

Hive

Hive

课程目标

- Hive 引言

 - 简介

- Hive 的架构

 - 1. 简介

 - 2. 图

- Hive的安装

 - 1. 安装mysql数据库

 - 2. 安装Hadoop

 - 3. 安装hive

 - 1 上传hive安装包到linux中

 - 2 解压缩hive

 - 3 配置环境变量

 - 4 加载系统配置生效

 - 5 配置hive

 - 4 启动

 - 1. 启动 hadoop

 - 2. 本地启动hive

 - 3. 启动Hive的两种方式

 - 3.hive的客户端和服务端

 - beeline客户端

 - DBeaver客户端(图形化界面)

 - JDBC

 - 4. 数据类型

- Hive数据导入

 - 1.自定义分隔符[重点]

 - 2.JSON分割符

 - 3.正则分隔符

- HQL高级

 - HQL高级

 - 全排序和局部排序

- Hive中表分类

 - 4.1 管理表

 - 4.2 外部表

 - 4.3 分区表

 - 4.3.1 创建分区表

- Hive自定义函数

 - 内置函数

 - 用户自定义函数UDF

 - 导入奇葩的依赖方法-pentahu

- 案例

 - 列自增长(不确定性函数)

 - 用户自定义函数UDTF

 - 表数据转存导入操作

课程目标

Hive 引言

简介

hive是facebook开源，并捐献给了apache组织，作为apache组织的顶级项目(hive.apache.org)。hive是一个基于大数据技术的数据仓库(DataWareHouse)技术，主要是通过将用户书写的SQL语句翻译成MapReduce代码，然后发布任务给MR框架执行，完成SQL到MapReduce的转换。可以将结构化的数据文件映射为一张数据库表，并提供类SQL查询功能。

总结

- Hive是一个数据仓库
- Hive构建在HDFS上，可以存储海量数据。
- Hive允许程序员使用SQL命令来完成数据的分布式计算，计算构建在yarn之上。(Hive会将SQL转化为MR操作)

优点：

简化程序员的开发难度，写SQL即可，避免了去写mapreduce,减少开发人员的学习成本

缺点：

延迟较高(MapReduce本身延迟，Hive SQL向MapReduce转化优化提交)，适合做大数据的离线处理(TB PB级别的数据，统计结果延迟1天产出)

Hive不适合场景：

1：小数据量

2：实时计算

- 数据库 DataBase
 - 数据量级小，数据价值高
- 数据仓库 DataWareHouse
 - 数据体量大，数据价值低

Hive 的架构

1. 简介

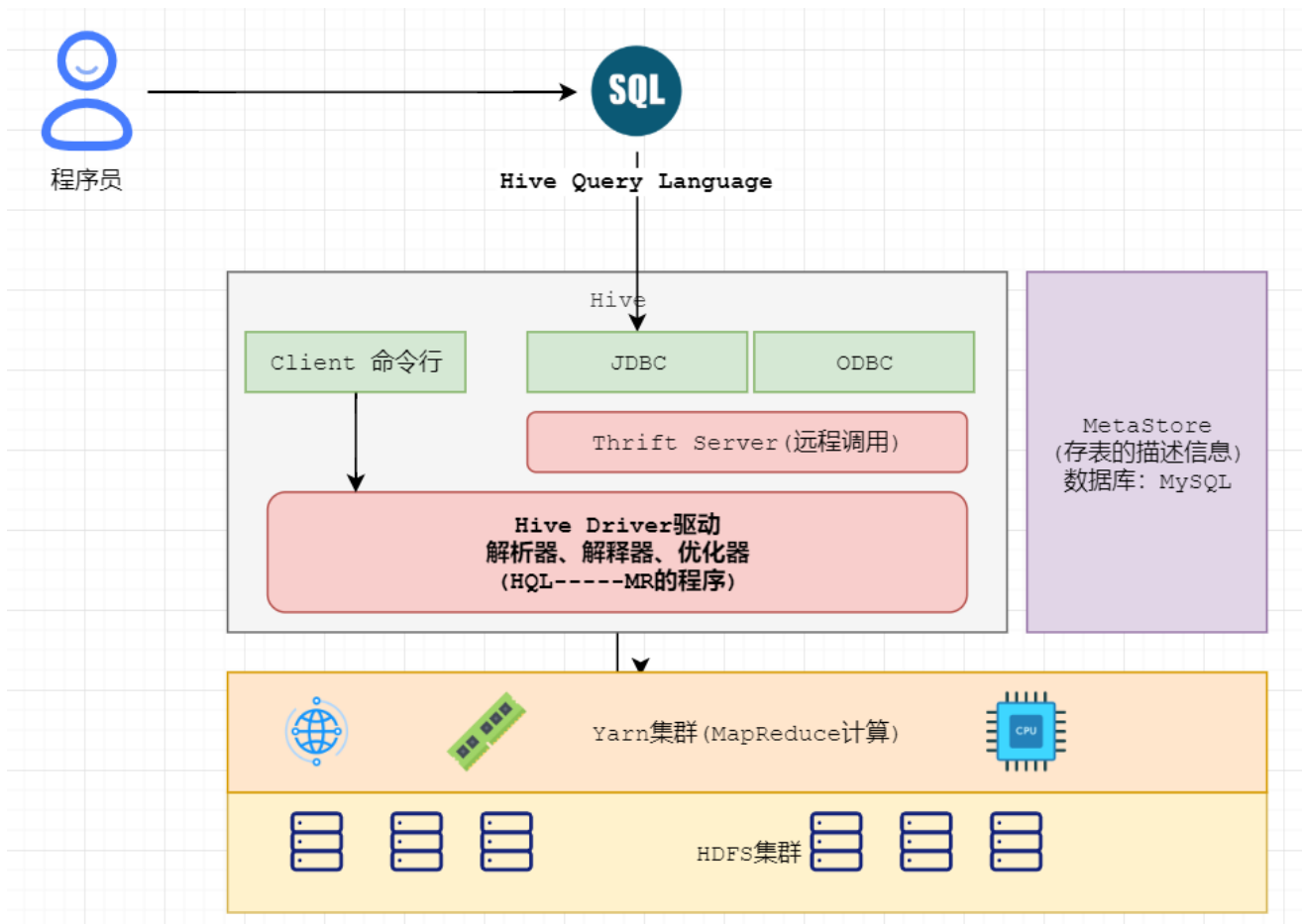
HDFS：用来存储hive仓库的数据文件

yarn：用来完成hive的HQL转化的MR程序的执行

MetaStore：保存管理hive维护的元数据

Hive：用来通过HQL的执行，转化为MapReduce程序的执行，从而对HDFS集群中的数据文件进行统计。

2. 图



Hive的安装

步骤

1. HDFS(Hadoop2.9.2)
2. Yarn(Hadoop2.9.2)
3. MySQL(5.6)
4. Hive(1.2.1)

虚拟机内存设置至少1G

1. 安装mysql数据库

参考MySQL安装文档

2. 安装Hadoop

配置hdfs和yarn的配置信息

```
[root@hive40 ~]# jps
1651 NameNode
2356 NodeManager
2533 Jps
1815 DataNode
2027 SecondaryNameNode
2237 ResourceManager
```

3. 安装hive

1 上传hive安装包到linux中

2 解压缩hive

```
[root@hadoop ~]# tar -zxvf apache-hive-1.2.1-bin.tar.gz -C /opt/installs
[root@hadoop ~]# mv apache-hive-1.2.1-bin hive1.2.1
```

3 配置环境变量

```
export HIVE_HOME=/opt/installs/hive1.2.1
export PATH=$PATH:$HIVE_HOME/bin
```

4 加载系统配置生效

```
[root@hadoop ~]# source /etc/profile
```

5 配置hive

hive-env.sh

拷贝一个hive-env.sh: [root@hadoop10 conf]# cp hive-env.sh.template hive-env.sh

```
# 配置hadoop目录
HADOOP_HOME=/opt/installs/hadoop2.9.2/
# 指定hive的配置文件目录
export HIVE_CONF_DIR=/opt/installs/hive1.2.1/conf/
```

hive-site.xml

拷贝得到hive-site.xml: [root@hadoop10 conf]# cp hive-default.xml.template hive-site.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <!--hive的元数据保存在mysql中，需要连接mysql，这里配置访问mysql的信息-->
  <!--url: 这里必须用ip-->
  <property>
    <name>javax.jdo.option.ConnectionURL</name>
    <value>jdbc:mysql://hadoop10:3306/hive</value>
  </property>
  <!--drivername-->
  <property>
    <name>javax.jdo.option.ConnectionDriverName</name>
    <value>com.mysql.jdbc.Driver</value>
  </property>
```

```
<!--username-->
<property>
  <name>javax.jdo.option.ConnectionUserName</name>
  <value>root</value>
</property>
<!--password-->
<property>
  <name>javax.jdo.option.ConnectionPassword</name>
  <value>root</value>
</property>
</configuration>
```

登录mysql创建hive数据库(使用命令行创建)

```
create database hive
```

复制mysql驱动jar到hive的lib目录中

4 启动

1. 启动 hadoop

启动hadoop

```
# 启动HDFS
start-dfs.sh
# 启动yarn
start-yarn.sh
```

```
[root@hive60 ~]# jps
8405 SecondaryNameNode
9174 Jps
8888 NodeManager
8794 ResourceManager
8110 NameNode
8239 DataNode
```

2. 本地启动hive

```
初始化元数据: schematool -dbType mysql -initSchema
```

初始化mysql的hivedatabase中的信息。

3. 启动Hive的两种方式

本地模式启动 【管理员模式】

启动hive服务器，同时进入hive的客户端。只能通过本地方式访问。

```
[root@hadoop10 ~]# hive
Logging initialized using configuration in jar:file:/opt/installs/hive1.2.1/lib/hive-common-1.2.1.jar!/hive-log4j.properties
hive>
```

1. 客户端操作之dfs命令

1. 查看dfs中的文件。
dfs -ls /;
2. 查看dfs中 /user 下的文件
dfs -ls /user;
3. 以递归的方式，查看/user下的所有文件
dfs -lsr /user;

2.客户端操作之HQL(Hive Query language)

- # 1.查看数据库
hive> show databases;
- # 2. 创建一个数据库
hive> create database baizhi;
- # 3. 查看database
hive> show databases;
- # 4. 切换进入数据库
hive> use baizhi;
- # 5.查看所有表
hive> show tables;
- # 6.创建一个表
hive> create table t_user(id string,name string,age int);
- # 7. 添加一条数据(转化为MR执行--不让用，仅供测试)
hive> insert into t_user values('1001','zhangsan',20);
- # 8.查看表结构
hive> desc t_user;
- # 9.查看表的schema描述信息。(表元数据，描述信息)
hive> show create table t_user;
明确看到，该表的数据存放在hdfs中。
- # 10 .查看数据库结构
hive> desc database baizhi;
- # 11.查看当前库
hive> select current_database();
- # 12 其他sql
select * from t_user;
select count(*) from t_user; (Hive会启动MapReduce)
select * from t_user order by id;

3.hive的客户端和服务端

启动hive的服务器，可以允许远程连接方式访问。

// 前台启动

```
[root@hadoop10 ~]# hiveserver2
```

// 后台启动

```
[root@hadoop10 ~]# hiveserver2 &
```

beeline客户端

启动客户端

```
[root@hadoop10 ~]# beeline
```

```
beeline> !connect jdbc:hive2://hadoop10:10000
```

回车输入mysql用户名

回车输入mysql密码

DBeaver客户端(图形化界面)

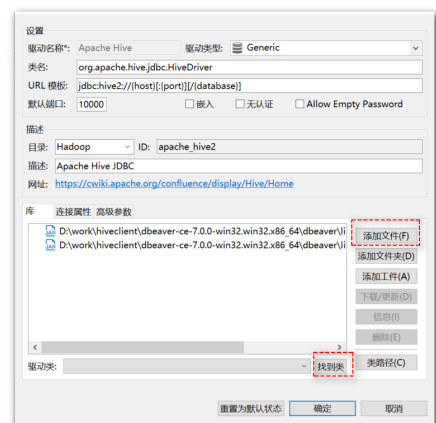
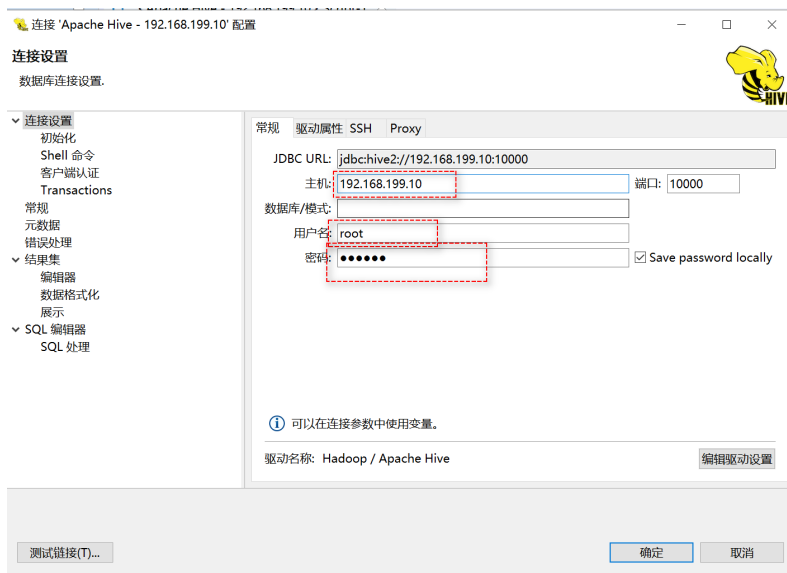
1: 解压

2: 准备dbeaver连接hive的依赖jar

hadoop-common-2.9.2

hive-jdbc-1.2.1-standalone

3: 启动



JDBC

导入依赖

```
<dependency>
  <groupId>org.apache.hive</groupId>
  <artifactId>hive-jdbc</artifactId>
  <version>1.2.1</version>
</dependency>
<dependency>
  <groupId>org.apache.hadoop</groupId>
  <artifactId>hadoop-common</artifactId>
  <version>2.9.2</version>
</dependency>
```

JDBC操作Hive

```
public static void main(String[] args) throws Exception {
    BasicConfigurator.configure();//开启日志
    //加载hive驱动
    Class.forName("org.apache.hive.jdbc.HiveDriver");
    //连接hive数据库
    Connection conn =
    DriverManager.getConnection("jdbc:hive2://hadoop10:10000/baizhi","root","admins");
    String sql = "select * from t_user1";
    PreparedStatement pstmt = conn.prepareStatement(sql);
    ResultSet rs = pstmt.executeQuery();
    while(rs.next()){
        String id = rs.getString("id");
        String name = rs.getString("name");
        int age = rs.getInt("age");
        System.out.println(id+": "+name+": "+age);
    }
    rs.close();
    pstmt.close();
    conn.close();
}
```

4. 数据类型

数据类型 (`primitive`, `array`, `map`, `struct`)

- primitive(原始类型):

hive数据类型	字节	备注
TINYINT	1	java-byte 整型
SMALLINT	2	java-short 整型
INT	4	java-int 整型
BIGINT	8	java-long 整型
BOOLEAN		布尔
FLOAT	4	浮点型
DOUBLE	8	浮点型
STRING		字符串 无限制
VARCHAR		字符串 varchar(20) 最长20
CHAR		字符串 char(20) 定长20
BINARY		二进制类型
TIMESTAMP		时间戳类型
DATE		日期类型

- array（数组类型）：

```
# 建表
create table t_tab(
    score array<float>,
    字段名 array<泛型>
);
```

- map（key-value类型）：MAP <primitive_type, data_type>

```
# 建表
create table t_tab(
    score map<string,float>
);
```

- struct（结构体类型）：STRUCT <col_name:data_type, ...>

```
# 建表
create table t_tab(
    info struct<name:string,age:int,sex:char(1)>,
    列名 struct<属性名:类型,属性名:类型>
);
```

Hive数据导入

1.自定义分隔符[重点]

分隔符设计

分隔符	含义	备注
,	用来表示每个列的值之间分隔符。 <code>fields</code>	
-	用来分割array中每个元素，以及struct中的每个值，以及map中kv与kv之间。 <code>collection items</code>	
	用来分割map的k和v之间 <code>map keys</code>	
\n	每条数据分割使用换行。 <code>lines</code>	

建表

```
create table t_person(  
    id string,  
    name string,  
    salary double,  
    birthday date,  
    sex char(1),  
    hobbies array<string>,  
    cards map<string,string>,  
    addr struct<city:string,zipCode:string>  
) row format delimited  
fields terminated by ','--列的分割  
collection items terminated by '-'--数组 struct的属性 map的kv和kv之间  
map keys terminated by '|'-- map的k与v的分割  
lines terminated by '\n';--行数据之间的分割
```

测试数据

```
1,张三,8000.0,2019-9-9,1,抽烟-喝酒-烫头,123456|中国银行-22334455|建设银行,北京-10010  
2,李四,9000.0,2019-8-9,0,抽烟-喝酒-烫头,123456|中国银行-22334455|建设银行,郑州-45000  
3,王五,7000.0,2019-7-9,1,喝酒-烫头,123456|中国银行-22334455|建设银行,北京-10010  
4,赵6,100.0,2019-10-9,0,抽烟-烫头,123456|中国银行-22334455|建设银行,郑州-45000  
5,于谦,1000.0,2019-10-9,0,抽烟-喝酒,123456|中国银行-22334455|建设银行,北京-10010  
6,郭德纲,1000.0,2019-10-9,1,抽烟-烫头,123456|中国银行-22334455|建设银行,天津-20010
```

导入数据

在hive命令行中执行

-- local 代表本地路径, 如果不写, 代表读取文件来自于HDFS
-- overwrite 是覆盖的意思, 可以省略。

```
load data [local] inpath '/opt/datas/person1.txt' [overwrite] into table t_person;
```

本质上就是将数据上传到hdfs中(数据是受hive的管理)

2.JSON分割符

jar添加和数据导入, 建表, 在beeline里面操作

数据

1.本地创建json文件

```
{"id":1,"name":"zhangsan","sex":0,"birth":"1991-02-08"}  
{"id":2,"name":"lisi","sex":1,"birth":"1991-02-08"}
```

添加格式解析器的jar(本地客户端命令)

在hive的客户端执行(临时添加jar到hive的classpath, 有效期本链接内)

```
add jar /opt/installs/hive1.2.1/hcatalog/share/hcatalog/hive-hcatalog-core-1.2.1.jar
```

补充: 永久添加, Hive服务器级别有效。

1. 将需要添加到hive的classpath的jar, 拷贝到hive下的auxlib目录下,
2. 重启hiveserver即可。

建表

```
create table t_person2(  
    id string,  
    name string,  
    sex char(1),  
    birth date  
)row format serde 'org.apache.hive.hcatalog.data.JsonSerDe';
```

加载文件数据(本地客户端命令)

注意: 导入的json数据dbeaver看不了。(因为导入后的表本质上就是该json文件。)

```
load data local inpath '/opt/person.json' into table t_person2;
```

查看数据

```
select * from t_person2;
```

Result					
select * from t_person2 输入一个 SQL 表达式来					
	id	name	sex	birth	
1	1	zhangsan	1	1991-02-08	
2	2	lisi	0	1991-02-08	

3.正则分隔符

数据: access.log

下边列于列之间的分割符没有完全统一

```
INFO 192.168.1.1 2019-10-19 QQ com.baizhi.service.IUserService#login
INFO 192.168.1.1 2019-10-19 QQ com.baizhi.service.IUserService#login
ERROR 192.168.1.3 2019-10-19 QQ com.baizhi.service.IUserService#save
WARN 192.168.1.2 2019-10-19 QQ com.baizhi.service.IUserService#login
DEBUG 192.168.1.3 2019-10-19 QQ com.baizhi.service.IUserService#login
ERROR 192.168.1.1 2019-10-19 QQ com.baizhi.service.IUserService#register
```

建表语句

```
create table t_access(
  level string,
  ip string,
  log_time date,
  app string,
  service string,
  method string
)row format serde 'org.apache.hadoop.hive.serde2.RegexSerDe' --正则表达式的格式转化类
with serdeproperties("input.regex"="(.*?)\\s(.*?)\\s(.*?)\\s(.*?)\\s(.*?)#(.*?)");--(.*?) 表示任意字符
\\s表示空格
```

导入数据

```
load data local inpath '/opt/access.log' into table t_access;
```

查看数据

```
select * from t_access;
```

select * from t_access;						
Result						
select * from t_access 输入一个 SQL 表达式来过滤结果 (使用 Ctrl+Space)						
	level	ip	log_time	app	service	method
1	INFO	192.168.1.1	2019-10-19	QQ	com.baizhi.service.IUserService	login
2	INFO	192.168.1.1	2019-10-19	QQ	com.baizhi.service.IUserService	login
3	ERROR	192.168.1.3	2019-10-19	QQ	com.baizhi.service.IUserService	save
4	WARN	192.168.1.2	2019-10-19	QQ	com.baizhi.service.IUserService	login
5	DEBUG	192.168.1.3	2019-10-19	QQ	com.baizhi.service.IUserService	login
6	ERROR	192.168.1.1	2019-10-19	QQ	com.baizhi.service.IUserService	register

HQL高级

-- SQL关键词执行顺序 from > where条件 > group by > having条件 > select > order by > limit

注意: sql一旦出现group by, 后续的关键词能够操作字段只有(分组依据字段, 组函数处理结果)

HQL高级

0. 各个数据类型的字段访问(array、map、struct)

```
select name,salary,hobbies[1],cards['123456'],addr.city from t_person;
```

1. 条件查询: = != >= <=

```
select * from t_person where addr.city='郑州';
```

2. and or between and

```
select * from t_person where salary>5000 and array_contains(hobbies,'抽烟');
```

3. order by[底层会启动mapreduce进行排序]

```
select * from t_person order by salary desc;
```

4. limit(hive没有起始下标)

```
select * from t_person sort by salary desc limit 5;
```

5. 去重

```
select distinct addr.city from t_person;
```

```
select distinct(addr.city) from t_person;
```

表连接

```
select ...
```

```
from table1 t1 left join table2 t2 on 条件
```

```
where 条件
```

```
group by
```

```
having
```

查询性别不同，但是薪资相同的人员信息。

```
select
    t1.name,t1.sex,t1.salary,
    t2.name,t2.sex,t2.salary
from t_person t1 join t_person t2 on t1.salary = t2.salary
where t1.sex != t2.sex;
```

id	姓名	name	salary	hobbies	city	name	salary	hobbies	city
		李四	9,000	["抽烟","喝酒","烫头"]	郑州	张三	8,000	["抽烟","喝酒","烫头"]	北京
		张三	8,000	["抽烟","喝酒","烫头"]	北京	李四	9,000	["抽烟","喝酒","烫头"]	郑州
		于谦	1,000	["抽烟","喝酒"]	北京	李四	9,000	["抽烟","喝酒","烫头"]	郑州
		赵6	1,000	["抽烟","烫头"]	郑州	王五	7,000	["喝酒","烫头"]	北京
		郭德纲	1,000	["抽烟","烫头"]	天津	王五	7,000	["喝酒","烫头"]	北京
		王五	7,000	["喝酒","烫头"]	北京	赵6	1,000	["抽烟","烫头"]	郑州
		郭德纲	1,000	["抽烟","烫头"]	天津	赵6	1,000	["抽烟","烫头"]	郑州
		李四	9,000	["抽烟","喝酒","烫头"]	郑州	于谦	1,000	["抽烟","喝酒"]	北京
		王五	7,000	["喝酒","烫头"]	北京	郭德纲	1,000	["抽烟","烫头"]	天津
		赵6	1,000	["抽烟","烫头"]	郑州	郭德纲	1,000	["抽烟","烫头"]	天津

单行函数(show functions) 查看所有函数

-- 查看hive系统所有函数

show functions;

1. array_contains(列,值);

select name,hobbies from t_person where array_contains(hobbies,'喝酒');

2. length(列)

select length('123123');

3. concat(列,列)

select concat('123123','aaaa');

4. to_date('1999-9-9')

select to_date('1999-9-9');

5. year(date),month(date),

6. date_add(date,数字)

select name,date_add(birthday,-9) from t_person;

组函数

概念:

max、min、sum、avg、count等。

select max(salary) from t_person where addr.city='北京';

select count(id) from t_person;

炸裂函数(集合函数)

```
-- 查询所有的爱好
select explode(hobbies) as hobby from t_person
```

```
# lateral view
-- 为指定表, 的边缘拼接一个列。(类似表连接)
-- lateral view: 为表的拼接一个列(炸裂结果)
-- 语法: from 表 lateral view explode(数组字段) 别名 as 字段名;
```

```
-- 查看id, name, 爱好。一个爱好一条信息。
select id,name,hobby
from t_person lateral view explode(hobbies) t_hobby as hobby
```

分组

```
1. group by(查看各个城市的均薪)
select addr.city,avg(salary) from t_person group by addr.city;
2. having(查看平均工资超过5000的城市和均薪)
select addr.city,avg(salary) from t_person group by addr.city having avg(salary)>5000;
3. 统计各个爱好的人数
--explod+lateral view
select hobby,count( * )
from t_person lateral view explode(hobbies) t_hobby as hobby
group by hobby;
4. 统计最受欢迎的爱好TOP1
SELECT hb,count( * ) num
      from t_person lateral view explode(hobbies) h as hb
      group by hb
      order by num desc limit 1;
```

子查询

```
-- 统计有哪些爱好, 并去重。
select distinct t.hobby from
(select explode(hobbies) as hobby from t_person ) t
```

行列相转

案例表和数据

```
--## 表 (电影观看日志)
```

```
create table t_visit_video (
    username string,
    video_name string,
    video_date date
)row format delimited fields terminated by ',';
--## 数据：豆瓣观影日志数据。（用户观影日志数据 按照天存放 1天一个日志文件）
张三,大唐双龙传,2020-03-21
李四,天下无贼,2020-03-21
张三,神探狄仁杰,2020-03-21
李四,霸王别姬,2020-03-21
李四,霸王别姬,2020-03-21
王五,机器人总动员,2020-03-21
王五,放牛班的春天,2020-03-21
王五,盗梦空间,2020-03-21
```

`select * from t_visit_video` | 输入一个 SQL 表达式来过滤结果

	username	video_name	video_date
1	张三	大唐双龙传	2020-03-21
2	李四	天下无贼	2020-03-21
3	张三	神探狄仁杰	2020-03-21
4	李四	霸王别姬	2020-03-21
5	李四	霸王别姬	2020-03-21
6	王五	机器人总动员	2020-03-21
7	王五	放牛班的春天	2020-03-21
8	王五	盗梦空间	2020-03-21

collect_list(组函数)

作用：对分组后的，每个组的某个列的值进行收集汇总。

语法：select collect_list(列) from 表 group by 分组列；

```
select username,collect_list(video_name) from t_visit_video group by username;
```

`select username,collect_set(video_name) from t_visit_video group by username;` | 输入一个 SQL 表达式来过滤结果

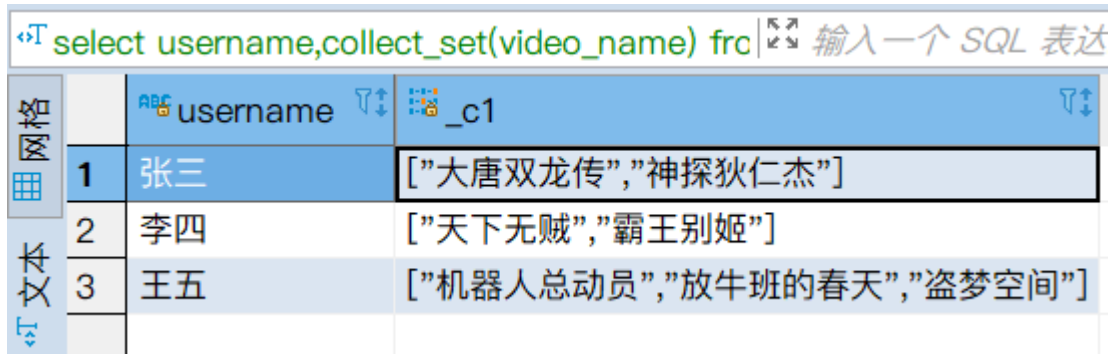
	username	_c1
1	张三	["大唐双龙传","神探狄仁杰"]
2	李四	["天下无贼","霸王别姬","霸王别姬"]
3	王五	["机器人总动员","放牛班的春天","盗梦空间"]

collect_set(组函数)

作用：对分组后的，每个组的某个列的值进行收集汇总，并去掉重复值。

语法：select collect_set(列) from 表 group by 分组列；

```
select username,collect_set(video_name) from t_visit_video group by username;
```



The screenshot shows a SQL query editor with the query: `select username,collect_set(video_name) from t_visit_video group by username;`. The result is displayed in a table with two columns: `username` and `_c1`. The data is as follows:

	username	_c1
1	张三	["大唐双龙传","神探狄仁杰"]
2	李四	["天下无贼","霸王别姬"]
3	王五	["机器人总动员","放牛班的春天","盗梦空间"]

concat_ws(单行函数)

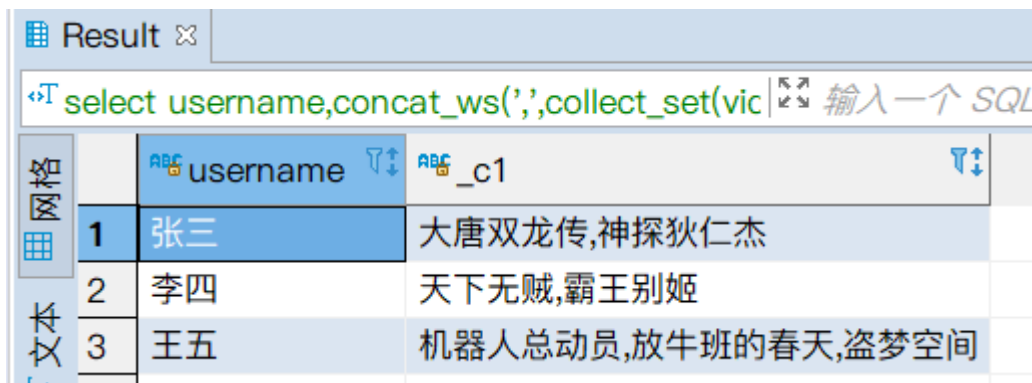
作用：如果某个字段是数组，对该值得多个元素使用指定分隔符拼接。

select id,name,concat_ws(',',hobbies) from t_person;

--# 将t_visit_video数据转化为如下图效果

--统计每个人，2020-3-21看过的电影。

```
select username,concat_ws(',',collect_set(video_name)) from t_visit_video group by username;
```



The screenshot shows a SQL query editor with the query: `select username,concat_ws(',',collect_set(video_name)) from t_visit_video group by username;`. The result is displayed in a table with two columns: `username` and `_c1`. The data is as follows:

	username	_c1
1	张三	大唐双龙传,神探狄仁杰
2	李四	天下无贼,霸王别姬
3	王五	机器人总动员,放牛班的春天,盗梦空间

全排序和局部排序

全局排序

语法：select * from 表 order by 字段 asc|desc;

-- 按照薪资降序排序

```
select * from t_person order by salary desc;
```

局部排序(分区排序)

概念：启动多个reduceTask，对数据进行排序(预排序)，局部有序。

局部排序关键词 `sort by`

默认reducetask个数只有1个，所有分区也只有一个。所以默认和全排序效果一样。

语法：`select * from 表 distribute by 分区字段 sort by 字段 asc|desc;`

```
-- 1. 开启reduce个数
-- 设置reduce个数
set mapreduce.job.reduces = 3;
-- 查看reduce个数
set mapreduce.job.reduces;
-- 2. 使用sort by排序 +distribute by 指定分区列。(使用distribute后select就只能*)
select * from t_person distribute by addr.city sort by salary desc;
-- 3. 可以将查询结果写入本地磁盘，用于测试sort by的效果
insert overwrite local directory '/opt/data/sortby'
select * from t_person distribute by addr.city sort by salary desc;
```

Hive中表分类

4.1 管理表

由Hive全权管理的表

所谓的管理表指hive是否具备数据的管理权限，如果该表是管理表，当用户删除表的同时，hive也会将表所对应的数据删除，因此在生产环境下，为了防止误操作，带来数据损失，一般考虑将表修改为非管理表-外部表

总结：Hive的管理表，包含表结构，hdfs中表的数据文件，都归Hive全权管理。---- hive删除管理表，HDFS对应文件也会被删除。

缺点：数据不安全。

4.2 外部表

引用映射HDFS数据作为表管理,但无法删除数据

外部表和管理表最大的区别在于删除外部表，只是将MySQL中对应该表的元数据信息删除，并不会删除hdfs上的数据，因此外部表可以实现和第三方应用共享数据。在创建外表的时候需要添加一个关键字"external"即可。`create external xxx()...`

创建外部表

1. 准备数据文件personout.txt
2. 上传至hdfs中，该数据文件必须被放在一个单独的文件夹内。该文件夹内的数据文件被作为表数据
3. 创建表：`create external location`
在最后使用location 指定hdfs中数据文件所在的文件夹即可。
`create external table t_personout(
 id int,
 name string,
 salary double,

 birthday date,`

```
sex char(1),
hobbies array<string>,
cards map<string,string>,
addr struct<city:string,zipCode:string>
)row format delimited
fields terminated by ',' --列的分割
collection items terminated by '-'-数组 struct的属性 map的kv和kv之间
map keys terminated by '|'
lines terminated by '\n'
```

4. 查询表数据

select * from t_personout 输入一个 SQL 表达式来过滤结果 (使用 Ctrl+Space)

	id	name	salary	birthday	sex	hobbies	cards	
1	10,002	huxz	10,000	1999-09-09	1	["smoke","drink"]	{"001":"中国银行","002":"建设银行"}	[{"c
2	10,003	wangmj	10,001	1909-09-09	0	["hothead","drink"]	{"001":"中国银行","002":"建设银行"}	[{"c

4.3 分区表

将表按照某个列的一定规则进行分区存放，减少海量数据情况下的数据检索范围，提高查询效率；

举例：电影表、用户表

分区方案：按照用户区域、电影类型

应用：依据实际业务功能，拿查询条件的列作为分区列来进行分区，缩小MapReduce的扫描范围，提高MapReduce的执行效率，

总结：

table中的多个分区的数据是分区管理

1：删除数据按照分区删除。如果删除某个分区，则将分区对应的数据也删除(外部表，数据删除，数据文件依然在)。

2：查询统计，多个分区被一个表管理起来。

select * from 表 where 分区字段为条件。

4.3.1 创建分区表

数据源文件

文件"bj.txt" (china bj数据)

1001,张三,1999-1-9,1000.0
1002,李四,1999-2-9,2000.0
1008,孙帅,1999-9-8,50000.0
1010,王宇希,1999-10-9,10000.0
1009,刘春阳,1999-9-9,10.0

文件"tj.txt" (china tj数据)

1006,郭德纲,1999-6-9,6000.0
1007,胡鑫喆,1999-7-9,7000.0

建表

```
create external table t_user_part(  
    id string,  
    name string,  
    birth date,  
    salary double  
)partitioned by(country string,city string)--指定分区列,按照国家和城市分区。  
row format delimited  
fields terminated by ','  
lines terminated by '\n';
```

创建分区表并导入数据

```
# 导入china和bj的数据  
load data local inpath "/opt/bj.txt" into table t_user_part  
partition(country='china',city='bj');  
# 导入china和heb的数据  
load data local inpath "/opt/tj.txt" into table t_user_part  
partition(country='china',city='tj');
```

查看分区信息

```
show partitions t_user_part;
```

使用分区查询:本质上只要查询条件在存在分区列

```
select * from t_user_part where city = 'bj'
```

表分类

1. 管理表

hive除了管理mysql中的元数据,也管理HDFS上的目录和文件

2. 外部表--常用--hdfs文件安全。

hive的table数据,如果删除hive中的table,外部hdfs的数据文件依旧保留。

3. 分区表--重要。

将table按照不同分区管理。

好处: 如果where条件中有分区字段,则Hive会自动对分区内的数据进行检索(不再扫描其他分区数据),提高hive的查询效率。

实际开发中 = 外部表 + 分区表

Hive自定义函数

udf(用户自定义单行函数) udaf(用户自定义聚合(组)函数) udtf(用户炸裂函数)

内置函数

```
# 查看hive内置函数
show functions;
# 查看函数描述信息
desc function max;
```

用户自定义函数UDF

用户自定义函数-UDF :user-defined function

操作作用于单个数据行，并且产生一个数据行作为输出。大多数函数都属于这一类（比如数学函数和字符串函数）。

1. 用户自定义函数-UDF

user-defined function

操作作用于单个数据行，并且产生一个数据行作为输出。大多数函数都属于这一类（比如数学函数和字符串函数）。

简单来说：

UDF:返回对应值，一对一

0. 导入hive依赖

3

1. 定义一个类继承UDF

1. 必须继承UDF
2. 方法名必须是evaluate

```
import org.apache.hadoop.hive.q1.exec.Description;
import org.apache.hadoop.hive.q1.exec.UDF;

public class HelloUDF extends UDF {
    // 方法名必须叫evaluate
    public String evaluate(String s1,String s2){
        return s1 + "-" + s2;
    }
}
```

2. 配置maven打包环境，打包jar

```
<properties>
  <!--解决编码的GBK的问题-->
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
<build>
  <finalName>funcHello</finalName>
</build>
```

打包

```
mvn package
```

3. 上传linux, 导入到函数库中。

在hive命令中执行

```
add jar /opt/data/funcHello.jar; # hive session级别的添加,
delete jar /opt/data/funcHello.jar; # 如果重写, 记得删除。
```

```
create [temporary] function hello as "function.HelloUDF"; # temporary是会话级别。
# 删除导入的函数
drop [temporary] function hello;
```

4. 查看函数并使用函数

--1. 使用函数进行查询

```
select hello(userid,cityname) from logs;
```

导入奇葩的依赖方法-pentahu

下载

```
https://public.nexus.pentaho.org/repository/proxied-pentaho-public-repos-
group/org/pentaho/pentaho-aggdesigner-algorithm/5.1.5-jhyde/pentaho-aggdesigner-algorithm-5.1.5-
jhyde-javadoc.jar
```

放在本地英文目录下

```
D:\work\pentaho-aggdesigner-algorithm-5.1.5-jhyde-javadoc.jar
```

执行mvn安装本地依赖的命令

```
D:\work> mvn install:install-file -DgroupId=org.pentaho -DartifactId=pentaho-aggdesigner-
algorithm -Dversion=5.1.5-jhyde -Dpackaging=jar -Dfile=pentaho-aggdesigner-algorithm-5.1.5-
jhyde-javadoc.jar
```

案例

列自增长(不确定性函数)

定义一个函数 get_number()

```
select get_num() num,id,name,salary from t_person;
```



	num	id	name	salary
1	1	1	张三	8,000
2	2	2	李四	9,000
3	3	3	王五	7,000
4	4	4	赵6	1,000
5	5	5	于谦	1,000
6	6	6	郭德纲	1,000

编码

1. 定义一个java类

继承UDF

书写evaluate方法

```
import org.apache.hadoop.hive.ql.exec.Description;  
import org.apache.hadoop.hive.ql.exec.UDF;  
import org.apache.hadoop.io.LongWritable;
```

@UDFType(deterministic = false) //输入确定，输出确定的函数，false，因为该函数没有输入，输出结果也会变化。

```
public class NumberUDF extends UDF {  
    private long index = 0;  
    public long evaluate(){  
        index++;  
        return index;  
    }  
}
```

2. 打包

```
mvn clean package
```

3. 上传linux

4. 导入到hive的依赖库中

```
add jar /opt/doc/myhive1.2.jar;
```

5. 创建函数

```
create temporary function get_num as 'function.NumberUDF';
```

6. 使用

```
select get_num() num,id,name,salary from t_person;
```

```

/**
 * 该对象hive只会创建1个，每行数据处理，调用一次evaluate方法。
 * UDF udf = new NumberUDF();
 * while(row.next){
 *     udf.evaluate();
 * }
 */

@UDFType(deterministic = false);
public class NumberUDF extends UDF {
    private long index = 0;
    public long evaluate() {
        index++;
        return index;
    }
}

```

标记该函数是不确定性函数。

用户自定义函数UDTF

自定义一个 UDTF 实现将一个任意分割符的字符串切割成独立的单词，例如：

Line:"hello,world,hadoop,hive" Myudtf(line, ",")

hello world hadoop hive

代码实现：

```

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

import java.util.ArrayList;
import java.util.List;

public class MyUDTF extends GenericUDTF {
    private ArrayList<String> outList = new ArrayList<String>();

    @Override
    public StructObjectInspector initialize(StructObjectInspector argOIs) throws
UDFArgumentException {
        //1.定义输出数据的列名和类型
        List<String> fieldNames = new ArrayList<String>();

        List<ObjectInspector> fieldOIs = new ArrayList<ObjectInspector>();

```



```

//2.添加输出数据的列名和类型
fieldNames.add("lineToWord");

field0Is.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);
return ObjectInspectorFactory.getStandardStructObjectInspector(fieldNames, field0Is);
}

@Override
public void process(Object[] args) throws HiveException {
    //1.获取原始数据
    String arg = args[0].toString();
    //2.获取数据传入的第二个参数，此处为分隔符
    String splitKey = args[1].toString();
    //3.将原始数据按照传入的分隔符进行切分
    String[] fields = arg.split(splitKey);
    //4.遍历切分后的结果，并写出
    for (String field : fields) {
        //集合为复用的，首先清空集合
        outList.clear();
        //将每一个单词添加至集合
        outList.add(field);
        //将集合内容写出
        forward(outList);
    }
}

@Override
public void close() throws HiveException {
}
}

```

测试方式同自定义UDF：打包、添加jar、创建函数...

```

add jar xxxxx.jar;
create temporary function myudtf as "com.baizhi.function.MyUDTF";
select myudtf(line, ",") word

```

表数据转存导入操作

```

# 1.将文件数据导入hive表中,
load data local inpath '文件的路径' overwrite into table 表。
# 2.直接将查询结果，放入一个新创建的表中。(执行查询的创建)
create table 表 as select语...
    1. 执行select语句
    2. 创建一个新的表，将查询结果存入表中。
# 3.将查询结果，导入已经存在表。
insert into 表

```

```
select语句...
```

```
insert overwrite table 表
```

```
select语句...
```

4. 将HDFS中已经存在文件，导入新建的hive表中

```
create table Xxx(
```

```
...
```

```
)row format delimited
```

```
fields terminated by ','
```

```
location 'hdfs的表数据对应的目录'
```

将SQL的执行结果插入到另一个表中

```
create table 表 as select语句
```

--## 例子:

--统计每个人, 2020-3-21看过的电影, 将结果存入hive的表: t_video_log_20200321

```
create table t_video_log_20200321 as select ...;
```