
+----------------------------------+--+
|                _c0               |
+----------------------------------+--+
| 122701f5d3994838bd02f49c26eb10a8  |
+----------------------------------+--+
1 row selected (0.051 seconds)
0: jdbc:hive2://node3:10000>
```

3-3 Hive的开窗函数(函数名（） over(partition by xxx order by yyy))

3-3-1 窗口函数(一) ROW_NUMBER,RANK,DENSE_RANK

3-3-1-1 数据准备

```
cookie1,2018-04-10,1
cookie1,2018-04-11,5
cookie1,2018-04-12,7
cookie1,2018-04-13,3
cookie1,2018-04-14,2
cookie1,2018-04-15,4
cookie1,2018-04-16,4
cookie2,2018-04-10,2
cookie2,2018-04-11,3
cookie2,2018-04-12,5
cookie2,2018-04-13,6
cookie2,2018-04-14,3
cookie2,2018-04-15,9
cookie2,2018-04-16,7

CREATE TABLE itcast_t1 (
cookieid string,
createtime string,   --day
pv INT
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

-- 加载数据:
load data local inpath '/export/data/hivedatas/itcast_t2.txt' into table
itcast_t1;
```

3-3-1-2 Row_number

- 语法

```
row_number() over(partition by 分组列 order by  排序列 desc<降序>) rn from 表名;
```

- 案例

```
select *,row_number() over(partition by cookieid order by pv desc) rn  from
itcast_t2;
+--------------------+----------------------+---------------+-----+--+
| itcast_t2.cookieid  | itcast_t2.createtime  | itcast_t2.pv  | rn  |
+--------------------+----------------------+---------------+-----+--+
| cookie1             | 2018-04-12           | 7             | 1   |
| cookie1             | 2018-04-11           | 5             | 2   |
| cookie1             | 2018-04-16           | 4             | 3   |
| cookie1             | 2018-04-15           | 4             | 4   |
| cookie1             | 2018-04-13           | 3             | 5   |
| cookie1             | 2018-04-14           | 2             | 6   |
| cookie1             | 2018-04-10           | 1             | 7   |
| cookie2             | 2018-04-15           | 9             | 1   |
| cookie2             | 2018-04-16           | 7             | 2   |
| cookie2             | 2018-04-13           | 6             | 3   |
| cookie2             | 2018-04-12           | 5             | 4   |
| cookie2             | 2018-04-11           | 3             | 5   |
| cookie2             | 2018-04-14           | 3             | 6   |
| cookie2             | 2018-04-10           | 2             | 7   |
```

```
+--------------------+----------------------+---------------+-----+--+
14 rows selected (5.822 seconds)
0: jdbc:hive2://node3:10000>
```

- 查询每个用户的前三名

```
select
    *
from
    (select
        *,
        row_number() over(partition by cookieid order by pv desc) rn
     from
        itcast_t2)  t
where
    t.rn >= 3;

+-------------+---------------+-------+-------+--+
| t.cookieid  | t.createtime  | t.pv  | t.rn  |
+-------------+---------------+-------+-------+--+
| cookie1     | 2018-04-16    | 4     | 3     |
| cookie1     | 2018-04-15    | 4     | 4     |
| cookie1     | 2018-04-13    | 3     | 5     |
| cookie1     | 2018-04-14    | 2     | 6     |
| cookie1     | 2018-04-10    | 1     | 7     |
| cookie2     | 2018-04-13    | 6     | 3     |
| cookie2     | 2018-04-12    | 5     | 4     |
| cookie2     | 2018-04-11    | 3     | 5     |
| cookie2     | 2018-04-14    | 3     | 6     |
| cookie2     | 2018-04-10    | 2     | 7     |
+-------------+---------------+-------+-------+--+
10 rows selected (8.033 seconds)
0: jdbc:hive2://node3:10000>
```

3-3-1-3 RANK 和 DENSE_RANK

- row_number() 依次排名， 不管值是否相等； 1-2-3-4-5-6
- RANK() 生成数据项在分组中的排名，排名相等会在名次中留下空位； 1-2-2-4-5-6 （没有3）
- DENSE_RANK() 生成数据项在分组中的排名，排名相等会在名次中不会留下空位； 1-2-2-3-4-5

```
select
    * ,
    row_number() over(partition by cookieid order by pv desc) rn1 ,
    rank() over(partition by cookieid order by pv desc) rn2,
    dense_rank() over(partition by cookieid order by pv desc) r3
from
    itcast_t2;
```

```
+---------------------+----------------------+---------------+------+------+----
-+--+
| itcast_t2.cookieid  | itcast_t2.createtime  | itcast_t2.pv  | rn1  | rn2  | r3
|
+---------------------+----------------------+---------------+------+------+----
-+--+
| cookie1             | 2018-04-12            | 7             | 1    | 1    | 1
  |
| cookie1             | 2018-04-11            | 5             | 2    | 2    | 2
  |
| cookie1             | 2018-04-16            | 4             | 3    | 3    | 3
  |
| cookie1             | 2018-04-15            | 4             | 4    | 3    | 3
  |
| cookie1             | 2018-04-13            | 3             | 5    | 5    | 4
  |
| cookie1             | 2018-04-14            | 2             | 6    | 6    | 5
  |
| cookie1             | 2018-04-10            | 1             | 7    | 7    | 6
  |
| cookie2             | 2018-04-15            | 9             | 1    | 1    | 1
  |
| cookie2             | 2018-04-16            | 7             | 2    | 2    | 2
  |
| cookie2             | 2018-04-13            | 6             | 3    | 3    | 3
  |
| cookie2             | 2018-04-12            | 5             | 4    | 4    | 4
  |
| cookie2             | 2018-04-11            | 3             | 5    | 5    | 5
  |
| cookie2             | 2018-04-14            | 3             | 6    | 5    | 5
  |
| cookie2             | 2018-04-10            | 2             | 7    | 7    | 6
  |
+---------------------+----------------------+---------------+------+------+----
-+--+
14 rows selected (8.041 seconds)
0: jdbc:hive2://node3:10000>
```

3-3-1-4 总结

- row_number: 1-2-3-4-5-6
- rank: 1-2-2-4-4-6
- dense_rank: 1-2-2-3-3-4

3-3-2 Hive分析窗口函数(2) SUM,AVG,MIN,MAX

3-3-2-1 数据准备

```
--建表语句:
create table itcast_t2(
cookieid string,
createtime string,   --day
pv int
) row format delimited
fields terminated by ',';

--加载数据:
load data local inpath '/root/hivedata/itcast_t2.txt' into table itcast_t2;

--开启智能本地模式
SET hive.exec.mode.local.auto=true;
```

3-3-2-2 SUM（结果和ORDER BY相关,默认为升序)

- 累计叠加

```
select *, sum(pv) over(partition by cookieid order by createtime ) sumpv from
itcast_t2;
+----------------------+-----------------------+---------------+-----+--+
| itcast_t2.cookieid  | itcast_t2.createtime  | itcast_t2.pv  | sumpv  |
+----------------------+-----------------------+---------------+-----+--+
| cookie1             | 2018-04-10            | 1             | 1   |
| cookie1             | 2018-04-11            | 5             | 6   |
| cookie1             | 2018-04-12            | 7             | 13  |
| cookie1             | 2018-04-13            | 3             | 16  |
| cookie1             | 2018-04-14            | 2             | 18  |
| cookie1             | 2018-04-15            | 4             | 22  |
| cookie1             | 2018-04-16            | 4             | 26  |
| cookie2             | 2018-04-10            | 2             | 2   |
| cookie2             | 2018-04-11            | 3             | 5   |
| cookie2             | 2018-04-12            | 5             | 10  |
| cookie2             | 2018-04-13            | 6             | 16  |
| cookie2             | 2018-04-14            | 3             | 19  |
| cookie2             | 2018-04-15            | 9             | 28  |
| cookie2             | 2018-04-16            | 7             | 35  |
+----------------------+-----------------------+---------------+-----+--+
14 rows selected (8.077 seconds)
0: jdbc:hive2://node3:10000>
```

- 累计从开始到当前行 rows between unbounded preceding and current row

```
select
    * ,
    sum(pv)
    over(
         partition by cookieid
         order by createtime
         rows between unbounded preceding and current row
    ) sumpv
from
    itcast_t2;
```

```
+---------------------+----------------------+---------------+--------+--+
| itcast_t2.cookieid  | itcast_t2.createtime  | itcast_t2.pv  | sumpv  |
+---------------------+----------------------+---------------+--------+--+
| cookie1             | 2018-04-10            | 1             | 1      |
| cookie1             | 2018-04-11            | 5             | 6      |
| cookie1             | 2018-04-12            | 7             | 13     |
| cookie1             | 2018-04-13            | 3             | 16     |
| cookie1             | 2018-04-14            | 2             | 18     |
| cookie1             | 2018-04-15            | 4             | 22     |
| cookie1             | 2018-04-16            | 4             | 26     |
| cookie2             | 2018-04-10            | 2             | 2      |
| cookie2             | 2018-04-11            | 3             | 5      |
| cookie2             | 2018-04-12            | 5             | 10     |
| cookie2             | 2018-04-13            | 6             | 16     |
| cookie2             | 2018-04-14            | 3             | 19     |
| cookie2             | 2018-04-15            | 9             | 28     |
| cookie2             | 2018-04-16            | 7             | 35     |
+---------------------+----------------------+---------------+--------+--+
14 rows selected (8.077 seconds)
```

- 如果没有order by排序语句 默认把分组内的所有数据进行sum操作

```
select
    * ,
    sum(pv)
    over(partition by cookieid) pvs
from
    itcast_t2;

+---------------------+----------------------+---------------+------+--+
| itcast_t2.cookieid  | itcast_t2.createtime  | itcast_t2.pv  | pvs  |
+---------------------+----------------------+---------------+------+--+
| cookie1             | 2018-04-10            | 1             | 26   |
| cookie1             | 2018-04-16            | 4             | 26   |
| cookie1             | 2018-04-15            | 4             | 26   |
| cookie1             | 2018-04-14            | 2             | 26   |
| cookie1             | 2018-04-13            | 3             | 26   |
| cookie1             | 2018-04-12            | 7             | 26   |
| cookie1             | 2018-04-11            | 5             | 26   |
| cookie2             | 2018-04-16            | 7             | 35   |
| cookie2             | 2018-04-15            | 9             | 35   |
| cookie2             | 2018-04-14            | 3             | 35   |
| cookie2             | 2018-04-13            | 6             | 35   |
| cookie2             | 2018-04-12            | 5             | 35   |
| cookie2             | 2018-04-11            | 3             | 35   |
| cookie2             | 2018-04-10            | 2             | 35   |
+---------------------+----------------------+---------------+------+--+
14 rows selected (5.932 seconds)
```

- 钱三行到当前行 累加 rows between 3 preceding and current row

```
select
    * ,
    sum(pv)
```

```
    over( partition by cookieid
         order by createtime
         rows between 3 preceding and current row ) pvs
from
    itcast_t2;
+---------------------+----------------------+---------------+------+--+
| itcast_t2.cookieid  | itcast_t2.createtime  | itcast_t2.pv  | pvs  |
+---------------------+----------------------+---------------+------+--+
| cookie1             | 2018-04-10           | 1             | 1    |
| cookie1             | 2018-04-11           | 5             | 6    |
| cookie1             | 2018-04-12           | 7             | 13   |
| cookie1             | 2018-04-13           | 3             | 16   |
| cookie1             | 2018-04-14           | 2             | 17   |
| cookie1             | 2018-04-15           | 4             | 16   |
| cookie1             | 2018-04-16           | 4             | 13   |
| cookie2             | 2018-04-10           | 2             | 2    |
| cookie2             | 2018-04-11           | 3             | 5    |
| cookie2             | 2018-04-12           | 5             | 10   |
| cookie2             | 2018-04-13           | 6             | 16   |
| cookie2             | 2018-04-14           | 3             | 17   |
| cookie2             | 2018-04-15           | 9             | 23   |
| cookie2             | 2018-04-16           | 7             | 25   |
+---------------------+----------------------+---------------+------+--+
14 rows selected (5.738 seconds)
```

- 从前三行到下一行（一共5行）累加 rows between 3 preceding and 1 following

```
select
    * ,
    sum(pv)
    over(partition by cookieid
         order by createtime
         rows between 3 preceding and 1 following )as pvs
from
    itcast_t2;

+---------------------+----------------------+---------------+------+--+
| itcast_t2.cookieid  | itcast_t2.createtime  | itcast_t2.pv  | pvs  |
+---------------------+----------------------+---------------+------+--+
| cookie1             | 2018-04-10           | 1             | 6    |
| cookie1             | 2018-04-11           | 5             | 13   |
| cookie1             | 2018-04-12           | 7             | 16   |
| cookie1             | 2018-04-13           | 3             | 18   |
| cookie1             | 2018-04-14           | 2             | 21   |
| cookie1             | 2018-04-15           | 4             | 20   |
| cookie1             | 2018-04-16           | 4             | 13   |
| cookie2             | 2018-04-10           | 2             | 5    |
| cookie2             | 2018-04-11           | 3             | 10   |
| cookie2             | 2018-04-12           | 5             | 16   |
| cookie2             | 2018-04-13           | 6             | 19   |
| cookie2             | 2018-04-14           | 3             | 26   |
| cookie2             | 2018-04-15           | 9             | 30   |
| cookie2             | 2018-04-16           | 7             | 25   |
+---------------------+----------------------+---------------+------+--+
14 rows selected (8.598 seconds)
```

- 从当前行到最后累加 rows between current row and unbounded following

```
select
    * ,
    sum(pv)
    over(partition by cookieid
         order by createtime
         rows between current row and unbounded following)as pvs
from
    itcast_t2;

+---------------------+-----------------------+---------------+------+--+
| itcast_t2.cookieid  | itcast_t2.createtime  | itcast_t2.pv  | pvs  |
+---------------------+-----------------------+---------------+------+--+
| cookie1             | 2018-04-10            | 1             | 26   |
| cookie1             | 2018-04-11            | 5             | 25   |
| cookie1             | 2018-04-12            | 7             | 20   |
| cookie1             | 2018-04-13            | 3             | 13   |
| cookie1             | 2018-04-14            | 2             | 10   |
| cookie1             | 2018-04-15            | 4             | 8    |
| cookie1             | 2018-04-16            | 4             | 4    |
| cookie2             | 2018-04-10            | 2             | 35   |
| cookie2             | 2018-04-11            | 3             | 33   |
| cookie2             | 2018-04-12            | 5             | 30   |
| cookie2             | 2018-04-13            | 6             | 25   |
| cookie2             | 2018-04-14            | 3             | 19   |
| cookie2             | 2018-04-15            | 9             | 16   |
| cookie2             | 2018-04-16            | 7             | 7    |
+---------------------+-----------------------+---------------+------+--+
14 rows selected (6.774 seconds)
```

- 总结

```
/*
- 如果不指定rows between,默认为从起点到当前行;
- 如果不指定order by，则将分组内所有值累加;
- 关键是理解rows between   and  含义,也叫做window子句：
  - preceding：往前
  - following：往后
  - current row：当前行
  - unbounded：起点
  - unbounded preceding 表示从前面的起点
  - unbounded following：表示到后面的终点
 */
```

3-3-2-3 AVG，MIN，MAX

AVG,MIN,MAX和SUM用法一样

```
select cookieid,createtime,pv,
avg(pv) over(partition by cookieid order by createtime rows between unbounded
preceding and current row) as pv2
from itcast_t2;

select cookieid,createtime,pv,
max(pv) over(partition by cookieid order by createtime rows between unbounded
preceding and current row) as pv2
from itcast_t2;

select cookieid,createtime,pv,
min(pv) over(partition by cookieid order by createtime rows between unbounded
preceding and current row) as pv2
from itcast_t2;
```

3-4 Hive自定义函数

3-4-1 概述

- 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。
- 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）。
- 根据用户自定义函数类别分为以下三种：
    1. UDF（User-Defined-Function）
        - 一进一出
        - 类似于:lower/lower/reverse
    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 explode()

3-4-2 自定义UDF 一进一出

编程步骤：

（1）继承org.apache.hadoop.hive.ql.exec.UDF

（2）需要实现 **evaluate** 函数；**evaluate** 函数支持重载；

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；

3-4-2-1 代码编写

- 第一步：创建maven java 工程，导入jar包

```
<dependencies>
    <dependency>
        <groupId>org.apache.hive</groupId>
        <artifactId>hive-exec</artifactId>
        <version>2.1.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-common</artifactId>
        <version>2.7.5</version>
    </dependency>
</dependencies>
```

- 第二步：开发java类继承UDF，并重载evaluate 方法

```
public class Myudf  extends UDF {

    /**
     * udf 函数
     *  一进一出
     * @return
     */
    public Text evaluate( final Text line){
        // 需求：  首字母大写，其它小写；Smith

        if (null == line || "".equals(line.toString()) ) {
            return null;
        }

        String strLine = line.toString();

        String resultLine = strLine.substring(0,1).toUpperCase() +
                strLine.substring(1).toLowerCase();

        return new Text(resultLine);
    }

}
```

3-4-2-2 函数使用方式1-临时函数 temporary

- 将我们的项目打包，并上传到hive的lib目录下
- 添加我们的jar包
    - 将jar包上传到 /export/server/hive-2.1.0/lib目录，并重命名我们的jar包名称

```
cd /export/server/hive-2.1.0/lib
mv day14_hive_udf-1.0-SNAPSHOT.jar myupper.jar
```

- hive的客户端添加我们的jar包

```
add jar /export/server/hive-2.1.0/lib/myupper.jar
```

- 设置函数与我们的自定义函数关联-临时函数 (temporary)

```
create temporary function my_lower as 'com.fiberhome.udf.Myudf';
```

- 使用自定义函数

```
select myupper('ASDFakdfjSDKFJa');
+------------------+--+
|       _c0        |
+------------------+--+
| Asdfakdfjsdkfja  |
+------------------+--+
```

3-4-2-3 函数使用方式2-永久函数 (jar 包必须放在hdfs上)

- 把自定义函数的jar上传到hdfs中

```
hadoop fs -mkdir /hive/funcs
hadoop fs -put myupper.jar /hive/funcs
```

- 创建永久函数

```
create function myupper as 'com.fiberhome.udf.Myudf' using jar
'hdfs://node1:8020/hive/funcs/myupper.jar';
```

- 验证

```
select myupper('ASDFakdfjSDKFJa');
+------------------+--+
|       _c0        |
+------------------+--+
| Asdfakdfjsdkfja  |
+------------------+--+
```

3-4-3 自定义UDTF 一进多出

3-4-3-1 需求

3-4-3-2 代码实现

3-4-3-3 添加我们的jar包

3-4-3-4 创建临时函数与开发后的udtf代码关联

3-4-3-5 使用自定义udtf函数

3-4-4 自定义UDAF 多进一出

略.....

4- hive的数据压缩

4-1 MR支持的压缩编码

4-2 压缩配置参数

4-3 开启Map输出阶段压缩

4-4 开启Reduce输出阶段压缩

5- hive的数据存储格式

5-1 列式存储和行式存储

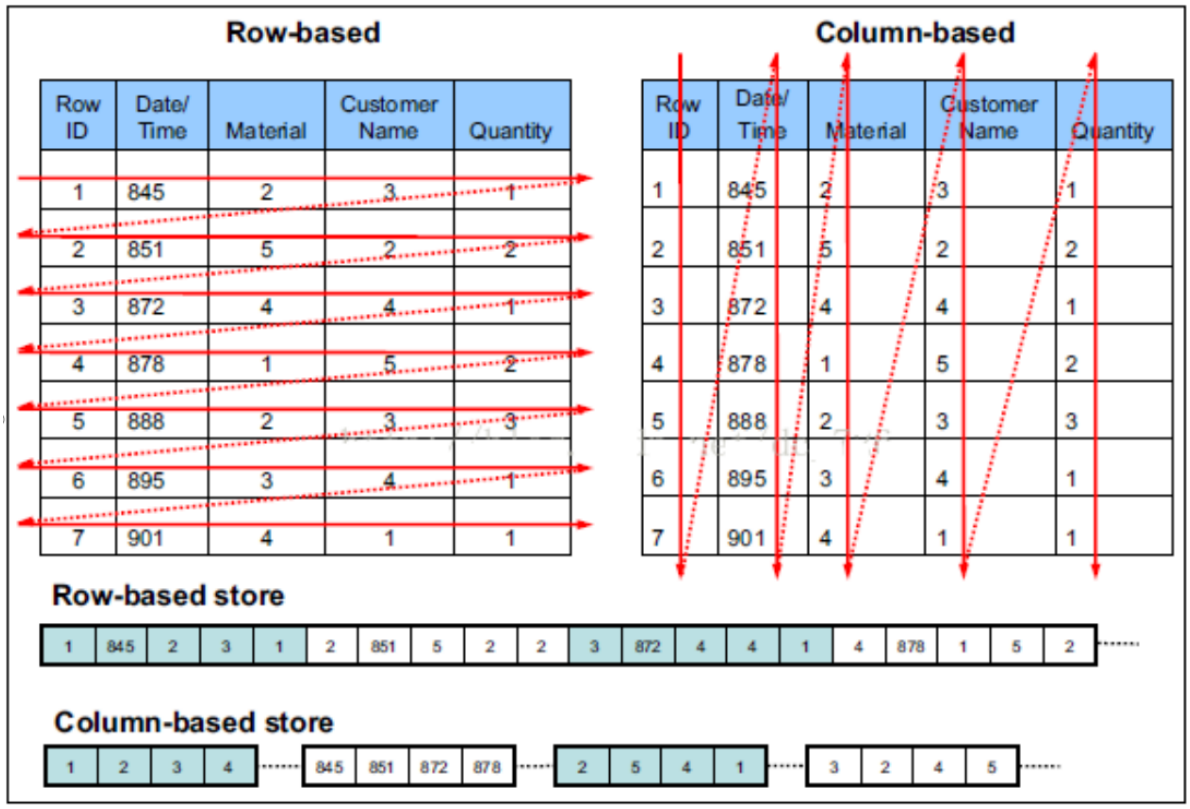


Figure 1-4 Row-based and column-based storage models

5-2 主流文件存储格式对比实验

5-3 存储和压缩结合

6- hive调优

6-1 本地模式

6-1-1 空key处理

6-1-1-1 空key处理

6-1-1-2 空key转换

6-2 SQL优化

6-2-1 Group By

6-2-2 Count(distinct)

6-2-3 笛卡尔积

6-3 并行执行

6-4 严格模式

6-5 存储方式和压缩方式

7- Hive综合案例

7-1 需求描述

7-2 项目表的字段

7-3 进行数据的清洗工作

7-4 准备工作

7-4-1 创建 hive 表

7-4-2 导入ETL后的数据

7-4-3 向ORC表插入数据

7-5 业务分析

7-5-1 统计视频观看数 top10

7-5-2 统计视频类别热度Top10

7-5-3 统计出视频观看数最高的20个视频的所属类别以及类别包含Top20视频的个数

7-5-4 统计每个类别中的视频热度Top10，以Music为例

7-5-5 统计每个类别中视频流量Top10

7-5-6 统计上传视频最多的用户Top10以及他们上传的观看次数在前10的视频