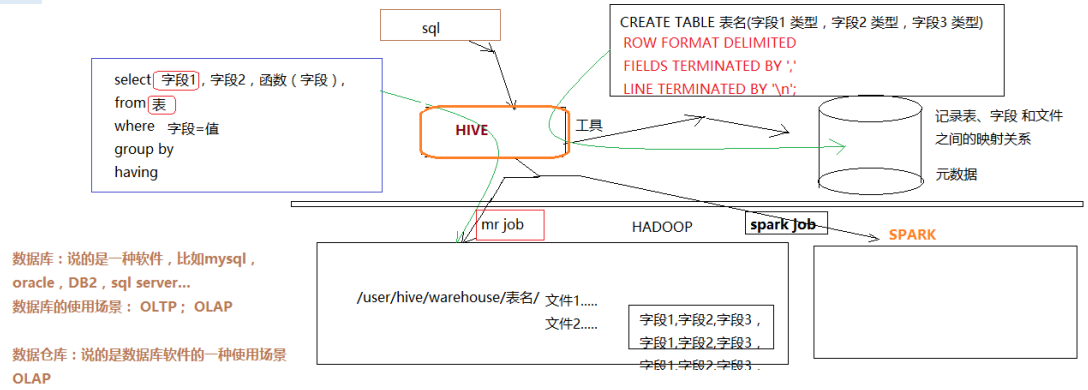
1.什么是hive

1.1. hive基本思想

Hive是基于Hadoop的一个数据仓库工具(离线)，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。



1.2. 为什么使用Hive

- 直接使用hadoop所面临的问题
  - 人员学习成本太高
  - 项目周期要求太短
  - MapReduce实现复杂查询逻辑开发难度太大

- 为什么要使用Hive
  - 操作接口采用类SQL语法，提供快速开发的能力。
  - 避免了去写MapReduce，减少开发人员的学习成本。
  - 功能扩展很方便。

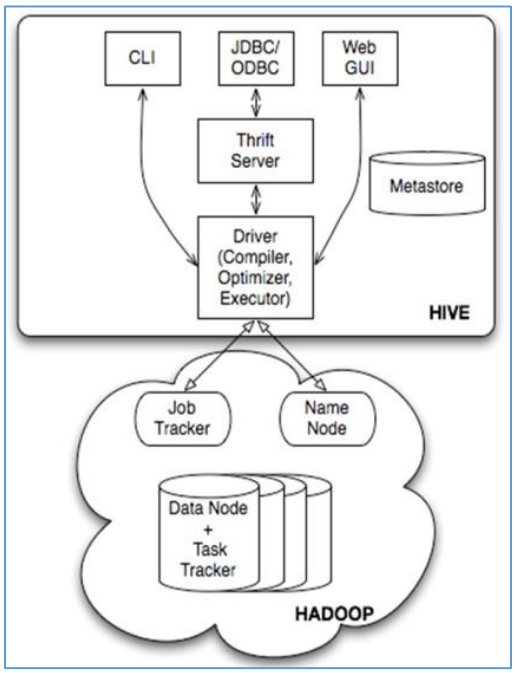
1.3. Hive的特点

- 可扩展
  - Hive可以自由的扩展集群的规模，一般情况下不需要重启服务。

- 延展性
  - Hive支持用户自定义函数，用户可以根据自己的需求来实现自己的函数。

- 容错
  - 良好的容错性，节点出现问题SQL仍可完成执行。

2.hive的基本架构



Jobtracker是hadoop1.x中的组件，它的功能相当于：

Resourcemanager+MRAppMaster

TaskTracker 相当于：

Nodemanager + yarnchild

3.hive安装

知识补充	补充：先将hadoop集群的机器配置时间同步 时间同步 yum install ntpdate -y　## 安装时间同步客户端 ntpdate 0.asia.pool.ntp.org　　## 与互联网时间服务器同步 若上面的时间服务器不可用，也可以选择以下服务器同步时间 time.nist.gov time.nuri.net 0.asia.pool.ntp.org 1.asia.pool.ntp.org 2.asia.pool.ntp.org 3.asia.pool.ntp.org

3.1. 最简安装：用内嵌derby作为元数据库

准备工作：安装hive的机器上应该有HADOOP环境（安装目录，HADOOP_HOME环境变量）
安装：直接解压一个hive安装包即可
此时，安装的这个hive实例使用其内嵌的derby数据库作为记录元数据的数据库
此模式不便于让团队成员之间共享协作

3.2. 标准安装：将mysql作为元数据库

3.2.1.mysql安装

1.上传mysql安装包
2.解压：
[root@mylove ~]# tar -xvf MySQL-5.6.26-1.linux_glibc2.5.x86_64.rpm-bundle.tar

3.安装mysql的server包
[root@mylove ~]# rpm -ivh MySQL-server-5.6.26-1.linux_glibc2.5.x86_64.rpm
依赖报错：
缺perl
yum install perl
安装完perl后，继续重新安装mysql-server
（可以配置一个本地yum源进行安装：

1、先在vmware中给这台虚拟机连接一个光盘镜像
2、挂在光驱到一个指定目录：mount -t iso9660 -o loop /dev/cdrom /mnt/cdrom
3、将yum的配置文件中baseURL指向/mnt/cdrom
)

[root@mylove ~]# rpm -ivh MySQL-server-5.6.26-1.linux_glibc2.5.x86_64.rpm
又出错：包冲突conflict with
移除老版本的冲突包：mysql-libs-5.1.73-3.el6_5.x86_64
[root@mylove ~]# rpm -e mysql-libs-5.1.73-3.el6_5.x86_64 --nodeps

继续重新安装mysql-server
[root@mylove ~]# rpm -ivh MySQL-server-5.6.26-1.linux_glibc2.5.x86_64.rpm

成功后，注意提示：里面有初始密码及如何改密码的信息
初始密码：/root/.mysql_secret
改密码脚本：/usr/bin/mysql_secure_installation

4.安装mysql的客户端包：
[root@mylove ~]# rpm -ivh MySQL-client-5.6.26-1.linux_glibc2.5.x86_64.rpm

5.启动mysql的服务端：
[root@mylove ~]# service mysql start
Starting MySQL. SUCCESS!

6.修改root的初始密码：
[root@mylove ~]# /usr/bin/mysql_secure_installation 按提示

7.测试：
用mysql命令行客户端登陆mysql服务器看能否成功
[root@mylove ~]# mysql -uroot -proot
mysql> show databases;

8.给root用户授予从任何机器上登陆mysql服务器的权限：
mysql> grant all privileges on *.* to 'root'@'%' identified by '你的密码' with grant option;
Query OK, 0 rows affected (0.00 sec)

mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)

注意点：要让mysql可以远程登录访问
最直接测试方法：从windows上用Navicat去连接，能连，则可以，不能连，则要去mysql的机器上用命令行客户端进行授权：
在mysql的机器上,启动命令行客户端：
mysql -uroot -proot

```
mysql>grant all privileges on *.* to 'root'@'%' identified by 'root的密码' with grant option;
mysql>flush privileges;
```

3.2.2.hive的元数据库配置

vi conf/hive-site.xml

```
<configuration>
<property>
<name>javax.jdo.option.ConnectionURL</name>
<value>jdbc:mysql://localhost:3306/hive?createDatabaseIfNotExist=true</value>
<description>JDBC connect string for a JDBC metastore</description>
</property>

<property>
<name>javax.jdo.option.ConnectionDriverName</name>
<value>com.mysql.jdbc.Driver</value>
<description>Driver class name for a JDBC metastore</description>
</property>

<property>
<name>javax.jdo.option.ConnectionUserName</name>
<value>root</value>
<description>username to use against metastore database</description>
</property>

<property>
<name>javax.jdo.option.ConnectionPassword</name>
<value>root</value>
<description>password to use against metastore database</description>
</property>
</configuration>
```

2.上传一个mysql的驱动jar包到hive的安装目录的lib中

3.配置HADOOP_HOME 和HIVE_HOME到系统环境变量中： /etc/profile
4.source /etc/profile
5、hive启动测试
然后用命令启动hive交互界面：
[root@hdp20-04 ~]# hive

4.hive使用方式

4.1. 最基本使用方式

启动一个hive交互shell
bin/hive
hive>

设置一些基本参数，让hive使用起来更便捷，比如：
1、让提示符显示当前库：
hive>set hive.cli.print.current.db=true;
2.显示查询结果时显示字段名称：
hive>set hive.cli.print.header=true;

但是这样设置只对当前会话有效，重启hive会话后就失效，解决办法：
在linux的当前用户目录中，编辑一个.hiverc文件，将参数写入其中：
vi .hiverc

```
set hive.cli.print.header=true;
set hive.cli.print.current.db=true;
```

4.2. 启动hive服务使用

启动hive的服务：

```
[root@hdp20-04 hive-1.2.1]# bin/hiveserver2 -hiveconf hive.root.logger=DEBUG,console
```

上述启动，会将这个服务启动在前台，如果要启动在后台，则命令如下：
nohup bin/hiveserver2 1>/dev/null 2>&1 &

启动成功后，可以在别的节点上用beeline去连接
- 方式（1）

[root@hdp20-04 hive-1.2.1]# bin/beeline 回车，进入beeline的命令界面
输入命令连接hiveserver2
beeline> !connect jdbc:hive2//mini1:10000
（hadoop01是hiveserver2所启动的那台主机名，端口默认是10000）
- 方式（2）

启动时直接连接：
bin/beeline -u jdbc:hive2://mini1:10000 -n root

接下来就可以做正常sql查询了

4.3. 脚本化运行

大量的hive查询任务，如果用交互式shell来进行输入的话，显然效率及其低下，因此，生产中更多的是使用脚本化运行机制：
该机制的核心点是：hive可以用一次性命令的方式来执行给定的hql语句

[root@hdp20-04 ~]# hive -e "insert into table t_dest select * from t_src;"

然后，进一步，可以将上述命令写入shell脚本中，以便于脚本化运行hive任务，并控制、调度众多hive任务，示例如下：
vi t_order_etl.sh

```
#!/bin/bash
hive -e "select * from db_order.t_order"
hive -e "select * from default.t_user"
hql="create table  default.t_bash as select * from db_order.t_order"
hive -e "$hql"
```

如果要执行的hql语句特别复杂，那么，可以把hql语句写入一个文件：
vi x.hql

```
select * from db_order.t_order;
select count(1) from db_order.t_user;
```

然后，用hive -f /root/x.hql 来执行

5.hive建库建表与数据导入(DDL)

data define language

5.1. 建库

hive中有一个默认的库：
库名： default
库目录：hdfs://hdp20-01:9000/user/hive/warehouse

新建库：
create database db_order;
库建好后，在hdfs中会生成一个库目录：
hdfs://hdp20-01:9000/user/hive/warehouse/db_order.db

5.2. 建表

5.2.1.基本建表语句

use db_order;
create table t_order(id string,create_time string,amount float,uid string);
表建好后，会在所属的库目录中生成一个表目录

/user/hive/warehouse/db_order.db/t_order
只是，这样建表的话，hive会认为表数据文件中的字段分隔符为 ^A（\001）

正确的建表语句为：
```
create table t_order(id string,create_time string,amount float,uid string)
row format delimited
fields terminated by ',';
```

这样就指定了，我们的表数据文件中的字段分隔符为 ","

注意：*hive*是不会检查用户导入表中的数据的！如果数据的格式跟表定义的格式不一致，*hive*也不会做任何处理（能解析就解析，解析不了就是*null*）；

5.2.2.删除表

```
drop table t_order;
```
删除表的效果是：
- hive会从元数据库中清除关于这个表的信息；
- hive还会从hdfs中删除这个表的表目录；

5.2.3.内部表与外部表

内部表(MANAGED_TABLE)：表目录按照hive的规范来部署，位于hive的仓库目录/user/hive/warehouse中

外部表(EXTERNAL_TABLE)：表目录由建表用户自己指定
```
create external table t_access(ip string,url string,access_time string)
row format delimited
fields terminated by ','
location '/access/log';
```
外部表和内部表的特性差别：
1.内部表的目录在hive的仓库目录中 VS 外部表的目录由用户指定
2.drop一个内部表时：hive会清除相关元数据，并删除表数据目录
3.drop一个外部表时：hive只会清除相关元数据；

一个hive的数据仓库，最底层的表，一定是来自于外部系统，为了不影响外部系统的工作逻辑，在hive中可建external表来映射这些外部系统产生的数据目录；
然后，后续的etl操作，产生的各种中间表建议用managed_table（内部表）

5.2.4.分区表

分区表的实质是：在表目录中为数据文件创建分区子目录，以便于在查询时，MR程序可以

针对指定的分区子目录中的数据进行处理，缩减读取数据的范围，提高效率！

比如，网站每天产生的浏览记录，浏览记录应该建一个表来存放，但是，有时候，我们可能只需要对某一天的浏览记录进行分析
这时，就可以将这个表建为分区表，每天的数据导入其中的一个分区；
当然，每日的分区目录，应该有一个目录名（分区字段）

5.2.4.1.一个分区字段的实例：

示例如下：
1.创建带分区的表

```
create table t_access(ip string,url string,access_time string)
partitioned by(dt string)
row format delimited
fields terminated by ',';
```

注意：分区字段不能是表定义中的已存在字段

2.向分区中导入数据

```
load data local inpath '/root/access.log.2017-08-04.log' into table t_access partition(dt='20170804');
load data local inpath '/root/access.log.2017-08-05.log' into table t_access partition(dt='20170805');
```

3.针对分区数据进行查询
a、统计*8*月*4*号的总*PV*：
select count(*) from t_access where dt='20170804';
实质：就是将分区字段当成表字段来用，就可以使用*where*子句指定分区了

b、统计表中所有数据总的*PV*：
select count(*) from t_access;
实质：不指定分区条件即可

5.2.4.2.多个分区字段示例

建表：

```
create table t_partition(id int,name string,age int)
partitioned by(department string,sex string,howold int)
row format delimited fields terminated by ',';
```

导数据：

```
load data local inpath '/root/p1.dat' into table t_partition
partition(department='xiangsheng',sex='male',howold=20);
```

5.2.5.CTAS建表语法

可以通过已存在表来建表：
1、create table t_user_2 like t_user;
新建的t_user_2表结构定义与源表t_user一致，但是没有数据

2.在建表的同时插入数据

```
create table t_access_user
as
select ip,url from t_access;
```

*t_access_user*会根据*select*查询的字段来建表，同时将查询的结果插入新表中！

补充：将查询出来的数据保存到一张表中：
方式1：create t_x as select .....
方式2：
如果事先存在一张表t_x
可以将select查询出来的结果数据insert到这张已存在的表中;
insert into t_x select .....

5.3. 数据导入导出

5.3.1.将数据文件导入hive的表

方式1：导入数据的一种方式：
手动用hdfs命令，将文件放入表目录；

方式2：在hive的交互式shell中用hive命令来导入本地数据到表目录
hive>load data local inpath '/root/order.data.2' into table t_order;

方式3：用hive命令导入hdfs中的数据文件到表目录
hive>load data inpath '/access.log.2017-08-06.log' into table t_access;
注意：导本地文件和导*HDFS*文件的区别：
本地文件导入表：复制
*hdfs*文件导入表：移动

方式4：如果目标表是一个分区表
hive> load data [local] inpath '.....' into table t_dest partition(p='value');

5.3.2.将hive表中的数据导出到指定路径的文件

1.将hive表中的数据导入HDFS的文件

```
insert overwrite directory '/root/access-data'
row format delimited fields terminated by ','
select * from t_access;
```

2.将hive表中的数据导入本地磁盘文件

```
insert overwrite local directory '/root/access-data'
row format delimited fields terminated by ','
select * from t_access limit 100000;
```

5.3.3.hive文件格式

HIVE支持很多种文件格式： SEQUENCE FILE | TEXT FILE | PARQUET FILE | RC FILE

```
create table t_text(movie string,rate int)  stored as textfile;
create table t_seq(movie string,rate int)  stored as sequencefile;
create table t_pq(movie string,rate int)  stored as parquetfile;
```

演示：

1、先建一个存储文本文件的表

```
create table t_access_text(ip string,url string,access_time string)
row format delimited fields terminated by ','
stored as textfile;
```

导入文本数据到表中：

```
load data local inpath '/root/access-data/000000_0' into table t_access_text;
```

2.建一个存储sequence file文件的表：

```
create table t_access_seq(ip string,url string,access_time string)
stored as sequencefile;
```

从文本表中查询数据插入sequencefile表中，生成数据文件就是sequencefile格式的了：

```
insert into t_access_seq
select * from t_access_text;
```

3.建一个存储parquet file文件的表：

```
create table t_access_parq(ip string,url string,access_time string)
stored as parquetfile;
```

5.4. 数据类型

5.4.1.数字类型

TINYINT (1字节整数)
SMALLINT (2字节整数)
INT/INTEGER (4字节整数)
BIGINT (8字节整数)
FLOAT (4字节浮点数)
DOUBLE (8字节双精度浮点数)

示例：
create table t_test(a string ,b int,c bigint,d float,e double,f tinyint,g smallint)

5.4.2.时间类型

TIMESTAMP (时间戳) (包含年月日时分秒毫秒的一种封装)
DATE (日期)（只包含年月日）
示例，假如有以下数据文件：

```
1,zhangsan,1985-06-31
2,lisi,1986-07-10
3,wangwu,1985-08-09
```

那么，就可以建一个表来对数据进行映射
create table t_customer(id int,name string,birthday date)
row format delimited fields terminated by ',';
然后导入数据
load data local inpath '/root/customer.dat' into table t_customer;
然后，就可以正确查询

5.4.3.字符串类型

STRING
VARCHAR(20) (字符串1-65535长度，超长截断)
CHAR (字符串，最大长度255)

5.4.4.其他类型

BOOLEAN（布尔类型）： true false

~~BINARY (二进制)：~~

举例：
1,zs,28,true
2,ls,30,false
3,ww,32,false
4,lulu,18,true

create table t_p(id int,name string,age int,is_married boolean)

select
from t_p where is_married;

5.4.5.复合（集合）类型

5.4.5.1.array数组类型

arrays: ARRAY<data_type>)

示例：array类型的应用
假如有如下数据需要用hive的表去映射：

战狼2,吴京:吴刚:余男,2017-08-16 三生三世十里桃花,刘亦菲:痒痒,2017-08-20 羞羞的铁拳,沈腾:玛丽:艾伦,2017-12-20

设想：如果主演信息用一个数组来映射比较方便

建表：

```
create table t_movie(moive_name string,actors array<string>,first_show date)
row format delimited fields terminated by ','
collection items terminated by ':';
```

导入数据：
load data local inpath '/root/movie.dat' into table t_movie;

查询：
select * from t_movie;
select moive_name,actors[0] from t_movie;
select moive_name,actors from t_movie where array_contains(actors,'吴刚');
select moive_name,size(actors) from t_movie;

5.4.5.2.map类型

maps: MAP<primitive_type, data_type>

1) 假如有以下数据：

```
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
```

可以用一个map类型来对上述数据中的家庭成员进行描述

2) 建表语句:

```
create table t_person(id int,name string,family_members map<string,string>,age int)
row format delimited fields terminated by ','
collection items terminated by '#'
map keys terminated by ':';
```

3) 查询

```
select * from t_person;

## 取map字段的指定key的值
select id,name,family_members['father'] as father from t_person;

## 取map字段的所有key
select id,name,map_keys(family_members) as relation from t_person;

## 取map字段的所有value
select id,name,map_values(family_members) from t_person;
select id,name,map_values(family_members)[0] from t_person;

## 综合:查询有brother的用户信息
方式1:
select id,name,brother
from
(select id,name,family_members['brother'] as brother from t_person) tmp
where brother is not null;
```

方式2:(效率高,因为不需要两个select去查询)

```
select * from t_map where array_contains(map_keys(family),'sister');
 select * from t_person where array_contains(map_keys(family_members),'brother');
```

5.4.5.3.struct类型

struct: STRUCT<col_name : data_type, ...>

1) 假如有如下数据:

```
1,zhangsan,18:male:beijing
2,lisi,28:female:shanghai
```

其中的用户信息包含:年龄:整数,性别:字符串,地址:字符串

设想用一个字段来描述整个用户信息,可以采用struct

2) 建表:

```
create table t_person_struct(id int,name string,info struct<age:int,sex:string,addr:string>)
```

row format delimited fields terminated by ','
collection items terminated by ':';

3) 查询

*select * from t_person_struct;*
select id,name,info.age from t_person_struct;

5.5. 修改表定义

仅修改Hive元数据，不会触动表中的数据，用户需要确定实际的数据布局符合元数据的定义。

修改表名：
ALTER TABLE table_name RENAME TO new_table_name

示例：alter table t_1 rename to t_x;

修改分区名：
alter table t_partition partition(department='xiangsheng',sex='male',howold=20) rename to partition(department='1',sex='1',howold=20);

添加分区：
alter table t_partition add partition (department='2',sex='0',howold=40);

删除分区：
alter table t_partition drop partition (department='2',sex='2',howold=24);

修改表的文件格式定义：
ALTER TABLE table_name [PARTITION partitionSpec] SET FILEFORMAT file_format

修改表的某个文件格式定义：
alter table t_partition partition(department='2',sex='0',howold=40) set fileformat sequencefile;

修改列名定义：
ALTER TABLE table_name CHANGE [COLUMN] col_old_name col_new_name column_type [COMMENTcol_comment] [FIRST|(AFTER column_name)]

alter table t_user change price jiage float first;

增加/替换列：
ALTER TABLE table_name ADD|REPLACE COLUMNS (col_name data_type[COMMENT col_comment], ...)

alter table t_user add columns (sex string,addr string);
alter table t_user replace columns (id string,age int,price float);

6.hive查询语法

sql是一门面向集合的编程语言；
select 1;

提示：在做小数据量查询测试时，可以让hive将mrjob提交给本地运行器运行，可以在hive会话中设置如下参数：
hive> set hive.exec.mode.local.auto=true;

6.1. 基本查询示例

select * from t_access;
select count(1) from t_access;
select max(ip) from t_access;

6.2. 条件查询

select * from t_access where access_time<'2017-08-06 15:30:20'
select * from t_access where access_time<'2017-08-06 16:30:20' and ip>'192.168.33.3';

6.3. join关联查询示例

假如有a.txt文件

```
a,1
b,2
c,3
d,4
```

假如有b.txt文件

```
a,xx
b,yy
d,zz
e,pp
```

进行各种join查询：
1.inner join（join）

```
select
a.name as aname,
a.numb as anumb,
b.name as bname,
b.nick as bnick
from t_a a
join t_b b
on a.name=b.name
```

结果：

```
+--------+--------+--------+--------+--+
| aname  | anumb  | bname  | bnick  |
+--------+--------+--------+--------+--+
| a      | 1      | a      | xx     |
| b      | 2      | b      | yy     |
| d      | 4      | d      | zz     |
+--------+--------+--------+--------+--+
```

2.left outer join（left join）

```
select
a.name as aname,
a.numb as anumb,
b.name as bname,
b.nick as bnick
from t_a a
left outer join t_b b
on a.name=b.name
```

结果：

```
+--------+--------+--------+--------+--+
| aname  | anumb  | bname  | bnick  |
+--------+--------+--------+--------+--+
| a      | 1      | a      | xx     |
| b      | 2      | b      | yy     |
| c      | 3      | NULL   | NULL   |
| d      | 4      | d      | zz     |
+--------+--------+--------+--------+--+
```

3.right outer join（right join）

```
select
a.name as aname,
a.numb as anumb,
b.name as bname,
b.nick as bnick
from t_a a
right outer join t_b b
on a.name=b.name
```

结果：

aname	anumb	bname	bnick
a	1	a	xx
b	2	b	yy
d	4	d	zz
NULL	NULL	e	pp

4.full outer join（full join）

```
select
a.name as aname,
a.numb as anumb,
b.name as bname,
b.nick as bnick
from t_a a
full join t_b b
on a.name=b.name;
```

结果：

aname	anumb	bname	bnick
a	1	a	xx
b	2	b	yy
c	3	NULL	NULL
d	4	d	zz
NULL	NULL	e	pp

6.4. left semi join

Left semi join ： 相当于*join*连接两个表后产生的数据中的左半部分
hive中不支持exist/IN子查询，可以用left semi join来实现同样的效果：

```
select
a.name as aname,
a.numb as anumb
from t_a a
left semi join t_b b
on a.name=b.name;
```

结果：

aname	anumb
a	1
b	2
d	4

注意： *left semi join*的 *select*子句中，不能有右表的字段

6.5. group by分组聚合

```
20170804,192.168.33.66,http://www.edu360.cn/job
20180804,192.168.33.40,http://www.edu360.cn/study
20180805,192.168.20.18,http://www.edu36.cn/job
20180805,192.168.20.28,http://www.edu36.cn/login
20180806,192.168.20.38,http://www.edu36.cn/job
20180806,192.168.20.38,http://www.edu36.cn/study
20180807,192.168.33.40,http://www.edu36.cn/login
20180807,192.168.20.88,http://www.edu36.cn/job
```

```
select dt,count(*),max(ip) as cnt from t_access group by dt;

select dt,count(*),max(ip) as cnt from t_access group by dt having dt>'20170804';

select
dt,count(*),max(ip) as cnt
from t_access
where url='http://www.edu360.cn/job'
group by dt having dt>'20170804';
```

注意： 一旦有group by子句，那么，在select子句中就不能有 （分组字段，聚合函数） 以外的字段

为什么where必须写在group by的前面，为什么group by后面的条件只能用having
因为，where是用于在真正执行查询逻辑之前过滤数据用的
having是对group by聚合之后的结果进行再过滤；

上述语句的执行逻辑：
*1.where*过滤不满足条件的数据
2.用聚合函数和*group by*进行数据运算聚合，得到聚合结果
*3.*用*having*条件过滤掉聚合结果中不满足条件的数据

6.6. 子查询

```
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
```

```
-- 查询有兄弟的人
select id,name,brother
from
(select id,name,family_members['brother'] as brother from t_person) tmp
where brother is not null;
```

另一种写法：
select id,name,family_members[‘brother’]
from t_person where array_contains(map_keys(family_members),”brother”);

7.hive函数使用

测试函数小技巧：
直接用常量来测试函数即可
select substr("abcdefg",1,3);
而且，可以将hive的本地运行自动模式开启：
hive>set hive.exec.mode.local.auto=true;

HIVE 的所有函数手册：
https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-Built-inTable-GeneratingFunctions(UDTF)
https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF

7.1. 常用内置函数

类型转select cast("5" as int) ;
select cast("2017-08-03" as date) ;
select cast(current_timestamp as date);

示例：

7.1.1.换函数

1	*1995-05-05 13:30:59*	*1200.3*
2	*1994-04-05 13:30:59*	*2200*
3	*1996-06-01 12:20:30*	*80000.5*

create table t_fun(id string,birthday string,salary string)
row format delimited fields terminated by ',';

select id,cast(birthday as date) as bir,cast(salary as float) from t_fun;

7.1.2.数学运算函数

select round(5.4); ## 5 四舍五入
select round(5.1345,3) ; ##5.135
select ceil(5.4) ; // select ceiling(5.4) ; ## 6 向上取整
select floor(5.4); ## 5 向下取整
select abs(-5.4) ; ## 5.4 绝对值
select greatest(id1,id2,id3) ; ## 6 单行函数
select least(3,5,6) ; ##求多个输入参数中的最小值

示例：
有表如下：

```
0: jdbc:hive2://hdp20-04:10000> select * from t_fun2;
+-------------+--------------+---------------------+-----------------+-----------------+--+
| t_fun2.id   | t_fun2.name  |    t_fun2.salay1    | t_fun2.salary2  | t_fun2.slary3   |
+-------------+--------------+---------------------+-----------------+-----------------+--+
| 1           | zhangsan     | 1000.7999877929688  | 2000.0          | 1800.0          |
| 2           | lisi         | 8000.7998046875     | 6000.0          | 9800.0          |
+-------------+--------------+---------------------+-----------------+-----------------+--+
```

select greatest(cast(s1 as double),cast(s2 as double),cast(s3 as double)) from t_fun2;
结果：

```
+---------+--+
|   _c0   |
+---------+--+
| 2000.0  |
| 9800.0  |
+---------+--+
```

select max(age) from t_person group by ..; 分组聚合函数
select min(age) from t_person group by...; 分组聚合函数

7.1.3.字符串函数

substr(string str, int start) ## 截取子串
substring(string str, int start)
示例：*select substr("abcdefg",2) ;*
substr(string, int start, int len)
substring(string, int start, int len)
示例：*select substr("abcdefg",2,3) ; ## bcd*

concat(string A, string B...) ## 拼接字符串
concat_ws(string SEP, string A, string B...)
示例：*select concat("ab","xy") ; ## abxy*
select concat_ws(".","192","168","33","44") ; ## 192.168.33.44

length(string A)
示例：*select length("192.168.33.44"); ## 13*

split(string str, string pat) ## 切分字符串，返回数组
示例：~~select split("192.168.33.44",".") ;~~ 错误的，因为.号是正则语法中的特定字符

select split("192.168.33.44","\\.") ;

upper(string str) ##转大写
lower(string str) ##转小写

7.1.4.时间函数

select current_timestamp; ## 返回值类型：timestamp，获取当前的时间戳(详细时间信息)
select current_date; ## 返回值类型：date，获取当前的日期

unix时间戳转字符串格式——**from_unixtime**
from_unixtime(bigint unixtime[, string format])
示例：select from_unixtime(unix_timestamp());
select from_unixtime(unix_timestamp(),"yyyy/MM/dd HH:mm:ss");

字符串格式转unix时间戳——**unix_timestamp：返回值是一个长整数类型**

如果不带参数，取当前时间的秒数时间戳long--(距离格林威治时间1970-1-1 0:0:0秒的差距)
select **unix_timestamp()**;

unix_timestamp(string date, string pattern)
示例： select unix_timestamp("2017-08-10 17:50:30");
select unix_timestamp("2017-08-10 17:50:30","yyyy-MM-dd HH:mm:ss");

将字符串转成日期date
select **to_date**("2017-09-17 16:58:32");

7.1.5.条件控制函数

7.1.5.1.IF

```
select id,if(age>25,'working','worked') from t_user;
select moive_name,if(array_contains(actors,'吴刚'),'好电影','烂片儿')
from t_movie;
```

7.1.5.2.case when （单行的函数）

语法：
CASE [expression]

```
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    ...
    WHEN conditionn THEN resultn
    ELSE result
END
```

示例：

```
1,zhangsan,18,beijing,20000
2,lisi,28,shanghai,1500
3,wangwu,38,wuhan,4000
4,zhaoliu,35,changsha,10050
5,tianqi,23,shijiazhuang,30000
6,wangba,45,qinhuangdao,28000
7,wushuai,55,haerbin,80000
```

查询：

1,zhangsan,少年,beijing,中产

```
建表：
create table t10(id int,name string,age int,addr string,income int)
row format delimited fields terminated by ',';

查询：
select id,name,
case
when age<=20 then '少年'
when age>20 and age<=40 then '青年'
when age>40 and age<=60 then '中年'
when age>60 then '老年'
end as status,
addr,
income,
case
when income<=5000 then '穷屌丝'
when income>5000 and income<=15000 then '富屌丝'
when income>15000 and income<=30000 then '还行'
when income>30000 and income<=60000 then '涛哥'
when income>60000 then '土豪'
end as level
from t10;
```

结果：

```
+-----+----------+--------+-------------+--------+-------+--+
| id  |   name   | status |     addr    | income | level |
+-----+----------+--------+-------------+--------+-------+--+
| 1   | zhangsan | 少年     | beijing     | 20000  | 还行    |
| 2   | lisi     | 青年     | shanghai    | 1500   | 穷屌丝   |
| 3   | wangwu   | 青年     | wuhan       | 4000   | 穷屌丝   |
| 4   | zhaoliu  | 青年     | changsha    | 10050  | 富屌丝   |
| 5   | tianqi   | 青年     | shijiazhuang | 30000  | 还行    |
| 6   | wangba   | 中年     | qinhuangdao | 28000  | 还行    |
| 7   | wushuai  | 中年     | haerbin     | 80000  | 土豪    |
+-----+----------+--------+-------------+--------+-------+--+
```

7.1.6.集合函数

array(3,5,8,9) 构造一个整数数组

array(‘hello’,’moto’,’semense’,’chuizi’,’xiaolajiao’) 构造一个字符串数组

array_contains(Array<T>, value) 返回boolean值

示例：
select moive_name,array_contains(actors,'吴刚') from t_movie;
select array_contains(array('a','b','c'),'c') ;

sort_array(Array<T>) 返回排序后的数组

示例：
select sort_array(array('c','b','a')) ;
select 'haha',sort_array(array('c','b','a')) as xx from (select 0) tmp;

size(Array<T>) 返回一个集合的长度，int值

示例：
select moive_name,size(actors) as actor_number from t_movie;

size(Map<K.V>) 返回一个imap的元素个数，int值
size(array<T>) 返回一个数组的长度,int值

map_keys(Map<K.V>) 返回一个map字段的所有key，结果类型为：数组
map_values(Map<K.V>) 返回一个map字段的所有value，结果类型为：数组

7.1.7.常见分组聚合函数

sum(字段) ： 求这个字段在一个组中的所有值的和
avg(字段) ： 求这个字段在一个组中的所有值的平均值
max(字段) ： 求这个字段在一个组中的所有值的最大值
min(字段) ： 求这个字段在一个组中的所有值的最小值

分组聚合函数练习：

聚合函数练习：

原始数据：
1,zhangsan,18:male:北京
2,lisi,28:female:上海
3,wangwu,30:male:天津
4,zhaoliu,32:female:石家庄

建表语句：
```
create table t_user_info(id int,name string,info struct<age:int,gender:string,addr:string>)
row format delimited fields terminated by ','
collection items terminated by ':';
```

1.求男性及女性分别的年龄总和

```
select info.gender,sum(info.age)
from t_user_info
group by info.gender;
```

```
select gender,sum(age)
from
(select id,
name,
info,
info.age as age,
info.gender as gender
from t_user_info) tmp
group by tmp.gender
;
```

2.求男性及女性分别的最大年龄

```
select info.gender,max(info.age)
from t_user_info
group by info.gender;
```

3.求男性及女性分别的最小年龄
```
select info.gender,min(info.age)
from t_user_info
group by info.gender;
```

4.求男性及女性分别的平均年龄
```
select info.gender,avg(info.age)
from t_user_info
group by info.gender;
```

5.求所有人的信息，及男、女性别各自的平均年龄

count()：求一个组中的满足某条件的数据条数！

举例说明：

1,mary,female,jiuye
2,tom,male,meijiuye
3,kitty,female,meijiuye
4,white,male,jiuye
5,jack,male,jiuye
6,rose,female,meijiuye

建表语句：

```
create table t_count(id int,name string,gender string,job string)
row format delimited fields terminated by ‘,’;
```

请求出，男生和女生中分别已就业的人数

方式1 ： 在count计数时进行判断是否需要计入

```
select sex,count(if(job='jiuye',1,null))
from  t11
group by sex;
```

方式3：在count计数时判断是否需要计入：

```
select sex,count(case when job='jiuye' then 1 else null end)
from  t11
group by sex;
```

方式3：先过滤掉不需要计入的数据，再分组计数

```
select sex,count(1)
from
(
select *
from t11
where job='jiuye') o1
group by o1.sex;
```

collect_set() :将某个字段在一组中的所有值形成一个集合（数组）返回

举例：

```
1,zhangsan,数学
1,zhangsan,化学
1,zhangsan,语文
1,zhangsan,搭讪学
2,lisi,数学
2,lisi,化学
2,lisi,聊骚
2,lisi,成人搏斗学
3,wangwu,防狼术
3,wangwu,跆拳道
```

```
create table t13(id int,name string,subject string)
row format delimited fields terminated by ‘,’;
```

需要查询出如下结果：

1	zhangsan	数学 化学 语文 搭讪学
2	lisi	数学 化学 聊骚
.....	....	....

```
select id,name,collect_set(subject)
from t13
group by id,name;
```

```
+-----+-----------+-------------------------------------------------------------+--+
| id  |   name    |                          _c2                                |
+-----+-----------+-------------------------------------------------------------+--+
| 1   | zhangsan  | ["数学","化学","语文","街头搭讪学"]                           |
| 2   | lisi      | ["数学","化学","聊骚","成人搏斗学"]                           |
| 3   | wangwu    | ["防狼术","跆拳道"]                                           |
+-----+-----------+-------------------------------------------------------------+--+
```

disctinct

数据：

```
1,zhangsan,28,beijing
1,lisi,29,shanghai
1,wangwu,30,beijing
2,zhaoliu,18,tianjin
2,tianqi,19,shenzhen
3,wangba,21,shenzhen
```

建表：

```
create table t_distinct(id int,
name string,
age int,
addr string)
row format delimited fields terminated by ',';


load data local inpath '/root/hivedata/distinct.dat' into table t_distinct;
```

7.1.8.表生成函数

7.1.8.1.行转列函数：explode()

假如有以下数据：

```
1,zhangsan,化学:物理:数学:语文
2,lisi,化学:数学:生物:生理:卫生
3,wangwu,化学:语文:英语:体育:生物
```

映射成一张表：

```
create table t_stu_subject(id int,name string,subjects array<string>)
row format delimited fields terminated by ','
collection items terminated by ':';

load data local inpath '/root/sb.dat' into table t_stu_subject;
```

使用explode()对数组字段“炸裂”

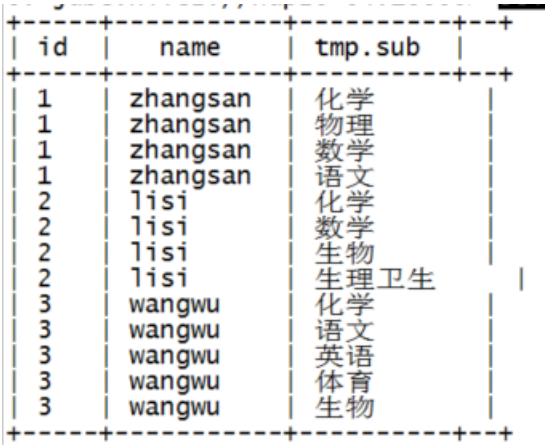
```
0: jdbc:hive2://hdp20-04:10000> select * from t_stu_subject;
+-------------------+---------------------+-----------------------------+--+
| t_stu_subject.id  | t_stu_subject.name  |   t_stu_subject.subjects    |
+-------------------+---------------------+-----------------------------+--+
| 1                 | zhangsan            | ["化学","物理","数学","语文"]       |
| 2                 | lisi                | ["化学","数学","生物","生理卫生"]     |
| 3                 | wangwu              | ["化学","语文","英语","体育","生物"]  |
+-------------------+---------------------+-----------------------------+--+
3 rows selected (0.095 seconds)
0: jdbc:hive2://hdp20-04:10000> select explode(subjects) from t_stu_subject;
+-------+--+
|  col  |
+-------+--+
| 化学    |
| 物理    |
| 数学    |
| 语文    |
| 化学    |
| 数学    |
| 生物    |
| 生理卫生  |
| 化学    |
| 语文    |
| 英语    |
| 体育    |
| 生物    |
+-------+--+
```

然后，我们利用这个explode的结果，来求去重的课程：

```
select distinct tmp.sub
from
(select explode(subjects) as sub from t_stu_subject) tmp;
```

7.1.8.2.表生成函数lateral view

```
select id,name,tmp.sub
from t_stu_subject lateral view explode(subjects) tmp as sub;
```



id	name	tmp.sub
1	zhangsan	化学
1	zhangsan	物理
1	zhangsan	数学
1	zhangsan	语文
2	lisi	化学
2	lisi	数学
2	lisi	生物
2	lisi	生理卫生
3	wangwu	化学
3	wangwu	语文
3	wangwu	英语
3	wangwu	体育
3	wangwu	生物

理解： *lateral view* 相当于两个表在*join*
左表：是原表
右表：是*explode(*某个集合字段*)*之后产生的表
而且：这个*join*只在同一行的数据间进行

那样，可以方便做更多的查询：
比如，查询选修了生物课的同学
select a.id,a.name,a.sub from
(select id,name,tmp.sub as sub from t_stu_subject lateral view explode(subjects) tmp as sub) a
where sub='生物';

7.1.9.json解析函数：表生成函数

需求：有如下json格式的电影评分数据：

```
{"movie":"1193","rate":"5","timeStamp":"978300760","uid":"1"}
{"movie":"661","rate":"3","timeStamp":"978302109","uid":"1"}
{"movie":"914","rate":"3","timeStamp":"978301968","uid":"1"}
{"movie":"3408","rate":"4","timeStamp":"978300275","uid":"1"}
{"movie":"2355","rate":"5","timeStamp":"978824291","uid":"1"}
{"movie":"1197","rate":"3","timeStamp":"978302268","uid":"1"}
```

需要做各种统计分析。
发现，直接对json做sql查询不方便，需要将json数据解析成普通的结构化数据表。可以采用hive中内置的json_tuple()函数

实现步骤：
1.创建一个原始表用来对应原始的json数据
create table t_json(json string);
load data local inpath ‘/root/rating.json’ into table t_json;

2.利用**json_tuple**进行json数据解析
测试，示例：
select json_tuple(json,'movie','rate','timeStamp','uid') as(movie,rate,ts,uid) from t_json limit 10;

产生结果：

movie	rate	ts	uid	
1193	5	978300760	1	
661	3	978302109	1	
914	3	978301968	1	
3408	4	978300275	1	
2355	5	978824291	1	
1197	3	978302268	1	
1287	5	978302039	1	
2804	5	978300719	1	
594	4	978302268	1	
919	4	978301368	1	

真正解析整张json表，将解析结果数据插入一张新表

```
create table t_movie_rate
as
select json_tuple(json,'movie','rate','timeStamp','uid') as(movie,rate,ts,uid) from t_json;
```

-- TODO 练习--

1/ 统计每部电影的平均得分

```
create table t_rate_m_avg as select movie,avg(rate) from t_rate group by movie;
```

2/ 统计数据中一共有多少部电影

```
select count(distinct movie) from t_rate;
```

3/ 统计每部电影的被评分次数（热门度）

```
create table t_rate_m_cnt as select movie,count(1) as cnt from t_rate group by movie order by cnt desc;
```

4/ 统计每个人的所有电影评分的平均值

```
create table t_rate_u_avg as select uid,avg(rate) from t_rate group by uid;
```

5/ 查询出每个人评分最高的20部电影信息

利用json_tuple从原始json数据表中，etl出一个详细信息表：

```
create table t_rate
as
select
uid,
movie,
rate,
year(from_unixtime(cast(ts as bigint))) as year,
month(from_unixtime(cast(ts as bigint))) as month,
day(from_unixtime(cast(ts as bigint))) as day,
hour(from_unixtime(cast(ts as bigint))) as hour,
minute(from_unixtime(cast(ts as bigint))) as minute,
from_unixtime(cast(ts as bigint)) as ts
from
(select
json_tuple(rateinfo,'movie','rate','timeStamp','uid') as(movie,rate,ts,uid)
from t_json) tmp
;
```

7.1.10.窗口分析函数：

7.1.10.1.row_number() over()——分组TOPN

7.1.10.1.1.需求

有如下数据：

```
1,18,a,male
2,19,b,male
3,22,c,female
4,16,d,female
5,30,e,renyao
6,26,f,female
7,16,d,renyao
8,30,e,renyao
9,26,f,female
10,14,b,male
```

需要查询出每种性别中年龄最大的2条数据
create table t_rn(id int,age int,name string,sex string)
row format delimited fields terminated by ‘,’;

7.1.10.1.2.实现：

使用row_number函数，对表中的数据按照性别分组，按照年龄倒序排序并进行标记

hql代码：

```
select id,age,name,sex,
row_number() over(partition by sex order by age desc) as rank
from t_rownumber
```

产生结果：

id	age	name	sex	rank
6	26	f	female	1
3	22	c	female	2
4	16	d	female	3
5	30	e	male	1
2	19	b	male	2
1	18	a	male	3

然后，利用上面的结果，查询出rank<=2的即为最终需求

```
select id,age,name,sex
```

```
from
(select id,age,name,sex,
row_number() over(partition by sex order by age desc) as rank
from t_rownumber) tmp
where rank<=2;
```

练习：求出电影评分数据中，每个用户评分最高的*topn*条数据

```
select movie, rate, ts, uid from
(select movie,rate,ts,uid,
row_number() over(partition by uid order by rate desc) as rank from t_json_rate) tmp
where rank <= 3;
```

7.1.10.2.sum() over()——级联求和

需求：同8.2.1
(共享单车：本月新增用户数以及累积到本月的新增用户数...
本月充值额以及累积到本月的充值额....
本月用户骑行总时长以及累计到本月的骑行总时长.....
.....
)

实现：使用sum()over()窗口分析函数来实现

步骤1：先求出每家店铺每个月的总金额

```
create table t_tmp as
select name,month,sum(sale) as amt from shop group by name,month;
+-------+--------+-------+--+
| name  | month  |  amt  |
+-------+--------+-------+--+
| a     | 01     | 350   |
| a     | 02     | 5000  |
| a     | 03     | 600   |
| b     | 01     | 7800  |
| b     | 02     | 2500  |
| c     | 01     | 470   |
| c     | 02     | 630   |
+-------+--------+-------+--+
```

步骤2：实现级联求和

```
select name,month,amt,
sum(amt) over(partition by name order by month rows between unbounded preceding
and current row) as accumulate
from t_tmp;

drop table t_tmp;
```

重要提示	sum()over()的累加范围指定语法： *sum() over(partition by x order by y rows between 8 preceding and current row)* *sum() over(partition by x order by y rows between 8 preceding and 5 following)* *sum() over(partition by x order by y rows between unbounded preceding and 5 following)* *sum() over(partition by x order by y rows between unbounded preceding and unbounded following)*

7.1.10.3.窗口分析函数综合应用案例

有以下数据：

```
a,2017-02-05,200
a,2017-02-06,300
a,2017-02-07,200
a,2017-02-08,400
a,2017-02-10,600
b,2017-02-05,200
b,2017-02-06,300
b,2017-02-08,200
b,2017-02-09,400
b,2017-02-10,600
c,2017-01-31,200
c,2017-02-01,300
c,2017-02-02,200
c,2017-02-03,400
c,2017-02-10,600
a,2017-03-01,200
a,2017-03-02,300
a,2017-03-03,200
a,2017-03-04,400
a,2017-03-05,600
```

请找出那些有过连续4天都有销售记录的店铺；
（需求还可以演化成：有过连续10天登录app的用户；
有过连续10天扫码骑车的用户；...
有过连续观看公司推送的广告的用户；
有过连续n月充值的用户统计.....
有过连续N天登录本公司网站的用户统计.....
)
结果数据：

```
b
d
```

答案sql：
步骤1：先按店铺分窗口打rownumber

```
select shop,s_date,amt,
```

```
row_number() over(partition by shop order by s_date) as rn
from t18;
```

```
+-------+------------+------+-----+--+
| shop  |   s_date   | amt  | rn  |
+-------+------------+------+-----+--+
| a     | 2017-02-05 | 200  | 1   |
| a     | 2017-02-06 | 300  | 2   |
| a     | 2017-02-07 | 200  | 3   |
| a     | 2017-02-08 | 400  | 4   |
| a     | 2017-02-10 | 600  | 5   |
| b     | 2017-02-05 | 200  | 1   |
| b     | 2017-02-06 | 300  | 2   |
| b     | 2017-02-08 | 200  | 3   |
| b     | 2017-02-09 | 400  | 4   |
| b     | 2017-02-10 | 600  | 5   |
| c     | 2017-01-31 | 200  | 1   |
| c     | 2017-02-01 | 300  | 2   |
| c     | 2017-02-02 | 200  | 3   |
| c     | 2017-02-03 | 400  | 4   |
| c     | 2017-02-10 | 600  | 5   |
+-------+------------+------+-----+--+
```

然后，用 日期-rownumber得到一个差值
Linux中，对日期的减法；date -d '-1 day' "+%Y%m%d"

```
select shop,s_date,amt,date_sub(s_date,rn) as diff
from
(
select shop,s_date,amt,
row_number() over(partition by shop order by s_date) as rn
from t18) o1;
```

```
+-------+------------+------+------------+--+
| shop  |   s_date   | amt  |    diff    |
+-------+------------+------+------------+--+
| a     | 2017-02-05 | 200  | 2017-02-04 |
| a     | 2017-02-06 | 300  | 2017-02-04 |
| a     | 2017-02-07 | 200  | 2017-02-04 |
| a     | 2017-02-08 | 400  | 2017-02-04 |
| a     | 2017-02-10 | 600  | 2017-02-05 |
| b     | 2017-02-05 | 200  | 2017-02-04 |
| b     | 2017-02-06 | 300  | 2017-02-04 |
| b     | 2017-02-08 | 200  | 2017-02-05 |
| b     | 2017-02-09 | 400  | 2017-02-05 |
| b     | 2017-02-10 | 600  | 2017-02-05 |
| c     | 2017-01-31 | 200  | 2017-01-30 |
| c     | 2017-02-01 | 300  | 2017-01-30 |
| c     | 2017-02-02 | 200  | 2017-01-30 |
| c     | 2017-02-03 | 400  | 2017-01-30 |
| c     | 2017-02-10 | 600  | 2017-02-05 |
+-------+------------+------+------------+--+
```

按店铺,差值　分组计数，并过滤出技术值>=4的数据，并对店铺去重，就是那些有过连续4天以上销售记录的店铺

```
select distinct shop
from
(
select shop,count(1) as cnt
from
(
select shop,s_date,amt,date_sub(s_date,rn) as diff
from
(
select shop,s_date,amt,
row_number() over(partition by shop order by s_date) as rn
from t18) o1) o2

group by shop,diff

having cnt>=4
) o3
;
```

```
+-------+--+
| shop  |
+-------+--+
| a     |
| c     |
+-------+--+
```

7.2. 自定义函数

7.2.1.需求1：

有如下数据：

```
id,salay,bonus,subsidy
a,100,50,120
b,220,150,20
c,220,450,220
```

3个字段分别表示： 用户id，基本工资，业绩提成，股权收益
需要查询出每个人的三类收益中最高的是哪一种收益

```
a,100,50,120,subsidy
b,220,150,20,salary
c,220,450,220,bonus
```

hive中有一个函数greatest(f1,f2,f3)可以求出n个字段中的最大值，满足不了本案例的需

求。此时，我们可以考虑自己开发一个hive的函数（hive具备这个机制）
create table t19(id int, salary int,bonus int,subsidy int)
row format delimited fields terminated by ‘,’;

通过需求分析，可得，如果有如下函数
select id,highest_income_type(salary,bonus,subsidy) from t19

highest_income_type(salary,bonus,subsidy) :能够接收3个整数，返回哪一个是最大的
就很容易实现需求；

可惜这样的函数在hive中没有；
解决：可以自定义一个这样的函数；

7.2.2.实现思路：

hive的函数无非也就是一个逻辑的封装，可以接收参数，返回结果，跟java中的方法本质上没有区别

hive就允许用户开发一个java的方法，来实现你想要的函数的功能；
然后在hive中定义一个自己命名的函数，并将这个函数跟你的java方法所在的类关联起来即可。

7.2.3.实现步骤：

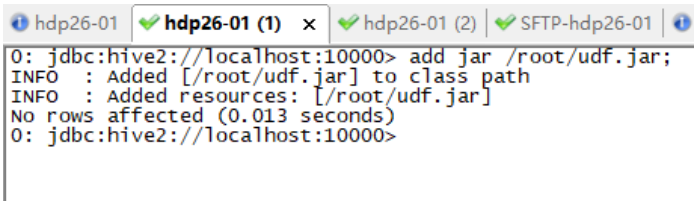
1/ 开发一个java类继承（HIVE的父类UDF），重载一个方法： evaluate()
方法的功能：
    输入：3个整数值
    返回：最大值所在的序号

```
public class HighestIncomType extends UDF{
    // 重载evaluate方法，而且必须是public
    public int evaluate(int a,int b,int c) {
        // 求出最大值是几
        int max = a;
        if(a<b) {
            max = b;
        }
        if(max<c) {
            max = c;
        }


        //判断谁是最大值
        if(a==max) {
            return 0;
        }
        if(b==max) {
            return 1;
        }else{
            return 2;
        }
    }
}
```

2/ 将java工程打成jar包，上传到hive所在的机器上

3/ 在hive的提示符中，将jar包添加到hive的运行时classpath



4/ 在hive的提示中，用hive语法声明一个自定义函数，并与jar包中的java类关联

```
0:jdbc:hive2://localhost:10000> create temporary function highest_income_type as
'top.ganhoo.hive.udf.HighestIncomType';
No rows affected (0.016 seconds)
```

5/ 就可以在sql查询中使用这个函数了

```
a,100,50,120
b,220,150,20
c,220,450,220
```

使用函数实现需求：

```
select id,
case
when type=0 then '基本工资'
when type=1 then '股权收益'
else '津贴'
end as type
from
(select id,highest_income_type(salary,bonus,subsidy) as type from t19) o1;
```

```
+------+---------+----------+---------+------+--+
| uid  | salary  | ticheng  | guquan  | idx  |
+------+---------+----------+---------+------+--+
| a    | 100     | 50       | 120     | 2    |
| b    | 220     | 150      | 20      | 0    |
| c    | 220     | 450      | 220     | 1    |
+------+---------+----------+---------+------+--+
```

用case-when改改显示值：

```
select uid,salary,ticheng,guquan,
case
when get_max_index(salary,ticheng,guquan) =0 then '基本工资'
when get_max_index(salary,ticheng,guquan) =1 then '业绩提成'
when get_max_index(salary,ticheng,guquan) =2 then '股权收益'
end
from t_employee;
```

```
+------+---------+----------+---------+-------+--+
| uid  | salary  | ticheng  | guquan  |  _c4  |
+------+---------+----------+---------+-------+--+
| a    | 100     | 50       | 120     | 股权收益  |
| b    | 220     | 150      | 20      | 基本工资  |
| c    | 220     | 450      | 220     | 业绩提成  |
+------+---------+----------+---------+-------+--+
```

7.2.4.需求2：

需要对json数据表中的json数据写一个自定义函数，用于传入一个json，返回一个数据值的数组

json原始数据表：

```
0: jdbc:hive2://hdp20-04:10000> select * from t_rating_json limit 2;
+-----------------------------------------------------------------+--+
|                        t_rating_json.json                       |
+-----------------------------------------------------------------+--+
| {"movie":"1193","rate":"5","timeStamp":"978300760","uid":"1"}   |
| {"movie":"661","rate":"3","timeStamp":"978302109","uid":"1"}    |
+-----------------------------------------------------------------+--+
```

需要做ETL操作，将json数据变成普通表数据，插入另一个表中：

```
0: jdbc:hive2://hdp20-04:10000> desc t_rate;
+-----------+------------+----------+--+
| col_name  | data_type  | comment  |
+-----------+------------+----------+--+
| movie     | string     |          |
| rate      | int        |          |
| ts        | bigint     |          |
| uid       | int        |          |
+-----------+------------+----------+--+
```

7.2.5.实现步骤：

1、开发JAVA的UDF类

```
public class ParseJson extends UDF{

    // 重载 ：返回值类型 和参数类型及个数，完全由用户自己决定
    // 本处需求是：给一个字符串，返回一个数组
    public String[] evaluate(String json) {

        String[] split = json.split("\"");
        String[] res = new String[]{split[3],split[7],split[11],split[15]};
        return res;
    }
}
```

2、打jar包

在eclipse中使用export即可

3.上传jar包到运行hive所在的linux机器

4.在hive中创建临时函数：
在hive的提示符中：
hive> add jar /root/jsonparse.jar;
然后，在hive的提示符中，创建一个临时函数：
hive>CREATE TEMPORARY FUNCTION jsonp AS 'cn.edu360.hdp.hive.ParseJson';

5.开发hql语句，利用自定义函数，从原始表中抽取数据插入新表

```
insert into table t_rate
select
split(jsonp(json),',')[0],
cast(split(jsonp(json),',')[1] as int),
cast(split(jsonp(json),',')[2] as bigint),
cast(split(jsonp(json),',')[3] as int)
from
t_rating_json;
```

注：临时函数只在一次*hive*会话中有效，重启会话后就无效

如果需要经常使用该自定义函数，可以考虑创建永久函数：
拷贝jar包到hive的类路径中：
cp wc.jar apps/hive-1.2.1/lib/
创建了：
create function pfuncx as 'com.doit.hive.udf.UserInfoParser';

删除函数：
DROP TEMPORARY FUNCTION [IF EXISTS] function_name
DROP FUNCTION[IF EXISTS] function_name

8. 综合查询案例

8.1. 用hql来做wordcount

有以下文本文件：

```
hello tom hello jim
hello rose hello tom
tom love rose rose love jim
jim love tom love is what
what is love
```

需要用hive做wordcount

```
-- 建表映射
create table t_wc(sentence string);

-- 导入数据
load data local inpath '/root/hivetest/xx.txt' into table t_wc;
```

hql答案：

```
SELECT word
    ,count(1) as cnts
FROM (
    SELECT explode(split(sentence, ' ')) AS word
    FROM t_wc
    ) tmp
GROUP BY word
order by cnts desc
;
```

8.2. 略难报表查询

8.2.1. 级联累计报表查询

有如下数据：

```
A,2015-01-08,5
A,2015-01-11,15
B,2015-01-12,5
A,2015-01-12,8
B,2015-01-13,25
A,2015-01-13,5
C,2015-01-09,10
C,2015-01-11,20
A,2015-02-10,4
A,2015-02-11,6
C,2015-01-12,30
C,2015-02-13,10
B,2015-02-10,10
B,2015-02-11,5
A,2015-03-20,14
A,2015-03-21,6
B,2015-03-11,20
B,2015-03-12,25
C,2015-03-10,10
C,2015-03-11,20
```

建表映射：

```
create table t_access_times(username string,month string,counts int)
row format delimited fields terminated by ',';
```

需要要开发hql脚本，来统计出如下累计报表：

店铺	月份	月总额	累计到当月的总额
A	2015-01	33	33
A	2015-02	10	43
A	2015-03	30	73
B	2015-01	30	30
B	2015-02	15	45

8.2.1.1.实现方式1：

——利用自jion

答案： 参见《复杂*sql*案例--套路--累计报表.*txt*》

8.2.1.2.实现方式2

——利用sum()over()窗口分析函数：

```
select shopid,month,amount,
sum(amount) over(partition by shopid order by month rows between unbounded preceding and current row ) as accumulate_amount

from (
select shopid,substr(dt,1,7) as month,sum(sale) as amount
from t_jd group by shopid,substr(dt,1,7)
) tmp;
```

8.2.2.连续销售记录查询

请查出整个数据集中，那些有过连续3天销售记录的店；

```
A,2017-10-11,300
A,2017-10-12,200
B,2017-10-11,400
B,2017-10-12,200
A,2017-10-13,100
B,2017-10-15,600
A,2017-10-15,100
C,2017-10-11,350
A,2017-10-16,300
C,2017-10-13,250
A,2017-10-17,150
C,2017-10-14,300
C,2017-10-15,400
A,2017-10-18,340
A,2017-10-19,360
C,2017-10-16,200
D,2017-10-13,500
E,2017-10-14,600
E,2017-10-15,500
D,2017-10-14,600
```

```
create  table t_continue (id string,sale_d date,sale int)
row format delimited fields terminated by ‘,’;
```

思路：让连续的日期行，都附带一个相同的差值

```
id ，day   ,sale,rn,diff
A,2017-10-11,300,1,2017-10-10
A,2017-10-12,200,2,2017-10-10
A,2017-10-13,100,3,2017-10-10
A,2017-10-15,100,4,2017-10-11
A,2017-10-16,300,5,2017-10-11
A,2017-10-17,150,6,2017-10-11
A,2017-10-18,340,7,2017-10-11
A,2017-10-19,360,8,2017-10-11
```

最终答案：

```
SELECT DISTINCT shopid
FROM (
	SELECT shopid
		,count(1) AS lianxu
	FROM (
		SELECT shopid
			,day
			,sale
			,date_sub(day, rn) AS diff
		FROM (
			SELECT shopid
				,day
				,sale
				,row_number() OVER (
					PARTITION BY shopid ORDER BY day
					) AS rn
			FROM t_jd_2
			) tmp
		) tmp2
	GROUP BY shopid
		,diff
	HAVING lianxu >= 3
	) tmp3;
```

9.数据ETL综合案例

需求：联想集团有一款app产品叫茄子快传（有上亿的活跃用户，集中在第三世界国家）
现在需要开发一个数据分析系统，来对app的用户行为数据做各类分析；
预处理需求（mapreduce）：
1/ 请对app事件请求日志进行预处理：
a)过滤掉一些不合法数据（缺失device_id，app_ver_name,os_name，app_token，city,release_channel字段需要过滤）
b)将原格式json，解析成csv（逗号分隔的文本）格式，并去掉"events"字段
c)在原始数据中，追加一个字段user_id（如果是苹果，就用device_id,如果是android，就用android_id）

数据分析（挖掘）需求：
2/ 对预处理后的数据建模（建表(分区表)），并将数据加载到hive仓库
3/ 分析每日活跃用户指标、和新增用户指标
事实：活跃用户（当天出现的用户，当天的活跃用户）
维度：时间，城市，渠道，版本

事实：新用户（之前从来没出现过，但今天出现了，就是今天的新用户）
维度：时间，城市，渠道，版本

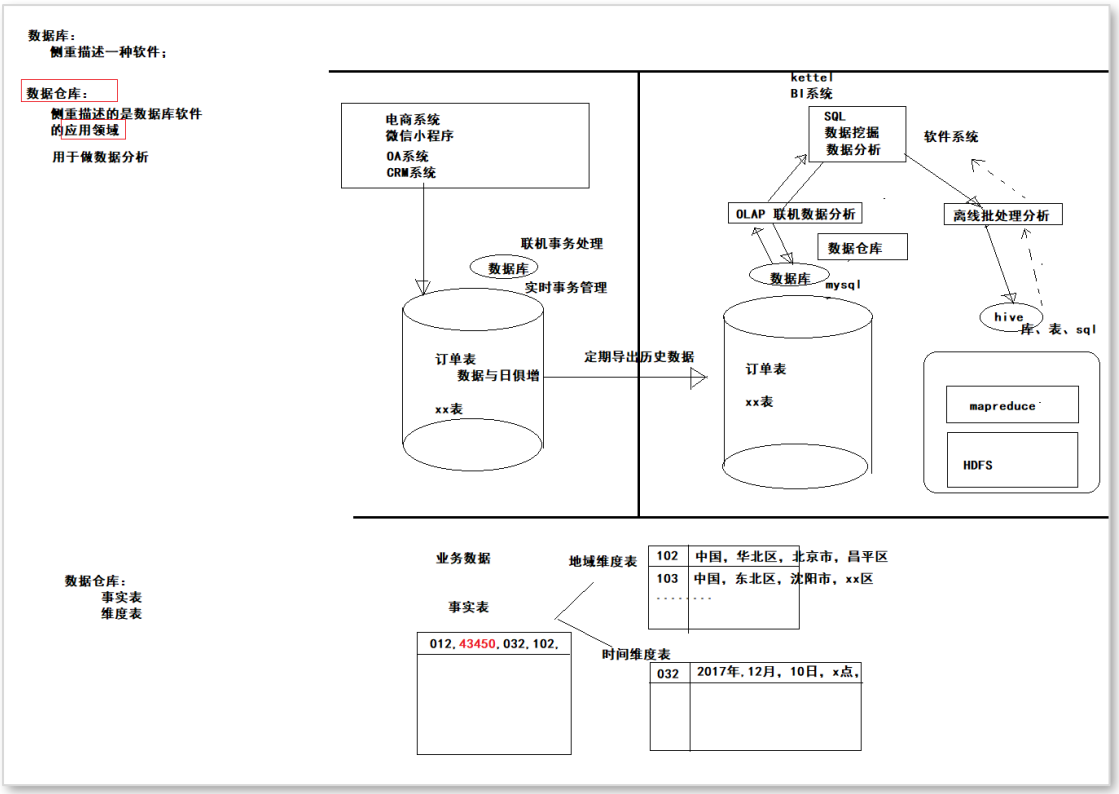
分步骤：
a)先抽取出当日的活跃用户信息
x,时间,城市,渠道,版本,.....
y,时间,城市,渠道,版本,.....
z,时间,城市,渠道,版本,.....

b)再进行各维度数据统计

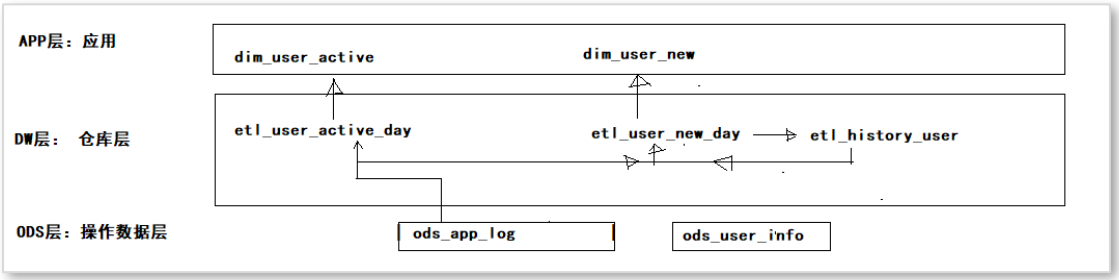
时间	app版本	城市	渠道	活跃用户数
2017-11-01	3.2	北京	1	6000
2017-11-01	3.2	北京	2	5000
			……	
2017-11-01	3.2	上海	1	7000
2017-11-01	3.2	上海	2	4000
			……	
2017-11-01	3.3	北京	1	5000

补充数据仓库一些专业术语：

数据仓库基本概念介绍：



数据仓库中的表分层模型：



数据仓库建模的经典模型：（课后扩展）

星型模型（大数据技术）：

雪花模型（传统关系型数据库）：

三范式模型（传统关系型数据库）：